LOG8415E - Assignment 1

Marie-Lee Brault (2050165)
Philippe Côté-Morneault (1905783)
Samuel Saito-Gagné (1902565)

Friday, October 9th, 2020

1 Introduction

1.1 Context

This document reports the analysis executed in order to assist a company to define the architecture of a cloud-based application. The company wants to be able to store historical data, then extract data from those storage to load them into a clustered database before visualizing the data for BI projects. In order to propose to the company an architecture that fit these needs, we performed multiple benchmarks regarding disk speed, storage capacity, IO speed and costs on mulitple types of VM instances offered by Amazon.

1.2 VM Instances Types

As part of our research we analyzed different types of EC2 Ubuntu machines available on AWS Clouds: m4.large, c4.xlarge, c5.xlarge, m5.xlarge, t2.2xlarge and t2.xlarge. Table 1 summarizes the properties of each of these instances. AWS offers a wide range of instance types, but these instance types are grouped by what they specialize in. m4 and m5 instances are general-purpose instances. They offer a balanced amount of resources to cater to most developers. c4 and c5 instances are compute-oriented. These instances usually have more vCPUs, more powerful CPU cores and a higher EBS bandwidth. t2 instances are also general-purpose instances, but they offer a burstable limit to CPU resources. These instances accumulate credits when not making extensive CPU usage and will spend these credits when they need more CPU resources.

We mentionned EBS in the previous paragraph. EBS, also known as Elastic Block Store, is a distributed block storage service designed for performance. This is the storage that all our chosen instances are using. The bandwidth to the EBS is dictated by the instance type and cost customers are willing to be.

The following sections will describe which benchmark were used for each of the following components: IO, IOPS, CPU Memory Disk, Memory, Disk and Network Throughput.

Table 1: Characteristics of benchmarked instances

Instance Characteristics						
	m4.large	c4.xlarge	c5.xlarge	m5.xlarge	t2.2xlarge	t2.xlarge
CPU	Intel Xeon E5-2676 v3	Intel Xeon E5-2666 v3	Intel Xeon Scalable Processors	Intel Xeon Platinum 8175M	Intel Scalable Processor	Intel Scalable Processor
vCPU	2	4	4	4	8	4
Clock Speed (GHz)	2.4	2.9	3.4	3.1	2.3	2.3
Memory Size (GiB)	8	7.5	8	16	32	16
Storage	EBS, 450Mbps	EBS, 750Mbps	EBS, 4750Mbps	EBS, 4750Mbps	EBS	EBS

2 Chosen Benchmarking Tools

2.1 IO

The company requires good IO performance since it intends to perform data extraction and load data into a clustered database. Having an EC2 instance that can handle reading and writing from the disk quickly is an important factor for the company. The first metric we are going to look at is IO throughput. IO throughput is how much data can be read/written per a given time frame. Contrary to IO throughput, IOPS measures how many reads/writes can be performed per second. IO throughput involves writing a large file in chunks to another file. This involves reading from said input file then writing it to an output file, effectively testing read and write throughput.

A simple benchmarking tool for IO throughput is *dd*, a linux command to copy a file. As explained in the previous paragraph, to properly test IO throughput, we need to read from a given file and write to another. This tool will perform simple sequential IO operations and output how much data per second it was able to transfer. Usage of this tool will be discussed in upcoming sections, but on a high-level, we will evaluate the time taken to write a certain amount of data to a file and the time taken to read a certain amount of data from a file. This will give us write and read throughput of instances.

We chose this tool as it was lightweight, came with most Linux distributions and allowed us to easily benchmark read and write throughput.

2.2 IOPS

While the company needs good IO throughput in order to transfer large datasets, it also needs good IOPS performance. IOPS is the amount of IO operations an instance can perform per second. To benchmark this important metric of our instances, we used *bonnie++*. This tool allows us to perform many file read, write and delete operations and measure how many of these operations can be done per second. Also, the tool allows us to perform these operations on sequentially and randomly allocated files. We chose this tool as it appeared to be easy to use, gave us valuable metrics on instances and was easy to install.

To perform our benchmarks, we used the amounts of read, write and delete operations per second that can be performed on both sequentially and randomly allocated files. We also tweaked the parameters of *bonnie++* to create more files and bigger files in order to perform these tests.

2.3 CPU Memory Disk

A goal the company wants to accomplish is to do data visualization from the processed historical data. This is no easy tasks since the services need to do heavy computations in order to achieve a result. As such, the performance of the system's components such as its CPU, memory and disk is incredibly important. In addition, the nature of business intelligence is to be able to access the data quickly and efficiently. The system should not slow us down for these operations.

Phoronix Test Suite is a powerful tool that gives us access to many different and specialized benchmarks from OpenBenchmarking.org. As such, it is the perfect tool to give us a quick overview on the instances' performance regarding CPU, memory and disk. While Phoronix offers test suites specifically curated to benchmark these components, they had too many tests for the context of this assignment. As such, we decided to create our own test suite from a few simple, small-sized yet popular tests for each of these categories to give us a preview of each instance's performance.

To benchmark the CPU, we chose SciMark2's Composite Test. It is a simple benchmark to evaluate a processor's single-threaded performance in Mflops, or millions of floating point operations per second, running various computational tests. On OpenBenmarking.org, it is really popular (more than 250k downloads) and is supported on many different operating systems. This gave us good confidence in this benchmark. We specifically chose the composite test because of its popularity among SciMark2's users.

For the memory, RAMspeed was the chosen benchmark. RAMspeed measures the bandwidth of the cache and the memory. Similarly, we chose this benchmark

because of its popularity (more than 250k downloads), but also because it seems to be maintained as it was updated in April 2020. It offers different operations to benchmark the memory, but we specifically chose the copy on integers, as it is a simple reliable test. Particularly, "copy" transfers the data (integers) from one part of the memory to another.

Finally, Flexible IO Tester (fio) was chosen to benchmark the disk. Among the recommended tests for the disk, this was one of the most efficient ones, but also the most popular one (more than 300k downloads). This benchmark offers many possibilities for IO-testing, but we chose only to evaluate with "random read" on 4 kB block size to give us an overview of the disk performance, as detailed IO and IOPS benchmarking will be done.

2.4 Memory

The company wants to build a cloud-based application that requires data extraction. System memory helps to reduce the time needed for processes to access information. Sysbench and stress-ng are two of the most popular tools to benchmark RAM performance. Memory performance are measured according to throughput and performance.
Sysbench is one of the most popular benchmark tool especially for MySQL databases. It is a very simple tool that offers various kind of benchmarks to test the hardware. It enables the user to generate different data distribution in order to stress them. For example, memory-intensive program usually access data that have uniform distribution, while databases with hot-spot is less disk-intensive since most of the data can be kept in the cache. We chose this tool since it has a general purpose use, so it is simpler to use.

2.5 Disk

The company wans to be able to keep historical data on an EC2 instance which needs to be storage optimized. Disk performance can be measured according to its IOPS, disk throughput and latency. Since IOPS is already covered in a previous section, we benchmarked the disk performance according to throughput and latency. The disk throughput measures the number of bits read or written per second and is a good indication of disk performance. The latency measures the delay of the process of reading or writing.

