Polytechnique Montréal

LOG8415 : Lab 1

Advanced Concepts in Cloud Computing

Selecting VM instances in the Cloud through benchmarking

Authors
Louis Boudreau (1791639)
Laurent Pepin (1739608)
Tristan Mercille-Brunelle (1740701)

February 24, 2019

Contents

1 Abstract

Cloud application architects can choose from a wide selection of VM instances to achieve high levels of performance. Choosing the right instance for the right job is not an exact science. Even if cloud providers classify their instances in categories to help consumers choose right, benchmarking instances is still the best way to properly choose instances for a cloud application. In this paper, a benchmarking analysis is presented on 5 different instances available on Amazon Elastic Compute Cloud in order to find the best instances for a specific cloud application architecture.

Keywords: Amazon Elastic Compute Cloud, Regression Analysis, Benchmark, Instance Performance, Cloud Application, Choice of Instance for Cloud Application

2 Introduction

Cloud providers offer multiple services such as platforms, infrastructure and software as a service. Each of these categories contain plenty of products users 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 choose from a large variety of server instances, each having their set of technical particularities. [1]

Since all cloud applications meet different requirements, having the possibility to choose from dozens of server instances, all optimized for specific job types, is a perk. 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. However, since no cloud application is the same, there is no clear recipe on how to choose the right instance. Benchmarking instances to compare their performances is a great way to find the instances matching the requirements of every module of a cloud application.

In this paper, we use benchmarking on multiple Amazon EC2 instances to find the best fit for a cloud application with storage, heavy data operations and heavy computation needs.

This paper presents the methodology used to find the best instances for the application (section 3), the results of the regression analysis done to find the right parameters for benchmarking (section 4), the benchmarking results and analysis (section 5) and the application of these results to choose the right instances for the cloud application (section 6).

3 Methodology

In order to find the best instances for a specific cloud application, it is necessary to understand the architecture of said application and the needs of the different modules it is made of. Then, server metrics must be chosen to serve as comparison tools. Tools to gather those metrics must also be selected along with their respective configuration parameters. Finally, benchmarking the instances must be done as a repeatable process.

3.1 Cloud Application Architecture

The cloud application targeted by this paper aims to help an online retailer company (Alpha X) get insights on how their business evolves over time. The application must produce data driven information and be boosted by intelligent models. The cloud application is organized as follow: 2 server instances for storage, 3 instances for data extraction and extensive input/output operations and, finally, 2 instances for data computation. These 7 instances can all be different. [2]

The architecture indicates that some instances might need to be more storage focused, while others should be chosen for their high input/output performance or heavy computations capabilities.

3.2 Metrics of Comparison

In order to compare instances performances in respect of the application needs, metrics on these instance characteristics were selected: CPU, IO, IOPS, memory, disk and network throughput.

3.2.1 CPU - Computation Time

CPU of different instances can be compared using the speed at which they are able to perform a computation. In this paper, we use the time it takes to compute prime numbers as the metric of comparison for CPU. The benchmarking tool *Sysbench* 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. [3]

3.2.2 IO - Throughput

Formally, I/O (Input/Output) is the operation of taking information from a source A and sending it to a destination B. In the context of cloud computing, multiple forms of I/O such as network I/O, disk I/O and files I/O exist. In this paper, we approached I/O in regards to the disk. The metric used to make the benchmarks is the throughput, i.e. the quantity of data transferred over the time it took to accomplish the task. The utility tool used to benchmark the I/O is *dd*, a Linux built-in software used to copy files from a specified source and destination.

3.2.3 IOPS

Input/output operations per second (IOPS) is a metric mainly used to evaluate an instance ability to support a high load of file operations. In the context of this paper, the number of creates per second, reads per second and deletes per second in both sequential and random ways will be used to compare instances IOPS capabilities. *Bonnie++* will be used to perform the tests. *Bonnie++*

supports multiple parameters, one of which is the number of files to create, read and delete on each test. In this paper, we will use the same number of files for every instance to have a comparison basis of IOPS performance. [4]

3.2.4 Memory

Memory (RAM) performance can be measured with the operations per second it can support. Using the tool *stress-ng*, it is possible to gather meaningful data on how well instances handle memory pressure. The tool offers a variety of stressors. To test instances memory performance, the following stressors were chosen: expand heap break point, expand stack and bigheap. These stressors impact the memory and monitor its performance by trying to consume memory at a high rate. The number of operations per second for the categories (heap break point, stack, bigheap) will be used to compare instances. All instances will be tested with the same number of stressors. [5]

3.2.5 Disk - Read Throughput and Latency

The disk of an instance is the physical device where data is persisted. We used two metrics in order to efficiently benchmark this device. The first one is the disk latency, it refers to the elapsed time between a request for some data and the actual return of the information. The latency is then critical for applications requiring a lot of small consecutive disk access. *Ioping* is the tool we used to compute this metric, it computes the latency mean of multiple consecutive access to the disk requesting for 4 kb of data. The second one is the read throughput and aims at measuring how fast can the disk deliver data. This metric is important for applications needing less frequent disk access but reading huge amount of data each time. Disk read throughput is measured with the quantity of data transferring over the time it took to accomplish this task. Nowadays, disks are built with multiple layers of cache to increase access speed of frequent addresses. The OS must then optimize the way it stores information to take the most out of this cache layer and further increase the system's performance. The tool used in order to retrieve this information is *Hdparm*. This command line program has options to compute read throughput with and without caching, which we both used.

3.2.6 Network Throughput

Network throughput can be evaluated using download and upload speeds, in bit/s. *Speedtest-cli* will be used to compare network throughput of all instances by evaluating their download and upload speeds. All instances will be tested against the same server, located in Montreal. [6]

3.3 Regression Analysis

Choosing the right parameters for the benchmarking tests described in the previous section has been done using regression analysis. Regression analysis aims to find the variables having impact on a set of data. [7] This work uses regression analysis to find the limits of each parameter of every benchmarking test. The goal 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, the selected value will be the average between the lower and top

limits. Section 4 exposes the method and results of the regression analysis done to find the right parameters for each test.

3.4 Instances Types

Amazon EC2 catalog contains dozens of server types. This paper focuses on Ubuntu machines running the last version of the OS (18.04). A total of 5 virtual machines will be compared. One of them is categorized as a general purpose instance, two of them as compute optimized, one of them as memory optimized and the last one as storage optimized. Here is a table showing the chosen instances and their characteristics, as provided by Amazon.

Table 1: Chosen Instances and their Characteristics [8]

Instance Model	Family Type	vCPU	Memory (GiB)	Disk (Mb/s)	Network Speed (Gb/s)
a1.large	General Purpose	2	4	N/A	Up to 10
c5.xlarge	Compute Optimized	4	8	Up to 3500	Up to 10
c5.2xlarge	Compute Optimized	8	16	Up to 3500	Up to 10
r5.large	Memory Optimized	2	16	Up to 3500	Up to 10
h1.2xlarge	Storage Optimized	8	32	N/A	Up to 10

Terms such as Computed Optimized or General Purpose are quite subjective and that's why it's interesting to define them further. [9][10][11][12]

General Purpose: A1 instances deliver significant cost savings for scale-out and Arm-based applications such as web servers, containerized micro-services, caching fleets, and distributed data stores that are supported by the extensive Arm ecosystem.

Compute Optimized: C5 instances offer the lowest price per vCPU in the Amazon EC2 family and are ideal for running advanced compute-intensive workloads.

Memory Optimized: R5 instances are well suited for memory intensive applications such as high performance databases, distributed web scale in-memory caches, mid-size in-memory databases, real time big data analytics, and other enterprise applications.

Storage Optimized: H1 instances are a new generation of Amazon EC2 Storage Optimized instances designed for applications that require low cost, high disk throughput and high sequential disk I/O access to very large data sets. Offering the best price/performance in the magnetic disk storage EC2 instance family, H1 instances are ideal for data-intensive workloads such as MapReduce-based workloads, distributed file systems, such as HDFS and MapR-FS, network file systems, log or data processing applications such as Apache Kafka, and big data workload clusters.

3.5 Benchmarking

The methodology used to benchmark the instances was to sequentially stress the instance with CPU, IO, IOPS, memory, disk and network, in this order, 5 times. The goal of the test is to find the

consequences of the benchmarks on each other and compare instances over multiple characteristics. The sequence of test is done 5 times to be able to take the average of the instance performance, but also analyze the effect of continuous stress on the performance.

3.6 Open Source Code

Traces of the work done on the instances and of the results analysis can be found online in a GitHub repository: `https://github.com/LaurentPepin/LOG8415E`

4 Regression Analysis

In this section, results of regression analyses done for each metric are presented.

4.1 CPU

For CPU, regression analysis was done by finding an integer big enough so that the most compute optimized instance would take 10 minutes to execute the Sysbench workload. Since computations can run forever, we, as a team, concluded that 10 minutes would be long enough to illustrate a pertinent variance between results of the different machines. We then made a script to gradually increase the max prime numbers until we reach the 10 minutes mark. The instance we took as the most powerful one is the c5.2xlarge since is has 8 virtual cpu. The following graph illustrates our results.

Figure 1: CPU Regression Analysis

150970010 is the retained value and therefore the following command is the one used in ours experiments :

*$ sysbench –max-time=1 cpu –cpu-max-prime=150970010 run**

It is worth saying that we used 1 for the max-time since Sysbench doesn't stop until he finishes computing all the prime numbers at least one time. It is also worth pointing out that the method used is not a formal regression analysis since we didn't took the average between a minimum and a maximum. The value retained doesn't push weaker instances outside of their hardware limits instances and therefore using the average between the minimum and the maximum would have only reduced the gap between computation times.

4.2 IO

Finding an instance IO maximum that would cause the machine to crash is unlikely, since storage is elastic. That being said, the regression analysis was done by finding a transfer load that was big enough so that the most efficient instance would take 10 minutes to finish the transfer. As for CPU, we, as a team, concluded that 10 minutes would be long enough to illustrate a pertinent variance between results of the different machines. The test employed for benchmarking aims at filling a disk partition until there is no more place left and save the elapsed time. In order to find the right EBS volume size for our test, we started by taking samples of the most powerful machine's throughput in order to roughly estimate the true one. The following table presents our samples.

Table 2: IO Regression Analysis

Transferred block quantity	Throughput (MB/s)
1	5.5
10	70.8
100	74.4
1000	58.4
10000	70.3
Average	55.8

The instance we took as the most powerful one is the h1.2xlarge since it is described as designed for high disk throughput and high sequential disk I/O access. Since the average throughput is 55.8 MB/s, we computed that EBS volume size needed was 31GiB.

$$55.88MB/s \times 10^6 B/MB \times 600s \div 2^{30} B/Gib = 31.22GiB$$

The following command is the one used in our IO throughput benchmark:

$ sudo dd if=/dev/zero of=/dev/$DISK_PARTITION bs=64K

dd will transfer 64kB data blocks from the /dev/zero file, which provides as many null character as you want, to the 31 GiB EBS volume until it's full.

4.3 IOPS

For IOPS, regression analysis was done by running the test on the smallest machine (a1.large) with different values for the *number of files* parameter. This parameter takes an integer and *bonnie++* multiplies it by 1024 to get the number of files it needs to work with. By playing with this parameter, we vary the load applied by the test on the instance. Table 3 shows results of the regression analysis with a starting value of 500.

Table 3: IOPS Regression Analysis Results

Test	File count (x1024)	Sequential			Random		
		Create (ops/s)	Read (ops/s)	Delete (ops/s)	Create (ops/s)	Read (ops/s)	Delete (ops/s)
1	500	43,333	480,990	4,102	41,652	552,323	2,879
2	750	27,400	475,891	3,661	27,948	542,583	2,385
3	1000	-	-	-	-	-	-

Results show that when the parameter is set to 1000, the test fails to produce any result. Considering this value as the maximum and 500 as the minimum, 750 (average between upper and lower bounds) will be used for benchmarking. It is also important to note that the value 1000 made the test fail on all 5 instances, while 750 worked in all cases.

Here is the command to be used for benchmark:

$ bonnie++ -d ./iopsTestFiles -s 0 -n 750 -m $INSTANCE -x 1 -q

4.4 Memory

Regression analysis for memory was done by varying the number of stressors and counting the number of memory errors produced per test (memory errors are recovered from and do not make the test crash). Every stressor (brk, stack, bigheap) had the same number of stressors, which is the varying parameter. For every instance tested, the goal was to find the number of stressors (same for all 3 categories of stressors) that would produce more than 11 memory errors, at which point the test would stop. The highest number of stressors for the last test across all instances would be the maximum value of the parameter. The minimum is 1. Table 4 shows the results obtained.

Table 4: Memory Regression Analysis Results

Number of stressors	Number of Memory Errors				
	a1.large	c5.xlarge	c5.2xlarge	r5.large	h1.2xlarge
1	2	2	0	0	0
2	4	4	4	4	1
3	6	6	6	6	6
4	8	8	8	8	8
5	10	10	10	10	10
6	12 (end)	12 (end)	12 (end)	12 (end)	12 (end)

Results show that 6 stressors per category, total of 18, is a maximum for all instances. Since the minimum is 1 stressor, an average of 3 stressors will be used for the benchmarks.

Here is the command to be used for benchmark:

$ stress-ng –brk 3 –stack 3 –bigheap 3 –metrics-brief –timeout 60s

4.5 Disk

There is no parameter other than the disk name used for the disk's benchmarks. Therefore, there is no need for regression analysis. Here are the commands to be used for benchmark:

$ ioping -c 10 ./

$ sudo hdparm -Tt /dev/$DISK_PARTITION

4.6 Network Throughput

There is no parameter for this test except the server to test against. Therefore, there is no need for regression analysis for network throughput. Here is the command to be used for benchmark:

$ speedtest-cli –csv –server 911

4.7 Summary

Table 5 summarizes the parameters values to be used for benchmark.

Table 5: Summary of Tests Execution

Category	Test Name	Parameters	Value
CPU	sysbench	cpu-max-prime	150970010
IO	dd	partition size	31GiB
IOPS	bonnie++	n	750
Memory	stress-ng	brk & stack & bigheap	3
Disk	Ioping	Folder path	./
	hdparam	Disk partition	EBS block partition
Network	speedtest-cli	-	-

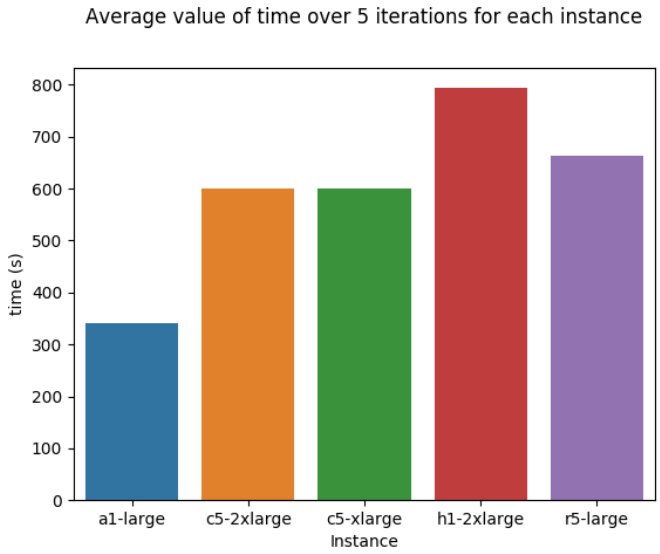
5 Benchmarking and Results Analysis

Benchmarking instances gives meaningful data regarding instances performances in categories. Also, running the test sequence 5 times in a row gives information on how instances perform under different levels of pressure over time. Following are the results of the benchmarks done on the 5 instances. It is worth noting that each benchmark result will be presented in two different forms. The first presents the results of all the instances through their 5 respective iterations and the second is a comparison of the 5 instances average results.

5.1 CPU

As discussed earlier, CPU performance was monitored by finding the time (in seconds) it took an instance to find the maximum prime number under a specific value. Figure 2 shows the average time needed by each instance to complete the CPU test over 5 executions.