We chose to use the dbench utility tool to benchmark the disk because it offers a comprehensive but simple benchmark, which is good for the purpose of this context. It is a synthethic benchmark that can measure disk performance by simulating Netbench on Linux, the standard for Windows file servers in the industry. dbench does not use as much hardware resources as Netbench and is free. Hence, we chose dbench since it was a powerful tool that is a free version of an industry-standard benchmark.

hdparm is another useful and popular tool in the industry. We used hdparm to complement this analysis to assess the hard drive speed of the instance regardless of the file system used. It gives a basic low-level disk performance.

2.6 Network Throughput

As the retailer company's application handles large amounts of data, it needs to have data pipelines to transfer them to all the application's components. As such, it is really important for the system to have a good network throughput to be able to communicate this data quickly and reliably.

For the benchmark, we chose SpeedTest as it had a tool usable from the command line, which is really useful for automation. It is also a tool we already used to test our network connection, so it was already familiar to us. Besides from the result formatting, no parameters were really needed, which makes it really simple to use.

3 Benchmarking Issues

3.1 IO

Choosing the right parameters for the *dd* tool has proven to be a challenge. We first searched for sources that suggested how to proceed to benchmark IO throughput online. Many sources had conflicted opinions about which parameters should be chosen. Choosing the *bs* and *count* parameters was proving to be a slight challenge for us. Some sources suggested to use a count of *1k* while others claimed it did not matter. What most sources seemed to agree on is that we should take a combined file size large enough that the file copying would take at least a few seconds to execute. The combined file size is the amount of chunks you send times the block size you send, making it *bs * count*.

We decided to take a combined file size of 50GB as a starting point. Unfortunately, this would not work out of the box. Amazon EC2 enforces a soft limit on the EBS size of 10GB. Running *dd* with a combined file size larger than 10GB would simply fail as the instance could not get more storage. To remedy to this problem, we would have had to change some configuration on our instances to remove this limit. We decided not to change the instances and instead lower the combined file size to 5GiB. Our philosophy for these benchmarks was to be as lightweight as possible and to avoid unnecessary configuration. Lowering our combined file size would still properly benchmark our instances. Since we had an upper bound of 10GB and that the root folder only had about 7 GiB of free space on our instances, we found that 5GiB would be a safe number to choose. We tried using 7GiB at first, but on some instances we ran out of space.

When we first ran the read tests with *dd*, we had some unexpectedly high results. Our read speeds were 2-5 times as fast as our write times. After doing research online, we found out that we had to flush the cache before performing the read test. At first, we were performing the write test and immediately performing the read test. This gave us bad results since most of the read data was already cached. Flushing the cache with *echo 3 > /proc/sys/vm/dropcaches* before performing the read test gave us results that were much more realistic.

3.2 IOPS

Similarly to *dd*, properly using *bonnie++* came with challenges. The first challenges were to properly understand what tests were performed by *bonnie++*, what parameters to change and how to interpret the output. After doing extensive research, we had understood what exactly *bonnie++* was testing and what parameters we could use. *bonnie++* can be used to test read and write throughput, but we had already done that with *dd*. We disabled these tests and instead used file creation and deletion tests. These tests actually benchmarked IOPS. The difficulty was choosing the *n* parameter. At first, we did not change this parameter and we were not getting results from *bonnie++*. The tests were being executed to quickly and we could not get actual values from testing. By testing a few values manually, we found a good parameter that would not use too much disk space and that would allow us to see the results.

We also had issues running *bonnie++* due to a misunderstanding of a parameter. When we first ran *bonnie++*, we were getting the following error message: *Writing intelligently...Can't write block.: No such file or directory* . Since there isn't much documentation online regarding *bonnie++*, it took a while to understand that this issue was caused by a lack of disk space. Once again, we had forgotten that the EBS storage was capped to 10GB and that *bonnie++* was trying to use twice as much disk space as the available RAM. We first tested this on the *m4.large* instance which has 8GiB of RAM. *bonnie++* was trying to create files that would have a combined size of 16GiB, when we actually had roughly 6.3GiB of free space. To solve this issue we capped the memory in *bonnie++* which would also cap the combined file size that was allocated during testing. *bonnie++* has a parameter, *r*, that allows to set another amount of RAM to be used. Just like IO throughput benchmarking, we capped the maximum amount of allocated files to 5GiB to leave a buffer between the approximately 6.3GiB of available space. Since *bonnie++* allocates twice as much space as there is available RAM, we passed 2.5GiB for the *r* parameter.

3.3 CPU Memory Disk

Phoronix Test Suite offers a wide variety of different tests and it is definitely one of its main strength. However, choosing between so many tests was also a challenge. For example, some tests in a same category might have a runtime varying from a few seconds to multiple hours. Additionally, the descriptions on

OpenBenchmarking.org were really brief and did not always specify the parameters the tests were gonna execute with. For each test in this category, we had to do additional research to find the documentation for those tests and if they fitted our requirements.

A problem also encountered is that we did not find a way to completely automate the execution of the tests with Phoronix Test Suite. We found some ways to facilitate the test execution by creating our own test suite and then exporting the configuration file or by doing benchmarks in batch. However, as soon as we run Phoronix for the first time, it prompts us for authorization anyway. Because of this, we found the automation to be really difficult and we had to compromise by leaving the prompts given by Phoronix.

The time taken by the tests was also definitely an issue. Ideally, we would have used a test suite to test each of the CPU, memory and disk. However, as explained before, these were taking so long and in the context of this assignment, where we try to minimize the uptime of our instance's, it would not be viable. Because of this, we had to choose relatively simple tests and only explore some of their options. For example, to benchmark the CPU, we chose SciMark2, which focuses on single-processor performance. If it wouldn't take so long, we would also add a benchmark that can evaluate multi-threaded performance, such as the Himeno benchmark.

However, in a context where we would want to benchmark these systems with detail, the available test suites or picking many specialized tests would be interesting options.

3.4 Memory

In order to benchmark memory, we first used to memory block-size of 1K with both read and write operations. 1K is the default, but we also decided to try with a bigger block size to reduce the number of operations and understand the throughput. We also used the default parameters of running 100G for random and sequential reads. As we are writing this report, we noticed that the parameters of the block-size is not the same for random and sequential reads, which should be fixed in another run of this benchmark.

3.5 Disk

dbench is a very simple tool that has some limitations in its option. With only seven options, the numbers of parameters for this benchmark was limited. We decided to emulate 120 clients per process with a timeout of 10 secs. We chose those numbers in order to really stress the disk in a short period of time. We also tried to use hdparm for all of the instances. The first error that we made is to use the default /dev/sda paramter to test the disk when it should

be /dev/root for the EC2 instances. We also has an issue with the instance t2.2xlarge where the benchmark had an error indicating us that there was an invalid argument in the command in the script. We also had the error that hdparm works exclusively with devices using the ATA protocol, but the disk used with theses EC2 volumes uses the NVMe protocol. The benchmark still appeared when this error appeared.