Figure 2

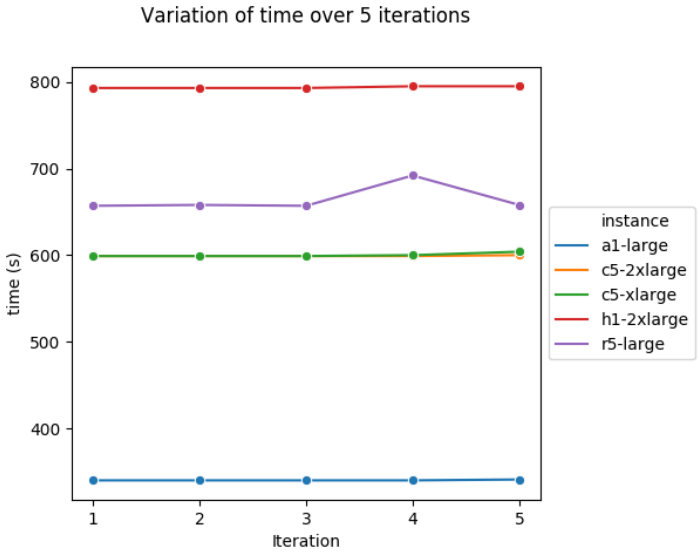


Results show that the a1.large machine is the instance that performs the best with an average time of 340.2s. This result is almost half the time required by the second best instance, c5.2xlarge, to complete the test (599.2s). Also, it is clear that h1.2xlarge is the instance that performs the worst with an average of 793.8s. The standard deviation for these averages is 165.23s which confirms that there is variability between the instances performances.

It is surprising to see a1.large, a general purpose instance, outperforming c5.xlarge and c5.2xlarge instances, since these last two instances are categorized as compute optimized by AWS.

As far is stability is concerned, all instances achieve an almost constant performance over the 5 iterations, as shown in figure 3.

Figure 3

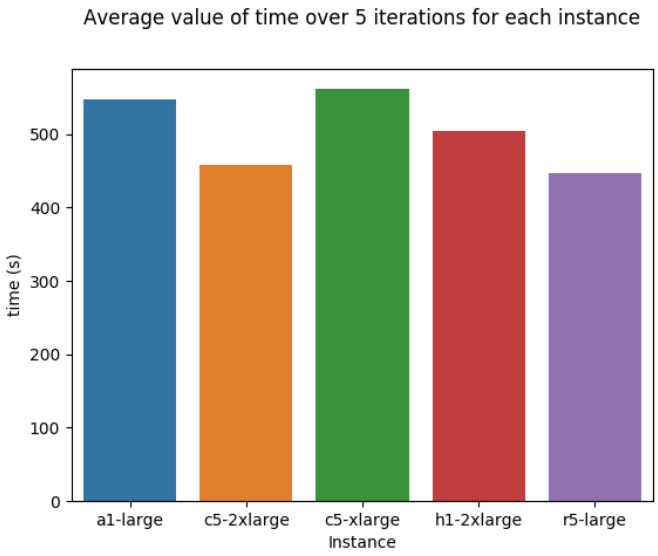


Therefore, there is no instance that dramatically loses CPU performance under high stress. Except for r5.large (standard deviation of 15.44s), all instances have a standard deviation less than 2.5s which confirms their stability.

5.2 IO

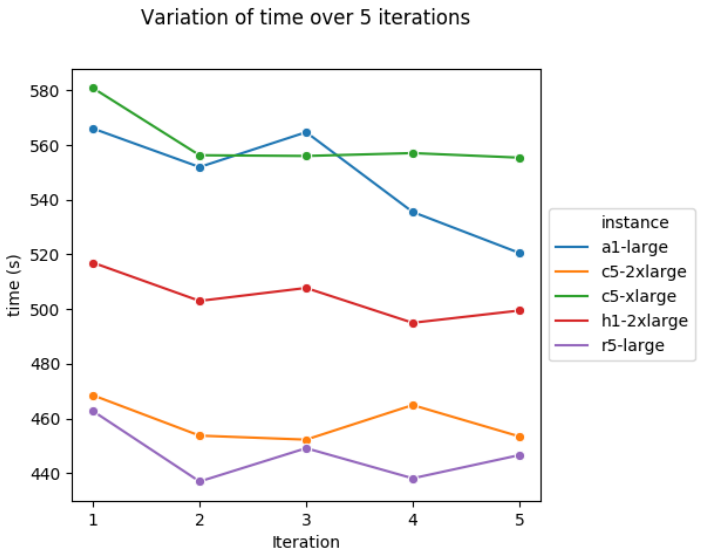
As presented in previous sections, IO throughput was computed by recording the time needed for an instance to transfer 31GiB of data. Figure 4 shows the average time needed by each instance to complete the data transfer over 5 executions.

Figure 4



Results show that the instance that performs the most was the r5.large with an average transfer time of 446.699 seconds which represents a throughput of 74.5 MB/s. On the other hand, the instance which has the worst performance is the c5.xlarge with an average transfer time of 561.13 seconds which represents a throughput of 59.32 MB/s. The results standard deviation is 51.3s which means that there was some variation between instances results but not a huge one. It's interesting to point out that the fastest instance was not the Storage Optimized one (h1.2xlarge). Figure 5 illustrates the variation of instances performance over their iterations.

Figure 5



By analyzing the graph we can conclude that the IO throughput is quite consistent with an average standard deviation of 11.38. Therefore, we can assume that the stress applied on the instances due to the other benchmarks isn't affecting IO. There is even one instance, a1.large, for which performance increases with iterations by an average of 11.37 seconds per iteration.

5.3 IOPS

Input/output per second benchmarks were measured for each instance 5 times, using a load of 750 (*1024 files) to create, read and delete sequentially and randomly at every iteration (6 measures). Table 6 summarizes the average results for each measure over the 5 iterations and 5 instances, with the associated standard deviation.

Table 6: IOPS Measures Averages and their Standard Deviations

Measure (qty/s)	Average (file/s)	Standard Deviation (file/s)	Coefficient of Variation (%)
Sequential Create	39,011.72	7,931.51	20.33
Sequential Read	541,838.84	204,016.20	37.65
Sequential Delete	2,685.64	313.91	11.69
Random Create	27,392.08	8,483.13	30.97
Random Read	670,588.44 &	197,159.75	29.40
Random Delete	1,433.48	18.38	1.28

These results show that create and read operations are associated with higher variance of performance between instances with a coefficient of variation over 20% compared to the delete operation. In fact, all instances delete files at almost the same rate while they do not perform at the same level for create and read operations, as the graphs in figure 6 confirm.

While it is clear that the a1.large instance underperforms all other instances, there is no clear winner. Instance h1.2xlarge outperforms other instances for create and delete operations but is second to last for the reading speed. Its high performance in creating and deleting file is not surprising since it is classified as a storage optimized instance. Among the three other instances, instance c5.xlarge is the one performing at a high level in all conditions.

As far as instances performances over 5 iterations (with 5 levels of stress) are concerned, all instances seem to behave the same way, as the graphs in figure 7 display.

It is clear that all instances performances in random file creation, sequential file deletion and random file deletion tend to dramatically go down when they reach the 3rd or 4th iteration. It is interesting to note that, for random creation, instances c5.xlarge and h1.2xlarge lose performance at the last iteration, while the 3 other lose their performance at the 4th iteration. For the other 3 measures, instances don't lose any noticeable performance due to stress. Therefore, there is no instance that sustains stress better than all the others.

Figure 6: IOPS Average Tests Results

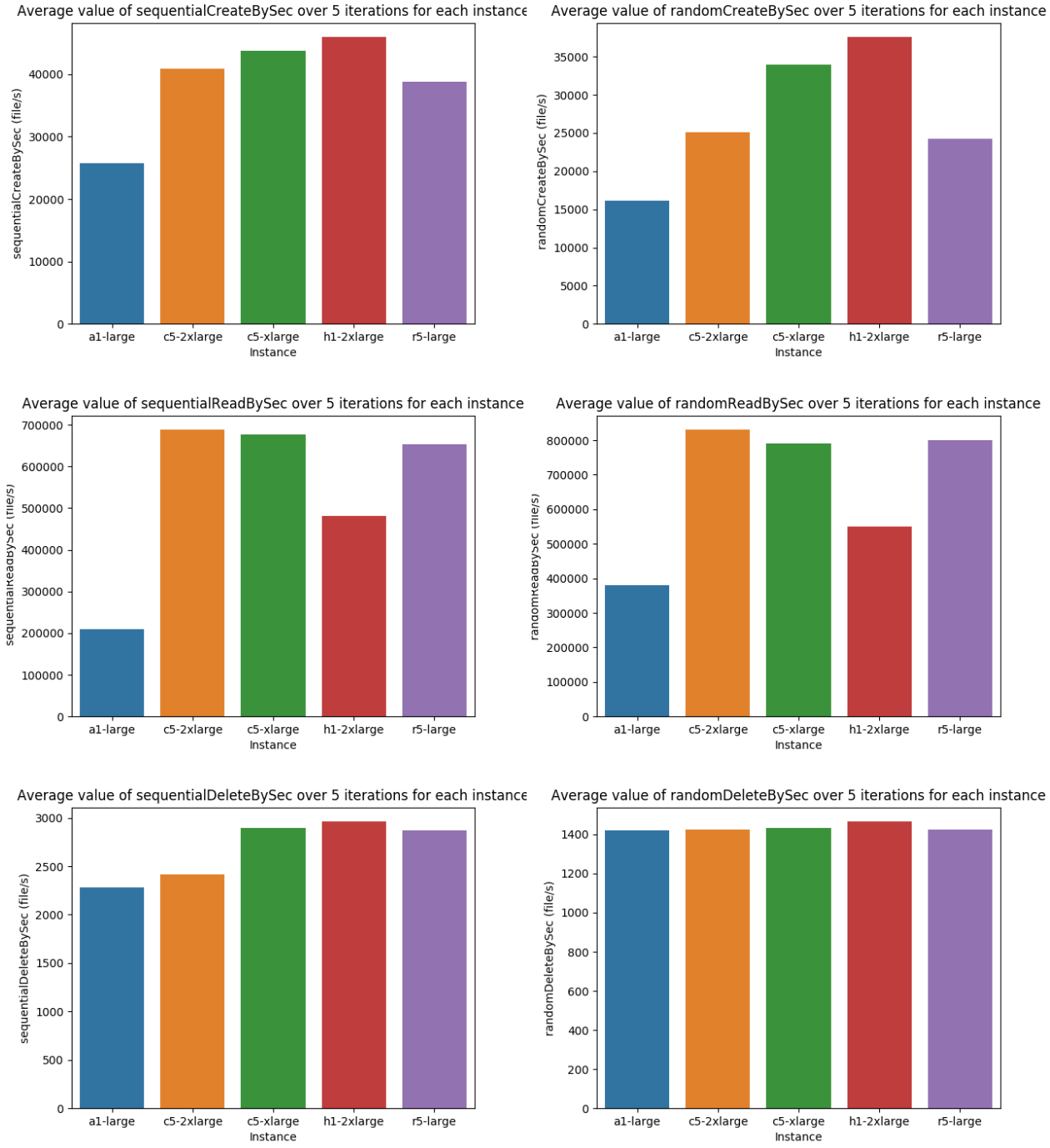
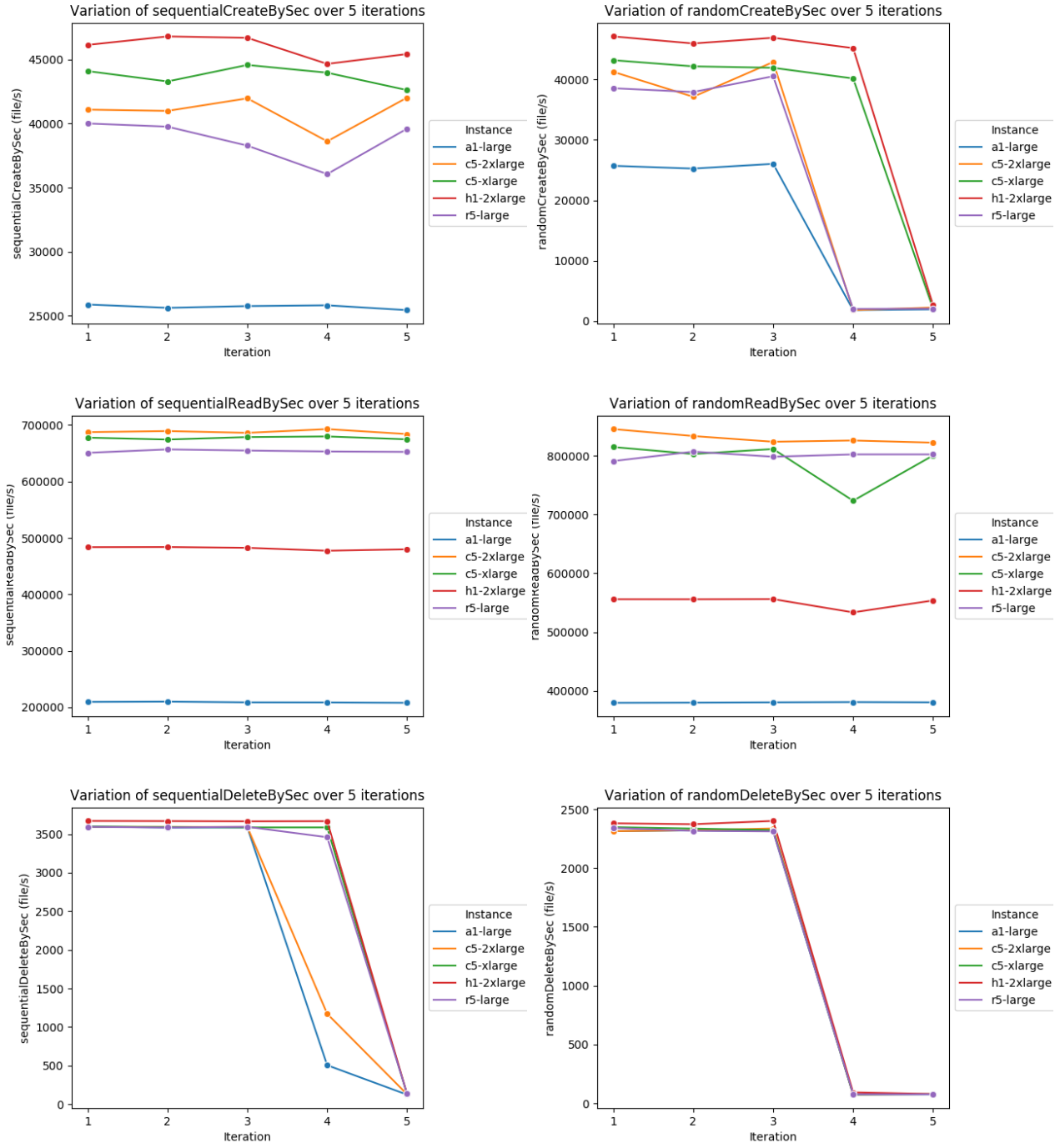


Figure 7: IOPS Variation Tests Results



5.4 Memory

Benchmarking for memory consisted of measuring the number of operations per second for 3 memory stressors. Results show instances memory performances. Figures 8, 9, 10 show the average results for every instance for the different measure techniques.