3.6 Network Throughput

As mentionned, there was not any complications with using Speedtest. The command-line interface is really simple and only takes parameters to change the output. The benchmark itself calls Speedtest.net which handles most of the logic by itself, so we did not have to adjust the parameters to get the desired result.

4 Results

4.1 IO

According to our benchmarks, the c5.xlarge and m5.xlarge instances are the top performers in terms of IO throughput. They identical write and read throughput, 135 MB/s and 136MNB/s for write and read throughput respectively. Right behind these instances, we find the t2.xlarge and t2.2xlarge instances. The t2.xlarge instance has an average of 128 MB/s for write throughput and 131MB/s for read throughput. The t2.2xlarge instance has an average of 124MB/s for write throughput and 124MB/s read throughput. Far behind, we have the c4.xlarge instance with a 94.5MB/s write throughput and 95.4MB/s for read throughput. Lastly, the m4.large instance has a 55.9MB/s write throughput and 56.9MB/s read throughput.

We have included figures to show how much time it took to complete file copying operations in order to get a more precise look at these instances. Since our benchmark consisted in reading and writing a 5GiB file, we measured the throughput by dividing 5GiB by the time it took to complete the operation, but also left graphs of how much time it took to copy 5GiB. By looking at these figures, we can see the c5.xlarge is just slightly faster than the m5.xlarge instance for writing, but it is the opposite for read speed. Based on this results, we cannot say that the c5.xlarge or m5.xlarge is the best instance since the numbers are just too close.

We are quite surprised to see that the m5.xlarge and c5.xlarge instances had similar throughputs. Since c5 instances are optimized for computational power, we expected them to be much more powerful for IO operations than the m5 instance which is supposed to be a general-purpose instance. We are also quite surprised to see the throughput difference between the c4 and c5 instances as

well as the difference between the m4 and m5 instances. Since these instances are supposed to be in the same category, but only a newer generation, we expected just a slight increase in throughput. We are surprised to see that the m5 instance is more than twice as quick as the m4 instance and that they c5 instance is almost 1.5 times quicker than the c4 instance. We believed this is due to the fact that the new generation instances have clock speeds that are much superior to their previous versions and also have a much higher EBS throughput.

Figure 1: Write throughput for copying 5GiB of data

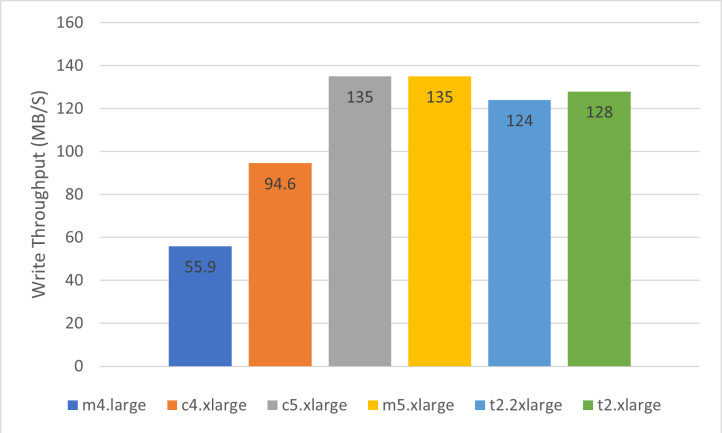


Figure 2: Time taken to write 5GiB of data

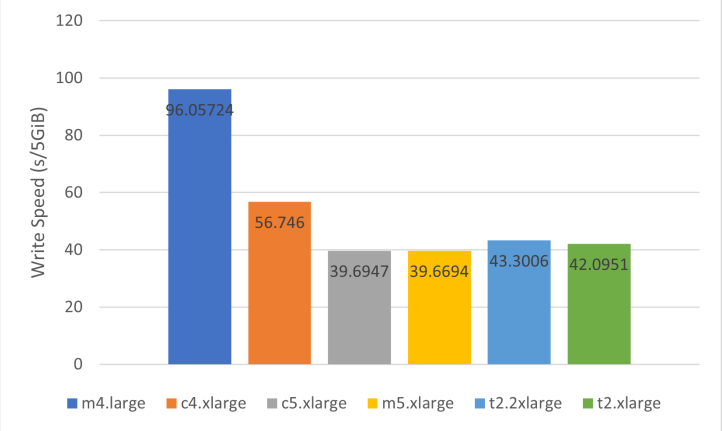
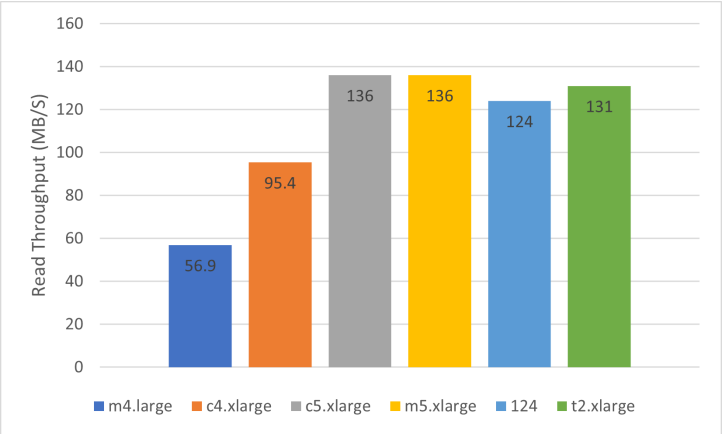


Figure 3: Read throughput for copying 5GiB of data



4.2 IOPS

According to our benchmarks, the c5.xlarge and m5.xlarge instances are once again the best. To perform these benchmarks, we used *bonnie++* to create files sequentially and randomly, and perform create, read and delete operations on 1024 * 100 files of size varying between 500 and 1000 bytes.

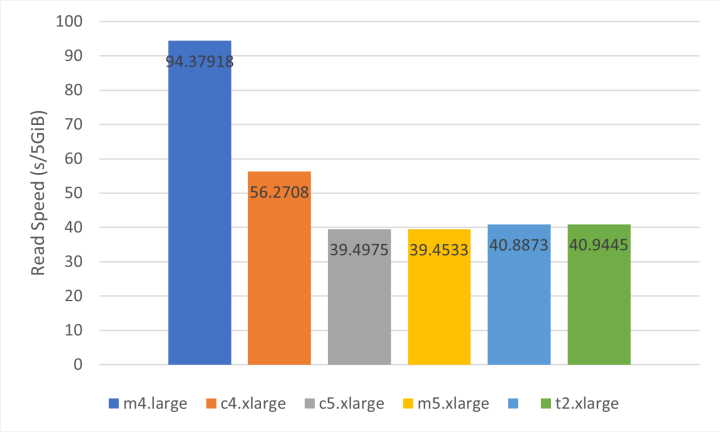
Figures 6 to 11 show the results of these tests. We have also included figure 5 to easily show the differences between instances. 5 makes an average over all six metrics and compares instances based on this average. This figure clearly shows that the c5.xlarge and m5.xlarge instances are able to perform the most IO operations per second.

Just like the IO benchmark, we notice just how big the difference is between the m4.large and m5.xlarge instances as well as the x4.xlarge and c5.xlarge instances. This once again shows that the new generation of EC2 instances are performing very well. We believe that this is due to the increased clock speed and EBS bandwidth of these instances.

4.3 CPU Memory Disk

First, Figure 12 shows the results for the CPU benchmark with the composite test from SciMark2. As we can see, the instances with the best performance are c5.xlarge, c4.xlarge and m5.xlarge with 520.90, 498.83 and 496.47 Mflops respectively. Lower than them, there is the m4.large instance with 456.57 Mflops, the t2.xlarge with 449.87 Mflops and t2.2xlarge instance with only 432.69 Mflops. Those two groups of instances form sets of similar performance. As expected, the compute optimized c4.xlarge and c5.xlarge instances both have great performance with this test. The disparity between the m4 and m5 instances is large,

Figure 4: Time taken to read 5GiB of data



but understandable: the m5.xlarge has a 3.1 GHz Intel Xeon® Platinum 8175M processor, which is noticeably more advanced than m4.large's processor (Intel XeonE5-2676 v3). Our hypothesis as for why the t2 instances have this result is because this test did not require the instances to "burst above the baseline" as the computations were not heavy enough.

Secondly, Figure 13 shows the results for the memory benchmark with RAM-speed. In the figure, the low result for the m4.large instance strikes, as its bandwidth is almost half the instance with the next lowest result. The m5.xlarge has the best result (26359 MB/s), and next is c5.xlarge. Both have good performance.

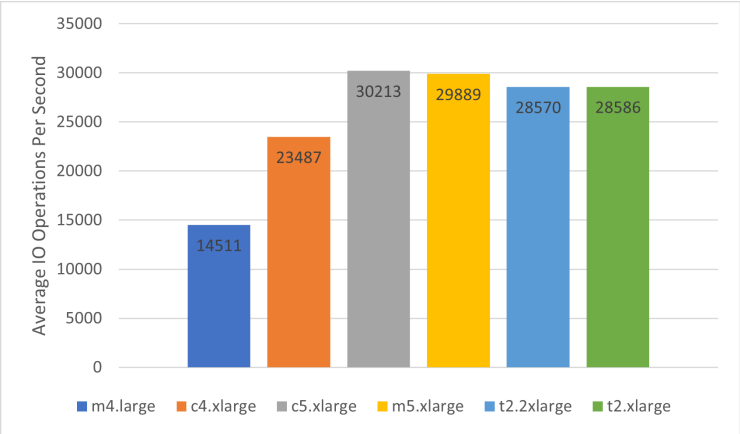
Finally, Figure 14 shows the results for random read on the disk. Again, we see that c5.xlarge and m5.xlarge outperform the other instances. They would be a good choice for this aspect as well.

4.4 Memory

Figures 15 and 16 below visualize the results of the 5 runs of benchmarks using the sysbench tool.

According to our benchmarks using sysbench, the c5.xlarge has the greatest memory throughput with an average of 17118 MiB/sec for sequential reads with a memory block size of 1024 KiB, and 1899 MiB/sec for random reads with a memory block of 1 KiB. Given the same benchmark parameters, the c4.xlarge and m5.xlarge instances are good second alternatives with m5.xlarge having a better throughput for random reads. Moreover, the c5.xlarge was among the lowest average maximum latency for both sequential and random. In fact, the

Figure 5: Average operations per second



c4.xlarge and c5.xlarge are the types of instances with the lowest maximum latency, respectively 0.12ms and 0.13ms for sequential read. We were surprised by the high 3.14ms latency for the m5.xlarge during our run. This may be an error, and should be retested to see if it was cause by an anomaly.

According to these two metrics, c5.xlarge is the best type of instance for memory. This particular instance is part of the Compute Optimized Instances offered by Amazon. Depending on the needs and budget constraints of the client, c4.xlarge could be a good second choice from these instances in terms of memory performance.

4.5 Disk

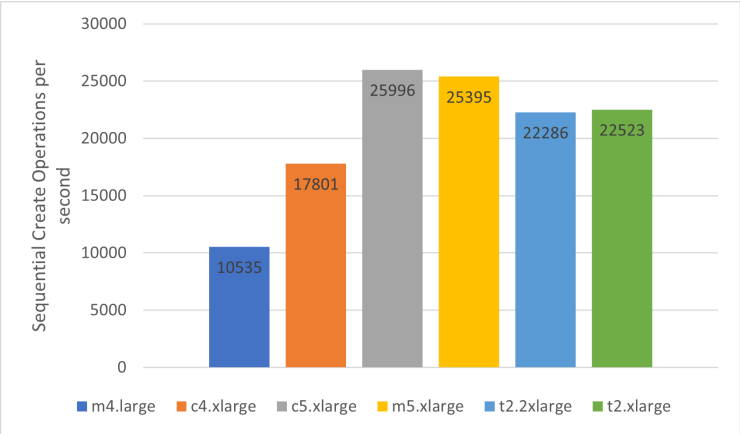
The Figures 17 and 18 visualize the results of the 5 runs of benchmarks using the dbench tool.

According to the results of the dbench benchmark, t2.2xlarge has the greatest disk throughput with 1073 MiB/sec, which is almost double the next best instances, t2.xlarge and m5.xlarge. The t2.2xlarge instance also has the best maximum latency with only 123 ms, which is also half the maximum latency computed for the same next best instances. The performance of latency and disk throughput are correlated for these instances according to this benchmark.

The Figures 19 and 20 present the results of the hdparm benchmark.

THe hdparm benchmark gives different results as the best average throughput. The c4.xlarge stands out with a throughput of 10481 MB/sec for cached reads, while the c5.xlarge has the best throughput of 170 MB/sec for buffered

Figure 6: Sequential create operations



disk reads. The hdparm direct benchmark that directly tests the hard drive regardless of the filesystem puts c5.xlarge and m5.xlarge at the top position with a direct cached reads of respectively 192 MB/sec and 171 MB/sec. Since we have missing values for the t2.2xlarge, we cannot used these results to confirm or not to confirm the dbench benchmark. Also, since we had a few errors including an inappropriate protocol for the volume, we will discard those results in our final recommendation. A better practive would be to fix the errors and reruns the benchmarks with other tools to confirm our results but due to a lack of time, we cannot run these tests again.

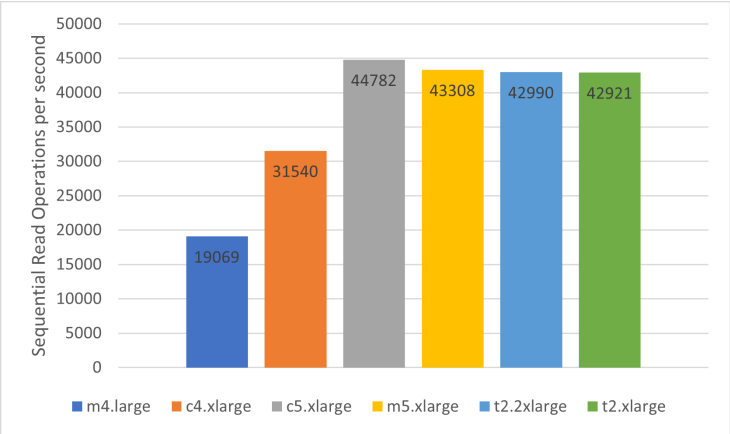
Hence, the t2.2xlarge would be the best option depending on the workload of the enterprise and the cost constraints.