Figure 8

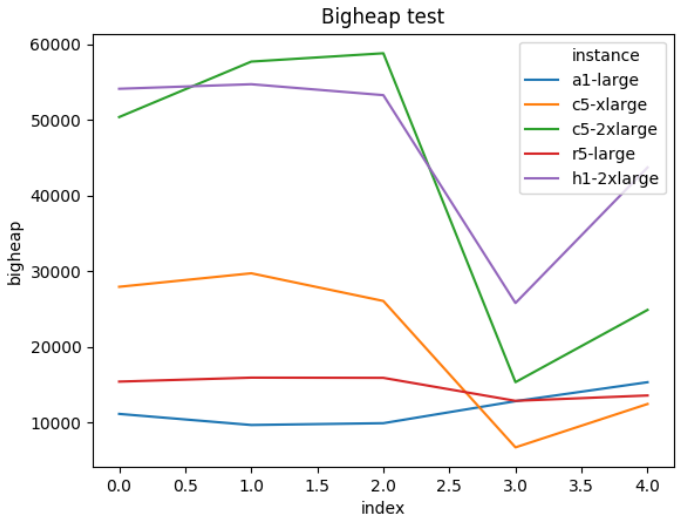


Figure 9

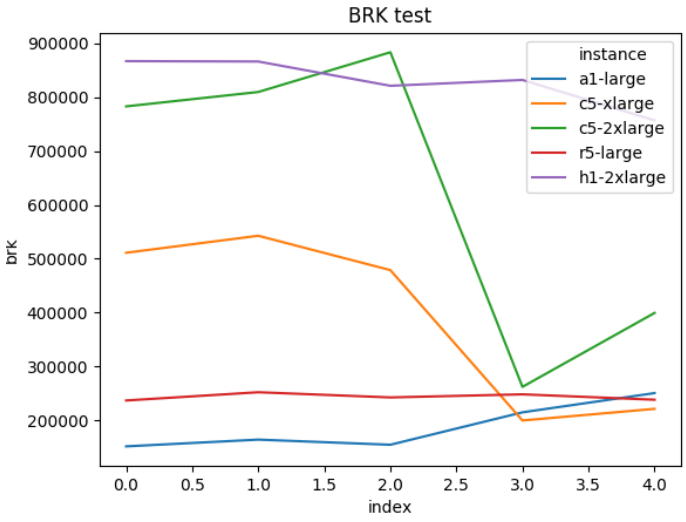
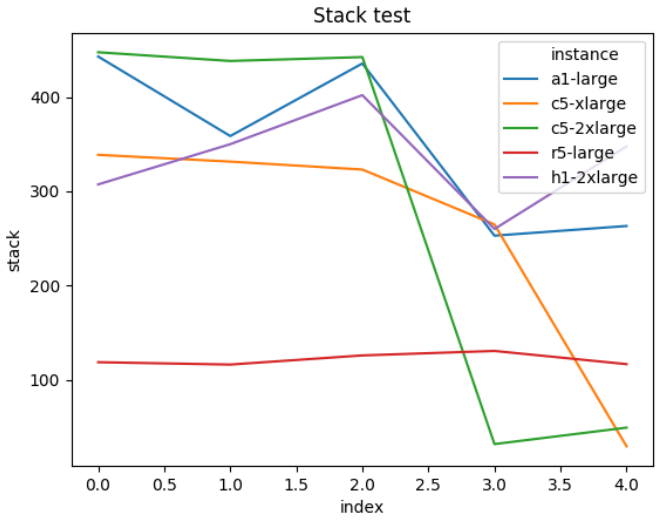


Figure 10



Average values for brk, stack and bigheap have respectively a coefficient of variation of 59.14%, 33.70% and 58.69%, which means that there is variation of performance between instances. In fact, figures 8, 9, 10 show that instance h1.2xlarge is the instance that performs the best overall, outperforming all other instances for the bigheap and brk stressors. Instance r5.large's performance is very surprising. It under-performs all instances for every stressors except for a1.large with bigheap stressor. Being classified as a memory optimized machine, instance r5.large's bad performance does not meet the expectations, which were that it would outperform all other machines.

For the memory performance variation relative to stress applied, table 7 and figures 11, 12, 13 show how the performances varied over 5 iterations.

Table 7: Variation of Memory Performance for Each Instance Over 5 Iterations

Instance	Stressor	Average (ops/s)	Standard Deviation (ops/s)	Coefficient of Variation (%)
a1.large	bigheap	187,004.98	43,709.34	23.37
	brk	350.76	90.91	25.92
	stack	11,805.27	2,344.92	19.86
c5.2xlarge	bigheap	627,648.31	277,832.59	44.27
	brk	281.89	220.56	78.24
	stack	41,439.82	20,001.59	48.27
c5.xlarge	bigheap	390,722.41	166,369.90	42.58
	brk	257.54	130.92	50.84
	stack	20,605.62	10,318.00	50.07
h1.2xlarge	bigheap	828,941.35	44,994.61	5.43
	brk	333.48	53.08	15.92
	stack	46,337.31	12,320.20	26.59
r5.large	bigheap	243,473.37	6,533.79	2.68
	brk	121.52	6.38	5.25
	stack	14,765.91	1,421.66	9.63

Figure 11

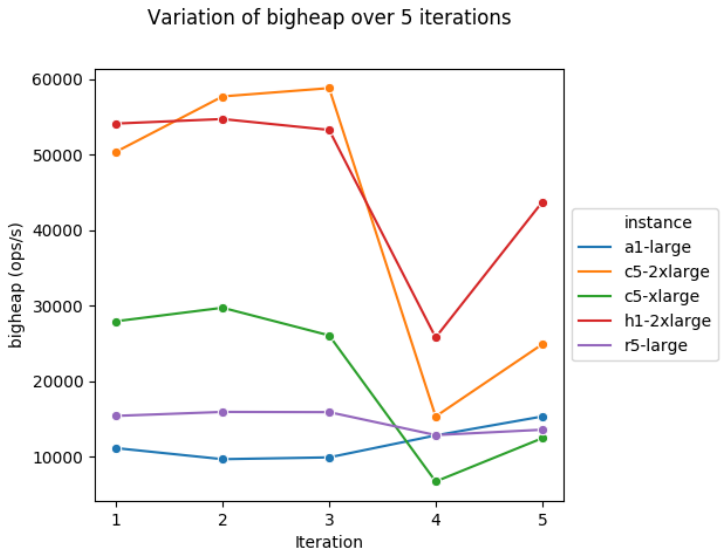


Figure 12

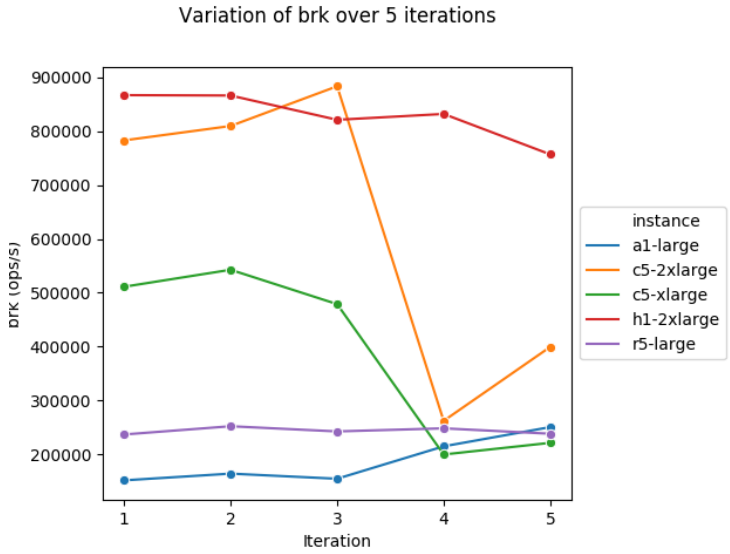
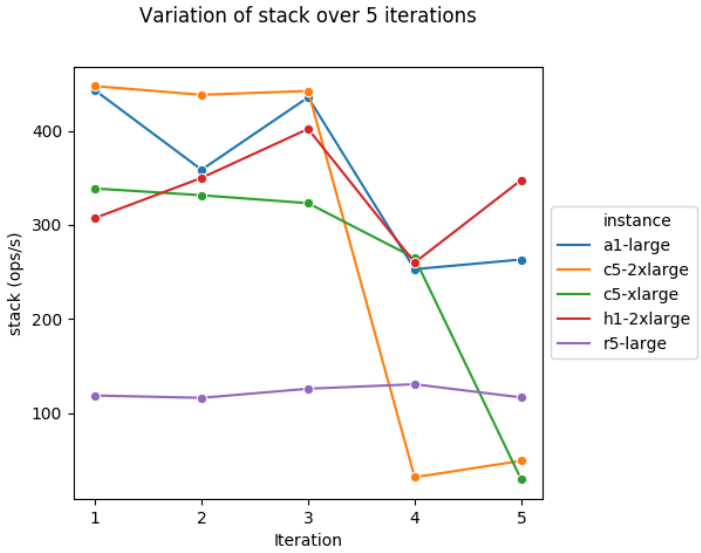


Figure 13

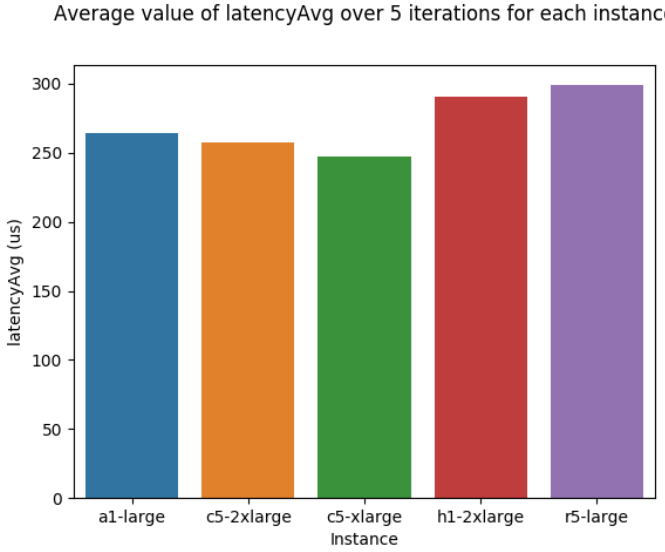


Results show that instance r5.large does not see a high variation in its memory performance over the 5 iterations. While it had the lowest overall performance, it is the instance that is the more stable across levels of memory stress, which makes it an interesting choice for a memory consuming application. Instance h1.2xlarge also keeps a relatively stable performance, compared to instances c5.xlarge and c5.2xlarge. These 2 instances see a drastic loss of performance once they reach the 4th iteration. They are not suited for intense memory work over longer durations but have decent performance for small bursts.

5.5 Disk

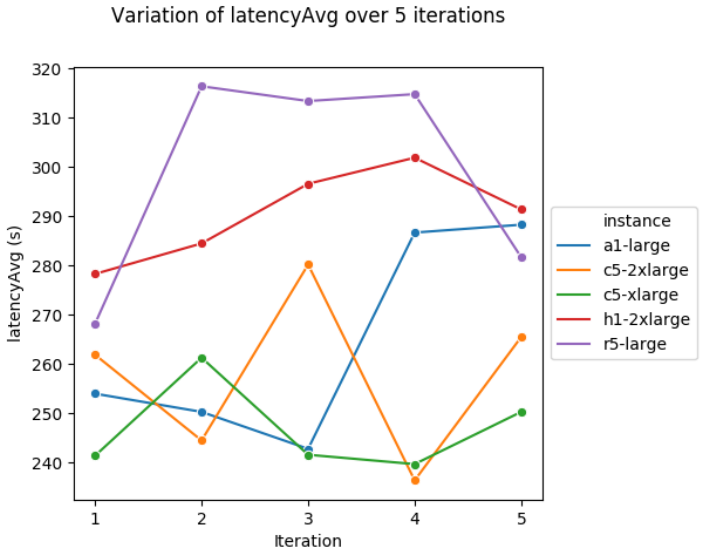
As presented in previous sections, Disk performance was benchmarked according to his latency and his read throughput. Figure 14 shows the average disk latency over 5 executions.

Figure 14



Results show that the instance that performs the best was the c5.xlarge with an average latency of 246.86 us. On the other hand, the instance that has the worst performance is the r5.large with an average latency of 290.54 us. We can clearly see two levels of performance. On one level there is a1.large, c5.2xlarge and c5.xlarge with an average latency of around 250us and on the other, there is h1.2xlarge and r5.large with a latency of around 295 us. The results average standard deviation is 22.14 which means that there was some variation between instances results but not a huge one. It's interesting to point out that the fastest instance was not the Storage Optimized one. Figure 15 illustrates the variation of instances performance over their iterations.

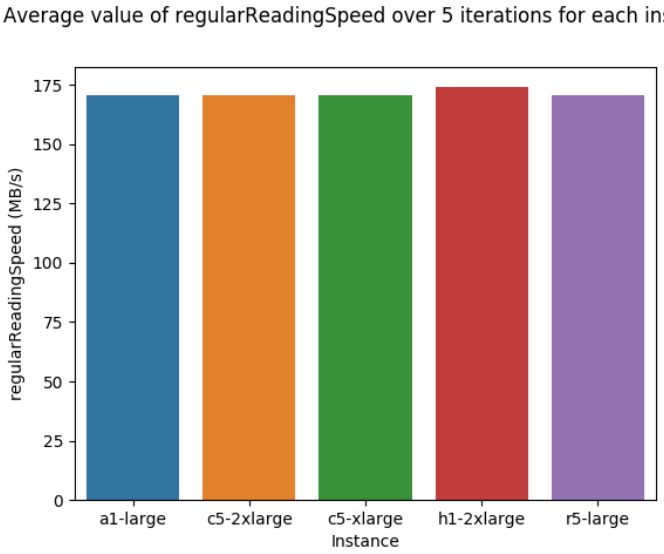
Figure 15



By analyzing the graph we can conclude that the disk average latency is highly variable with an average standard deviation of 15.94. Therefore, we can assume that the stress applied on the instances due to the other benchmarks is strongly affecting the disks latency. This high variability throughout time could be explained by the fact that this benchmark is executed just after the IO throughput test. c5.xlarge and h1.2xlarge were more stable than the others with respective standard variation of 9.07 us and 9.39 us. The others instances, c5.2xlarge, r5.large and a1.large had respective standard variation of 17.4 us, 22.4 us and 21.46 us.

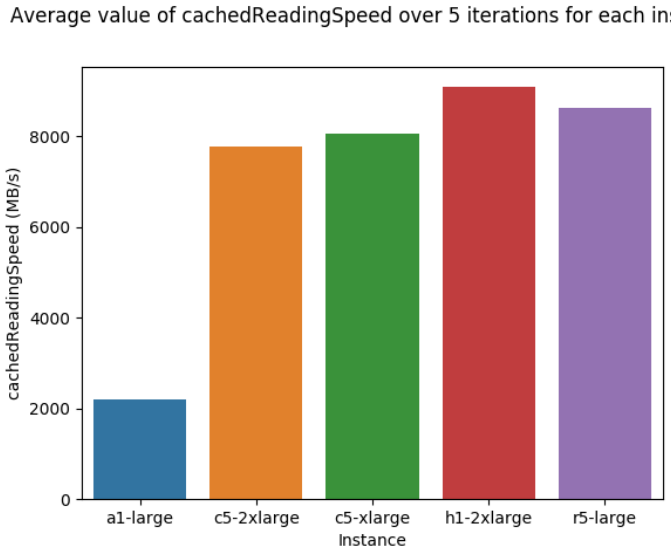
Regarding the disk read throughput, figures 16 and 17 illustrate read throughput performance with and without cache layers respectively.

Figure 16



After analyzing this graph we can conclude that without cache layers, disk read throughput is almost identical throughout instances. The h1.2xlarge was the one with the highest one with a throughput of 173.93 MB/s. The average standard deviation of 1.522 reinforces this point.

Figure 17



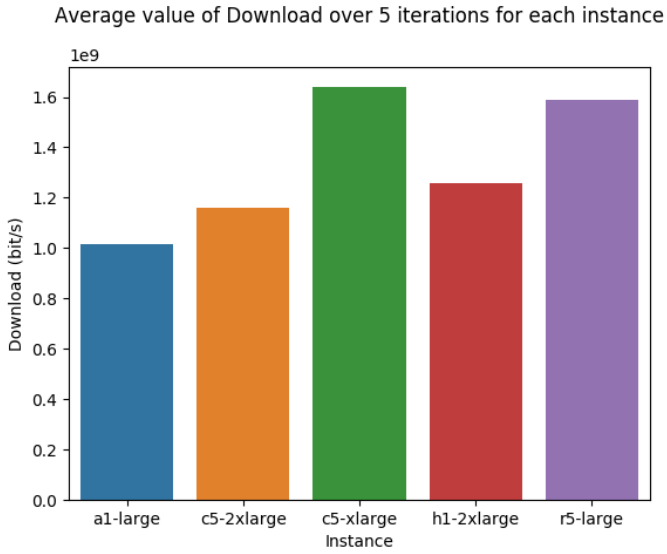
The results regarding the read throughput with caching are more variable. In fact, the average standard variation of these results is 2811.35. We can clearly see the gap that separates the best and the worst instances. The worst instances is the a1.large with a read throughput of 2207.50 MB/s while the h1.2xlarge one is more than three times faster with 9091.11 MB/s. It is interesting

to see the that with caching, the h1.2xlarge instance is almost 53 times faster therefore having a huge impact on performance.