4.6 Network Throughput

After running the Speedtest benchmark many times, the average for each instance gives the upload speed seen in Figure 21. Similarly, the results for the download speed is seen in Figure 22. Unsurprisingly, the network upload and download speed is highly correlated: instances with high upload speed will have high download speed, and vice versa.

From these results, we see clearly that the c5.xlarge and m5.xlarge instances largely outclass the other instances. These results fit with the data from the first table: both of c5.xlarge and m5.xlarge have an EBS bandwidth of 4750 Mbps and both of the instances have the highest network speed, averaging around 1950 Mbps and 2250 Mbps for the upload and download, respectively. On the website, Amazon specifies that m5.xlarge and c5.xlarge instances both have up to 10 Gbps for the network bandwidth, which is high and seems representative

Figure 7: Sequential read operations



with our results.

For the m4.large and t2.2xlarge instances, their network performance is described as "Moderate" by Amazon. This also corresponds with our results, as the c4.xlarge instance has "high" network performance and is slightly higher than both from about 250 Mbps. However, the t2.xlarge is an outlier: while its performance is also described as "Moderate", the average network bandwidth is higher than 50 to 100 Mbps for upload and download speed than c4.xlarge. Besides this instance, the results seem valid.

Figure 8: Sequential delete operations

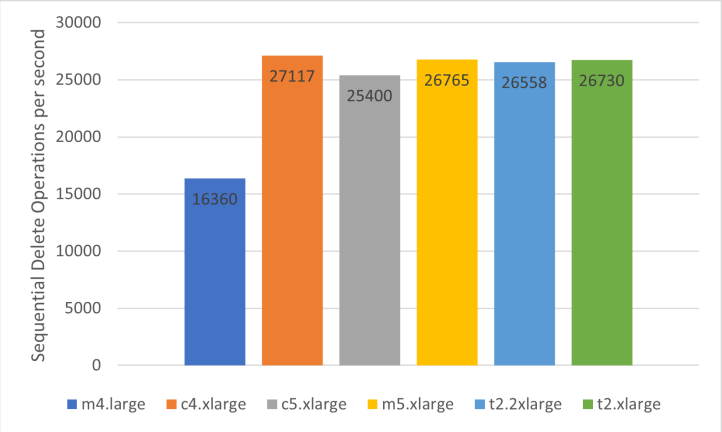


Figure 9: Random create operations

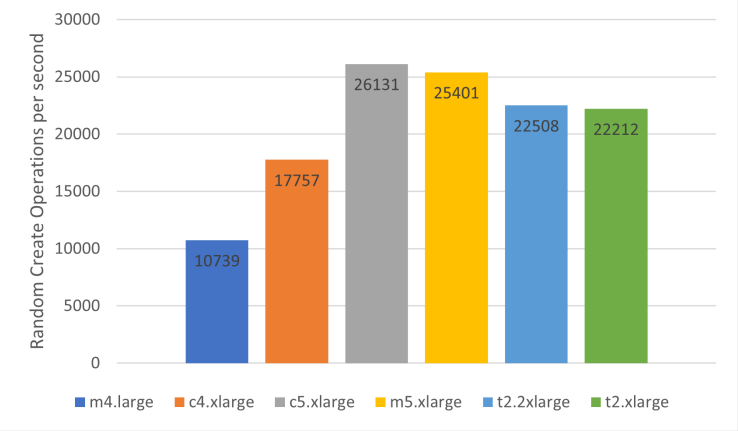


Figure 10: Random read operations

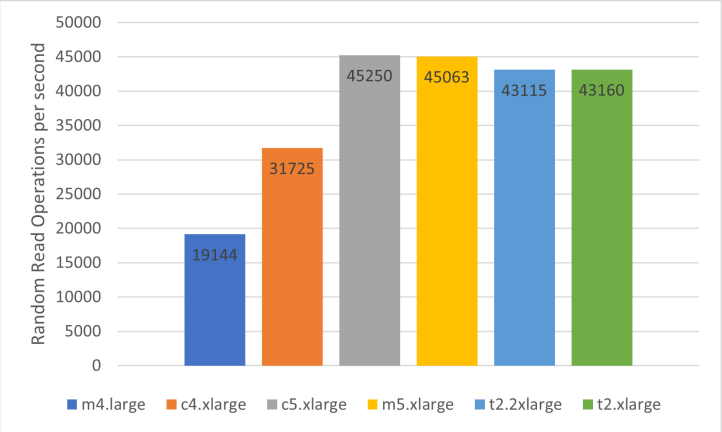


Figure 11: Random delete operations

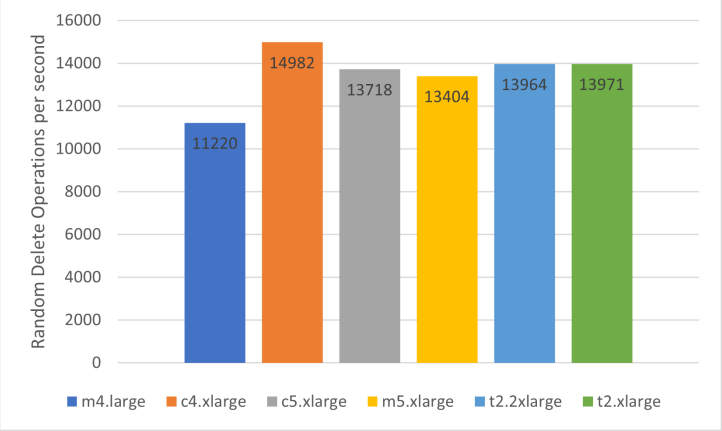


Figure 12: CPU Performance running SciMark2's Composite test

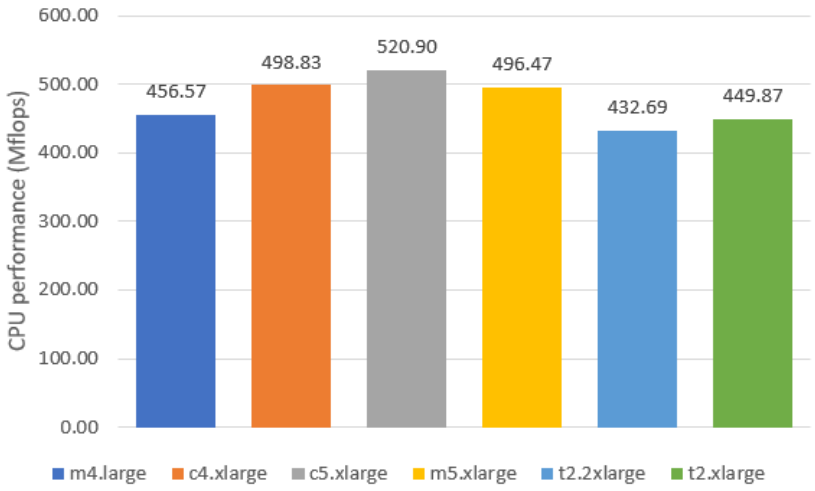


Figure 13: Memory Bandwidth with Copy operations on Integers

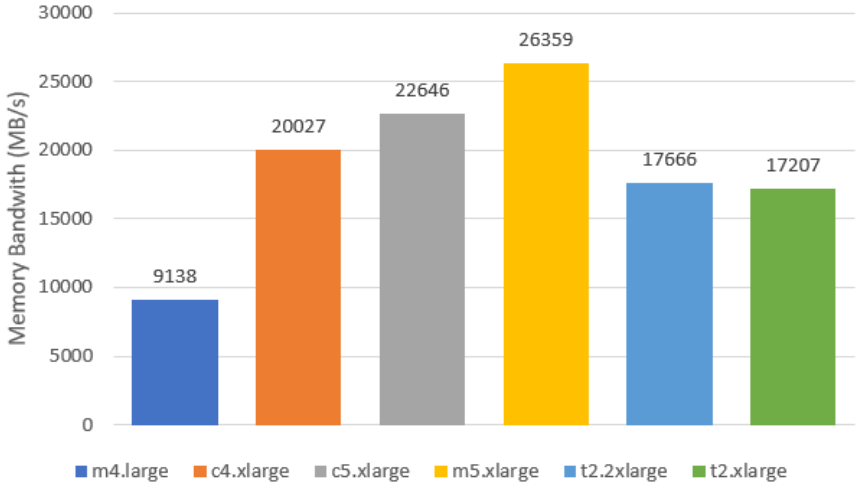


Figure 14: Disk random-read speed on 4kB of data

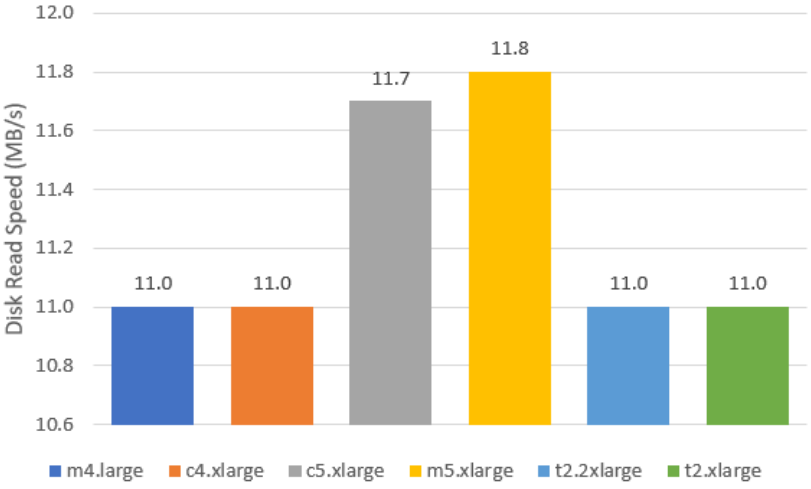


Figure 15: Memory throughput using sysbench

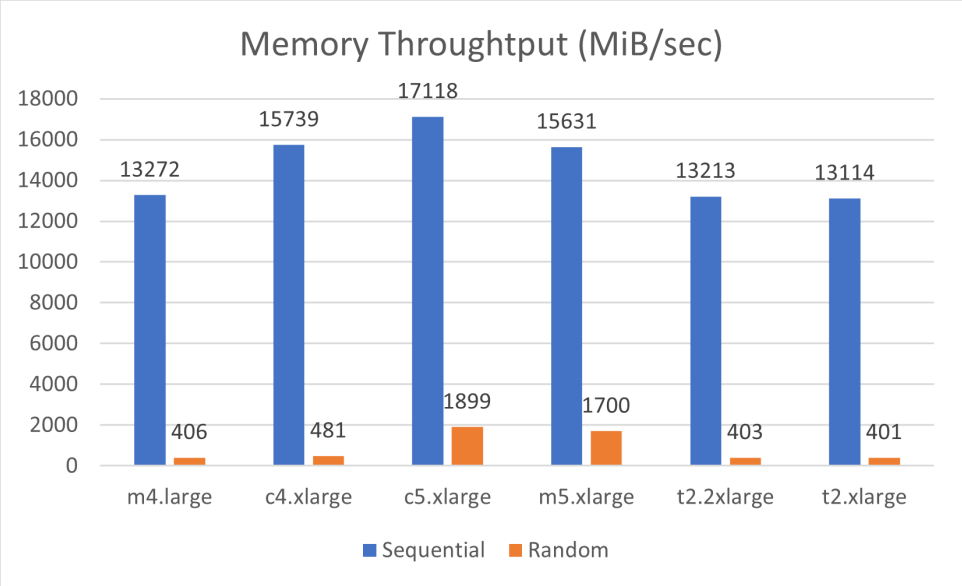


Figure 16: Maximum memory latency using sysbench

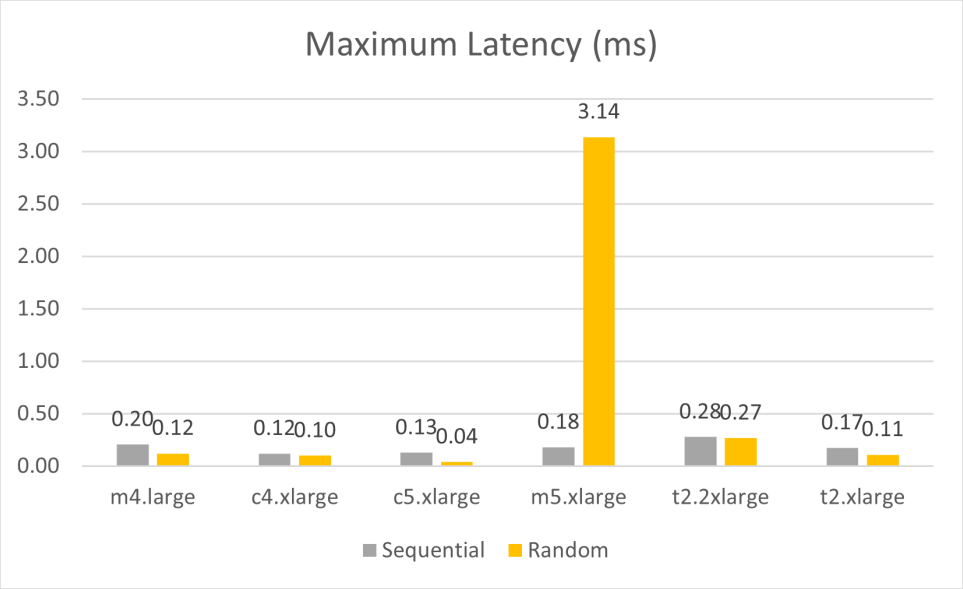


Figure 17: Disk Throughput using dbench

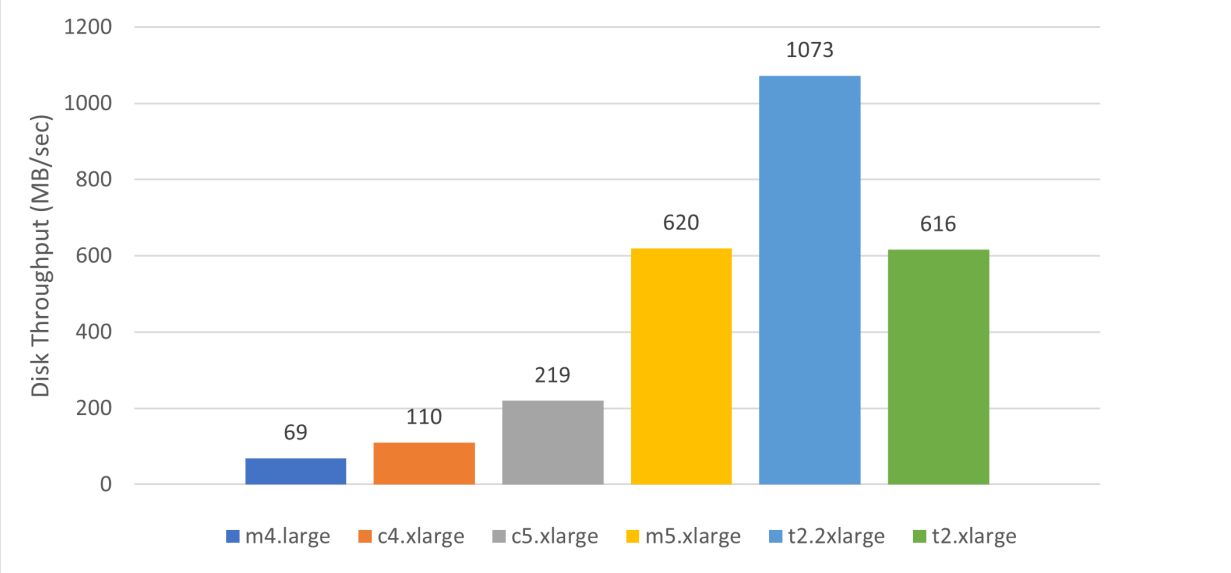


Figure 18: Disk Latency using dbench

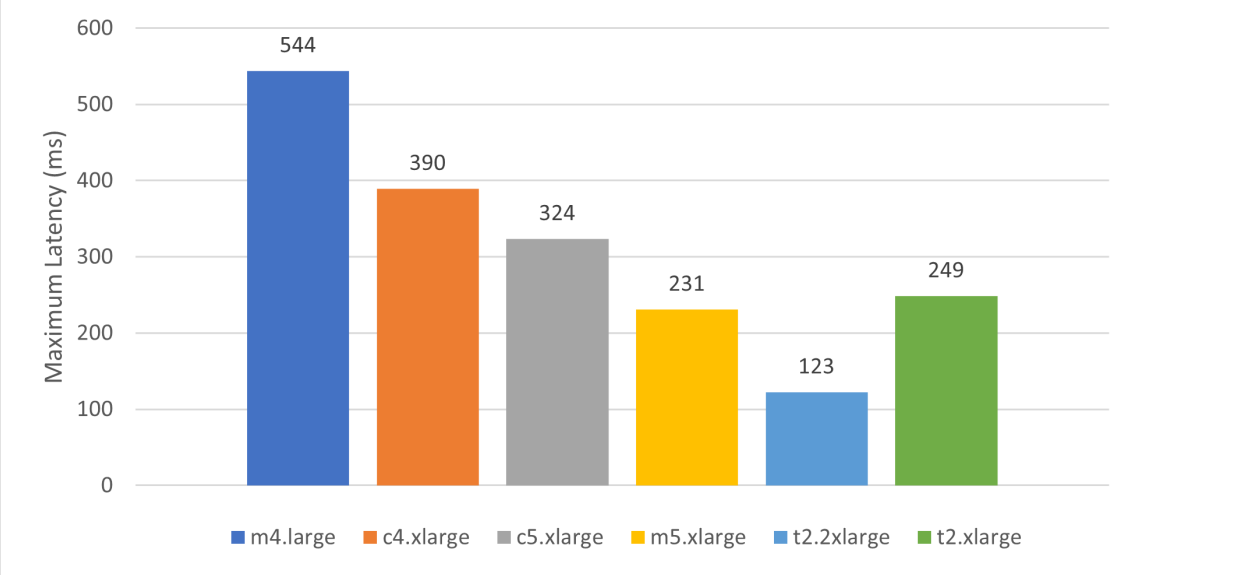


Figure 19: Disk Throughput using hdparm

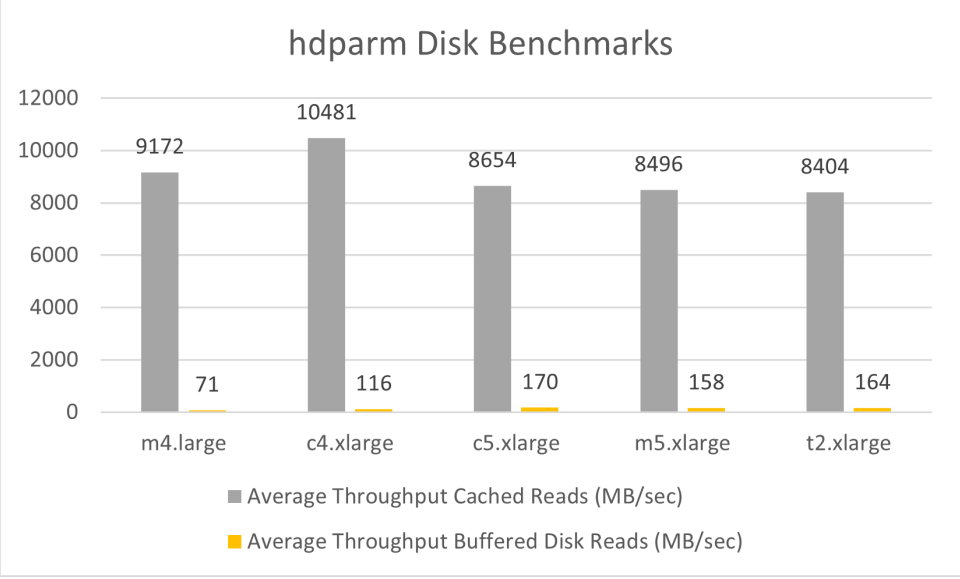


Figure 20: Direct Disk Throughput using hdparm

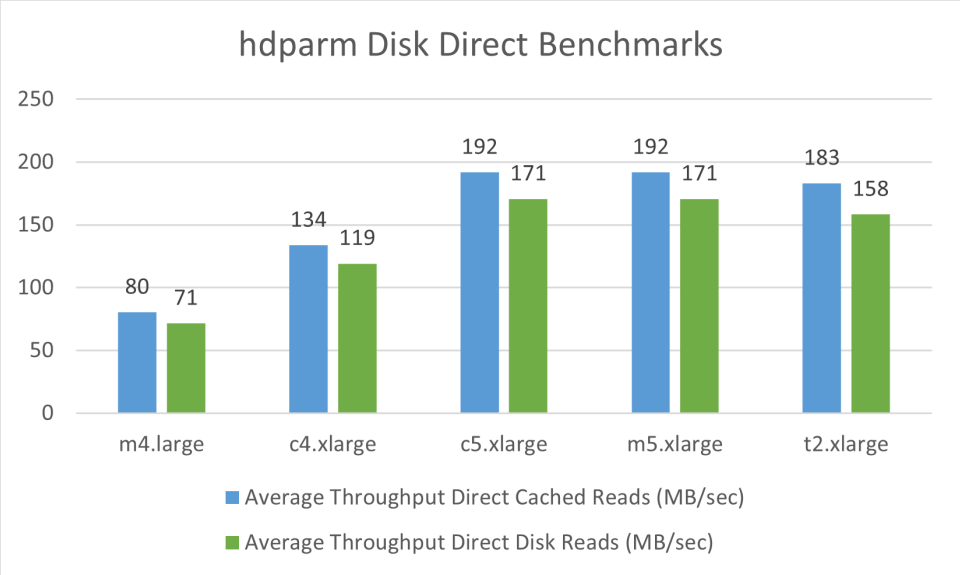