5.6 Network Throughput

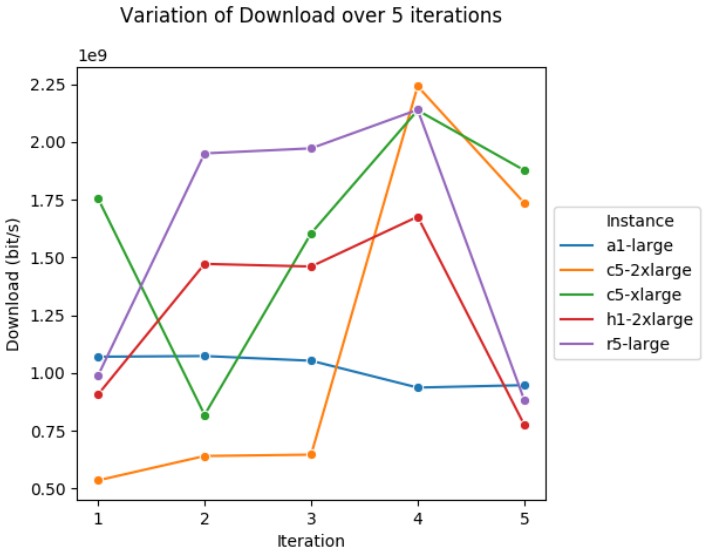
As we discussed in previous sections, the network performance of an instance will be analyzed in regard of the following metrics: download speed and upload speed.

Figure 18



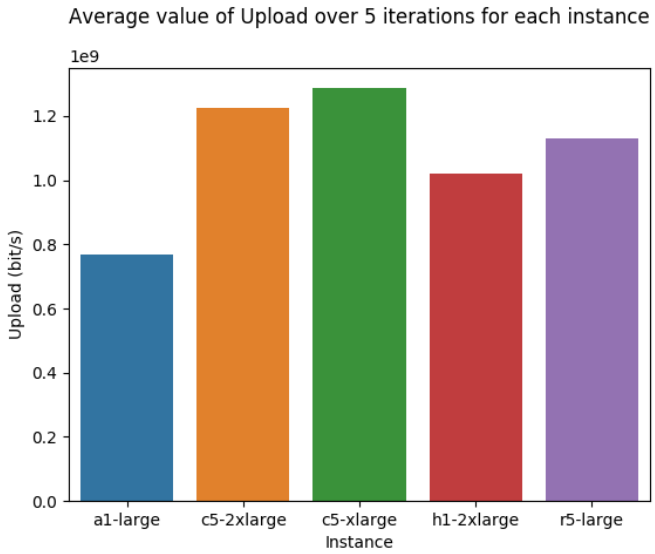
By looking at figure 18, we can clearly state that the c5.xlarge instance has the best download speed with an average download speed of 1.64×10^9 bits/s which is equivalent to 205 MB/s. On the other hand, the a1.large instance is the worst one with an average download speed of 1.02×10^9 bits/s or 127.5 MB/s. These results illustrate a significant gap since the least performing instance is 61% slower than the best one. The average standard deviation is 0.271143343×10^9(33,8875 MB/s), we can therefore conclude that a certain variation exist between instances. It is also worth pointing out that the r5.large instance's performance is close to the c5.xlarge one with a difference of only 7.5 MB/s.

Figure 19



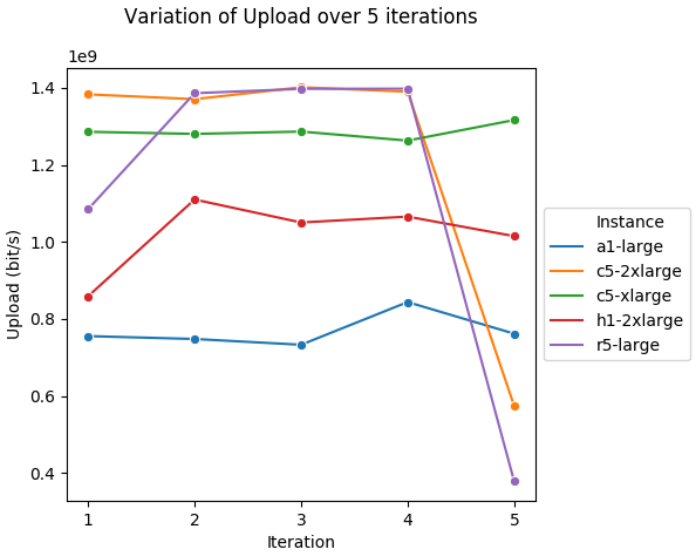
Results from figure 19 show that instances download speed is highly volatile throughout iterations. The average standard deviation is 0.54×10^9 bits/s or 67.33MB/s. The instance that showed the highest variability in its download speed is the c5.2xlarge with a standard deviation of 0.778×10^9 bits/s or 97.37 MB/s. This instance also showed the worst and best download speed. a1.large is the only instance that showed stable results and is also the least performing one. 4 out of 5 instances demonstrated a significant performance decrease between the fourth and the fifth iteration.

Figure 20



By looking at the graphic (figure 20), we can clearly affirm that the c5.xlarge instance is the best instance in regard to upload with an average download speed of 1.29×10^9 bits/s which is equivalent to 161.25 MB/s. On the other hand, the a1.large instance is the worst one with an average upload speed of 7.8×10^8 or 96 MB/s. These results illustrate a significant gap since the least performing instance is 68% slower than the best one. The average standard deviation is 0.2×10^9 or 25.49 MB/s. We can therefore conclude that a certain variation exists between instances. It is also worth pointing out that the c5.2xlarge instance's performance is close to the c5.xlarge one with a difference of only 8.75 MB/s.

Figure 21



Results (figure 21 show that instances upload speed is quite stable throughout iterations. It's worth pointing out that the instances r5.large and c5.2xlarge showed a significant performance drop in their last iteration. The average standard deviation of 0.3×10^9 bits/s or 37.75 MB/s doesn't really reflect the true variation since it is affected by the drop of performance we just discussed.

By analyzing results for both upload and download speeds, we can conclude that the c5.xlarge is the most performing instance. We can also affirm that the a1.large one is the least performing one with respect to the two metrics. One interesting thing to note is the fact that the upload speed showed more stable results than download. Also, the download speed was faster than the upload speed for all of the instances. AWS claims that these instances have network capacity going up to 10 Gb/s while the highest one we recorded was of 2.24 Gb/s.

5.7 Summary

This section aims to provide a visualization of the conclusions that have been exposed in the preceding section. Table 8 illustrates the instances that outperformed others (++) or performed slightly better than the others (+).

Table 8: Summary of Benchmarks Analyses

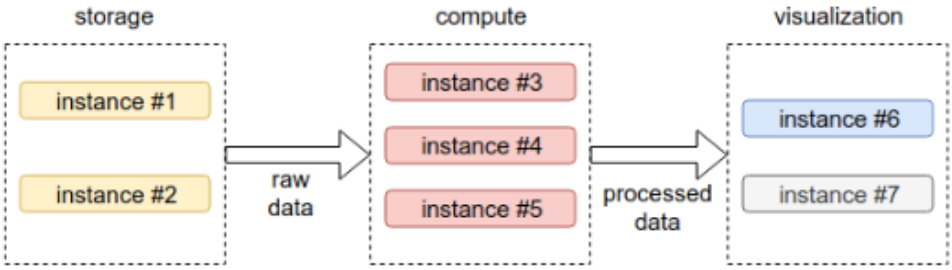
	Metric							
	CPU	IO Throughput	IOPS	Memory	Disk Read Throughput	Disk Latency	Network Download	Network Upload
a1.large	++				+	+		
c5.large	+		++		+	++		++
c5.2xlarge	+	++	+	+	+	+	++	+
r5.large		++		+	+			
h1.2xlarge		+	++	++	+			

6 Results Application

Benchmarking the instances gave meaningful results. Results showed that instances do not always perform as expected from their classification. Using 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 22 shows the way these instances should interact with each other. [2]

Figure 22: Design Architecture Elements [2]



Storage Module

The 2 instances in the storage module need high storage performance. Storage optimized instances can be recognized by their high rate of read and write accesses with low latency. Also, they support very high levels of input/output operations per second. [13]

Following is an overview of characteristics used to evaluate whether or not instances would fit in the storage module, and a recap of the instances performances in each of them.

IO Throughput: Instances that performed well and achieved stability for IO throughput were c5.2xlarge, r5.large and h1.2xlarge.