Figure 21: Average network upload speed using SpeedTest

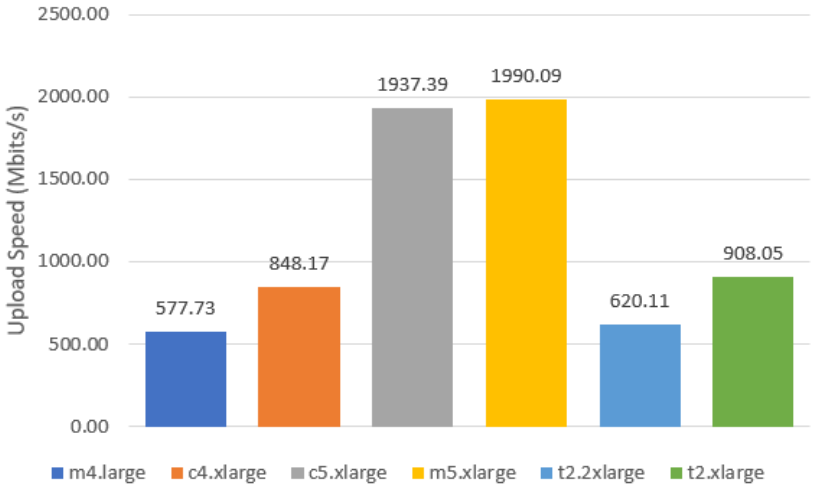
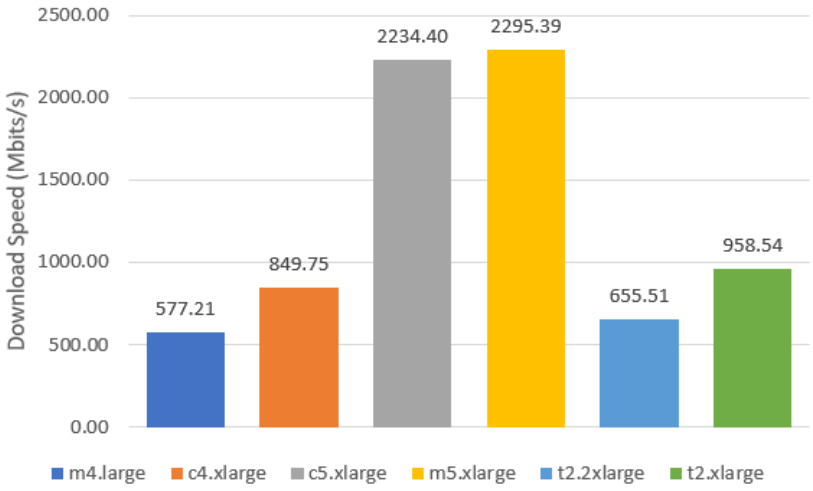


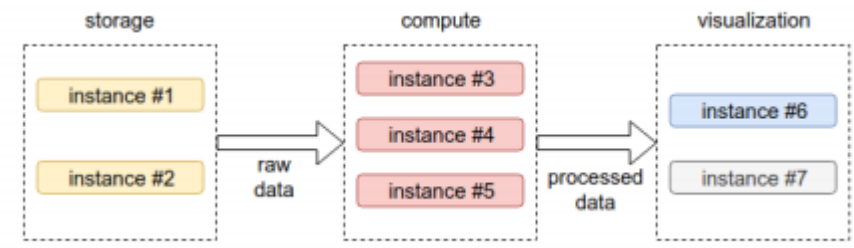
Figure 22: Average network download speed using SpeedTest



5 Proposed Architecture

Based on the benchmark tests described in the previous section and the business needs of the client, we propose the AWS Cloud architecture depicted in Figure X. Below, we describe which VM instances we propose for each section based on our benchmarks and requirements.

Figure 23: Design Architecture Elements



5.1 Storage

According to our requirements, we understand that the storage module requires a large amount of storage space, high IOPS and large throughput.

Thankfully, all instances are using the EBS, so we don't have to worry about running out of space. The EBS will allow us to use as much space as possible. computational power. Memory is also important to be able to handle large datasets. Disk performance as well as IO and IOPS performance was great for both m5.xlarge and c5.xlarge. The network upload was as good on m5.xlarge as c5.xlarge. We propose using two m5.xlarge instances since the performance is one of the best and the cost is much lower than c5.xlarge instances.

5.2 Compute

According to our requirements, we understand that the compute module requires extensive use of IO operations as well as good ropose using two m5.xlarge instances since they perform very well under all criter

For IO and IOPS, we believe that the m5.xlarge instance is much better suited than the c5.xlarge. They have very similar performance, but the m5.xlarge is much cheaper. The same goes for CPU performance, the c5.xlarge doesn't provide a big advantage to the m5.xlarge instance. The network upload and download speeds are good for m5.xlarge and x5.large. The only disadvantage the m5.xlarge instance has over the c5.xlarge is the max latency for memory.

To balance this out, our suggestion is to use two m5.xlarge instances and one c5.xlarge instance.

5.3 Visualization

According to our requirement and our research, business intelligence requires a lot of read operations and needs to download the data efficiently. As such, we have to consider the instances with good read I/O operations and good network throughput, especially for the download. Also, visualization requires heavy computations, so CPU and memory have an important role.

From these criterias, it seems clear that the best instances are both m5.large and c5.xlarge. They exceed in CPU performance as well as network download speed and random read operations.