IOPS: Instances c5.2xlarge and h1.2xlarge, along with instance c5.xlarge, also outperformed other instances in the IOPS benchmarks. However, instance c5.2xlarge tended to lose performance earlier than the 2 others, and should be considered as less stable.

Disk Read Throughput: All instances achieved almost the same disk reading speed.

Disk Latency: Instances c5.xlarge and c5.2xlarge were the best regarding disk latency, but the difference with other instances is less than 60us.

Network Upload: Storage module's need of high network upload performance could be assured by either c5.xlarge or c5.2xlarge, but instance c5.xlarge achieved performance stability across the 5 iterations while instance c5.2xlarge saw its network upload capacity dramatically go down in the 5th iteration.

Using these results, it is recommended to use one c5.xlarge instance and h1.2xlarge instance for the storage module because they offer great performance for storage related characteristics and because they are more stable that others regarding storage related work.

Compute Module

The 3 instances in the compute module need to be able to download a lot of data over the network. They also need to be able to do a lot of input/output operations. They should have great memory performance.

Here is an overview of characteristics used to evaluate whether instances would fit in the compute module or not, and a recap of the instances performances in each of them.

IOPS: Instances c5.2xlarge and h1.2xlarge, along with instance c5.xlarge, also outperformed other instances in the IOPS benchmarks. However, instance c5.2xlarge tended to lose performance earlier than the 2 other, and should be considered as less stable.

Memory: Instance h1.2xlarge performed very well in the memory benchmarking. While instance c5.2xlarge also performed well, it was not stable and loss a lot of performance as the tests progressed. Instance r5.large was the most stable, but showed a poor overall performance.

Cache Reading Speed: Instance h1.2xlarge achieved the best performance regarding cache reading speed, with instances c5.xlarge, c5.2xlarge and r5.large not far behind. Cache reading speed performance is key for data operations.

Network Download: Instances c5.xlarge and r5.large achieved the best performances for download, but instance r5.large showed less stability in its performance. Therefore, instance c5.xlarge should be prioritized.

Network Upload: Instances c5.xlarge or c5.2xlarge performed well, but instance c5.xlarge achieved performance stability across the 5 iterations while instance c5.2xlarge saw its network upload capacity dramatically go down in the 5th iteration.

Because of its cache reading speed, iops and memory performances, two instances h1.2xlarge should be in the compute module. For its network performance and overall good performance in data operation activities, one instance c5.xlarge should also be in the the compute module.

Visualization Module

The visualization module needs 2 instances with good CPU and memory performance because they will need to make heavy computations on processed data.

Here is an overview of characteristics used to evaluate whether instances would fit in the visualization module or not, and a recap of the instances performances in each of them.

CPU: Instance a1.large was the instance that performed the best in the CPU benchmarks, followed by instances c5.2xlarge and c5.xlarge. All instances achieved good performance stability during the benchmarks.

Memory: Instance h1.2xlarge performed very well in the memory benchmarking. While instance c5.2xlarge also performed well, it was not stable and loss a lot of performance as the tests progressed. Instance r5.large was the most stable, but showed poor overall performance.

Cache Reading Speed: Instance h1.2xlarge achieved the best performance regarding cache reading speed, with instances c5.xlarge, c5.2xlarge and r5.large not far behind. Cache reading speed performance is key for data operations.

Network download: Instances c5.xlarge and r5.large achieved the best performance for download, but instance r5.large showed less stability in its performance. Therefore, instance c5.xlarge should be prioritized.

Because this module needs to do a lot of heavy computations, it should be composed of two c5.xlarge. This instance has good CPU, network download and cache reading performances and achieve respectable memory performance.

Summary

Table 9 is a summary of the instances chosen for each module.

Table 9: Chosen Instances for each Module

Module	Instance
Storage - Instance 1	c5.xlarge
Storage - Instance 2	h1.2xlarge
Compute - Instance 3	h1.2xlarge
Compute - Instance 4	h1.2xlarge
Compute - Instance 5	c5.xlarge
Visualization - Instance 6	c5.xlarge
Visualization - Instance 7	c5.xlarge

7 Future Directions

First, due to a limited timeline for this lab, we had to cut corners at some points. The regression analysis is a good example. In order to speed up the execution of our tests, we arbitrarily chose time limits that we thought made sense. For the CPU tests, we simply picked a 10 minutes timeout out of our hats.

Secondly, we think it would have been interesting to push our analysis further with payloads and stress that would be closer to a real application. In our experiments we did our tests sequentially, with a pretty heavy payload, but in a real context, we might have had different results.

Third, there are multiple factors that weren't considered in our analysis. One of them is the price of each instance, which would have a huge impact on the instance choice. It's easy to choose the best instance in terms of performance, but the balance between cost and performance is harder to find. Another factor we could have considered is the time at which we ran our benchmarks. We know that cloud computing is all about sharing resources, which could have induced some more stress on the machines on which our instances were running.

8 Conclusion

In conclusion, the realization of this study helped us achieve a deeper understanding of the virtual machines benchmarking world. We also learned to execute our own experiments in order to build an opinion about what service providers offer instead of blindly trusting what they're telling us. These skills are primordial in today's industry in order to achieve optimal cost versus performance of a cloud architecture. The next step would be to deploy an actual application with such services and see how it performs.

References

[1] Amazon Web Services. (2019) Amazon EC2. [Online] Available: https://aws.amazon.com/ec2/?sc_channel=PS&sc_campaign=acquisition_CA&sc_publisher=google&sc_medium=ACQ-R%7CPS-GO%7CBrand%7CDesktop%7CSU%7CCompute%7CEC2%7CCA%7CEN%7CText%7CHV&sc_content=ec2_p&sc_detail=aws%20ec2&sc_category=Compute&sc_segment=293632235728&sc_matchtype=p&sc_country=CA&s_kwcid=AL!4422!3!293632235728!p!!g!!aws%20ec2&ef_id=CjwKCAiAhp_jBRAxEiwAXbniXW4VNB3YH9HfaHYWdecb0vJMXWoO5hI77E3TBV3QuJ7pgtdhgfhRpBoCK34QAvD_BwE:G:s

[2] S. A. Abtahizadeh, *LOG8415: Lab 1 Selecting VM instances in the Cloud through benchmarking*, LOG8415: Advanced Concepts of Cloud Computing, 2019.

[3] A. Kopytov, SysBench manual, MySQL AB, 2009. [Online]. Available: http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf

[4] B. Martin. (2008) Using Bonnie++ for filesystem performance benchmarking. [Online]. Available: https://www.linux.com/news/using-bonnie-filesystem-performance-benchmarking

[5] UbuntuWiki. (2018) stress-ng. [Online]. Available: https://wiki.ubuntu.com/Kernel/Reference/stress-ng

[6] Mankier. (2019) Speedtest-cli man page. [Online]. Available: https://www.mankier.com/1/speedtest-cli#--bytes

[7] SurveyGizmo. (2018) Regression Analysis. [Online]. Available: https://www.surveygizmo.com/resources/blog/regression-analysis/

[8] Amazon Web Services. (2019) Amazon EC2 Instance Types. [Online]. Available:https://aws.amazon.com/ec2/instance-types/

[9] Amazon Web Services. (2019) Amazon EC2 C5 Instances. [Online]. Available:https://aws.amazon.com/ec2/instance-types/c5/

[10] Amazon Web Services. (2019) Amazon EC2 A1 Instances. [Online]. Available: https://aws.amazon.com/ec2/instance-types/a1/

[11] Amazon Web Services. (2019) Amazon EC2 H1 Instances. [Online]. Available: https://aws.amazon.com/ec2/instance-types/h1/

[12] Amazon Web Services. (2019) Amazon EC2 R5 Instances. [Online]. Available: https://aws.amazon.com/ec2/instance-types/r5/

[13] Amazon Web Services. (2019) Storage Optimized Instances. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/storage-optimized-instances.html

[14] E. Shanks. (2013) Disk Latency Concepts. [Online]. Available: https://theithollow.com/2013/11/18/disk-latency-concepts/

[15] Linux Die. hdparm(8) - Linux man page. [Online]. Available: https://linux.die.net/man/8/hdparm

[16] K. Khlebnikov and K. Kolyshkin. Ioping (1) - Linux Man Pages. [Online]. Available:https://www.systutorials.com/docs/linux/man/1-ioping/