Our Title

Mika V. Mäntylä[1], Foutse Khomh[2], Bram Adams[2], Emelie Engström[3], Kai Petersen[4]
[1] *Dep. of Computer Science and Engineering, Aalto University, Finland*
[2] *SWAT–MCIS, École Polytechnique de Montréal, Québec, Canada*
[3] *Dep. of Computer Science, Lund University, Sweden*
[4] *School of Computing, Blekinge Institute of Technology, Sweden*
mika.mantyla@aalto.fi, {foutse.khomh, bram.adams}@polymtl.ca, emelie.engström@cs.lth.se, kai.petersen@bth.se

***Abstract*—One of the most important challenges to migrate software application to Cloud is to choose the right instance type, which gives the right resources to our application. Our analysis based upon five benchmarks on five Cloud instances provides a difference of performance, helping users to choose the right instance for their applications.**

I. Introduction

Quisque a dignissim purus. Vivamus ut ultrices orci, ac tincidunt magna. Mauris lacinia feugiat dignissim. Donec congue mi risus, in cursus eros adipiscing eu. Proin cursus nulla et sapien vulputate feugiat. Mauris vehicula consectetur ipsum non lobortis. Nunc nec eros fermentum, suscipit purus nec, sagittis metus. Sed vulputate nisl in dapibus condimentum. Aliquam erat volutpat. Proin vitae aliquet ante. Maecenas vulputate nisl ac tellus volutpat, vel pulvinar risus tincidunt. In hac habitasse platea dictumst. Etiam tincidunt sit amet turpis aliquet tristique.

II. Background and Related Work

A. Instances

We have used in our case study five different instances based on Ubuntu. The Table I provides their differences

B. Benchmarks

In this section, we provide the Benchmarks used in our analyzes. We have used *UnixBench* to analyze the CPU performance, *dd if=/dev/zero of=sb-io-test bs=1M count=1k conv=fdatasync* to analyze the IO, ioping to analyze IOPS, Redis to analyze the memory performance, and Dbench to analyze Disk and Network Throughput performances.

1) UnixBench: UnixBench is a benchmark used to evaluate the general performances of a server, specially it is used to measure the CPU. It runs the following tests:

- Dhrystone: This test focus on the operations on the strings.
- Whetstone: This test focus on the operations on the floats.
- Execl Throughput: This test aims at measuring the number of call *execl* that could be done by second.
- Pipe Throughput: This test aims at evaluating the pipes' bandwidth, a process read or write on a pipe.
- Pipe-based Context Switching: This test aims at measuring the communication between process.
- Process Creation:
- Shell Scripts: It represents the execution time of a script by simulating the common operations.
- System Call Overhead:

2) dd if=/dev/zero of=sb-io-test bs=1M count=1k conv=fdatasync: This benchmark is used to compute the performance of reading/writing on a disc, bellow this command : *dd if=/dev/zero of=sb-io-test bs=1M count=1k conv=fdatasync*. The command *dd* is used to copy a file by the selection of just a part of the file. The parameters of this command are:

- if: What we should write, in our example, we write NULL (/dev/zero).
- of: The file name where the command *dd* writes.
- bs: The size of block to write.
- count: The count of blocks to write.
- conv: The conversion to do. For example: *conv=lcase* will convert the character to lower case characters.

Therefore, the command *dd if=/dev/zero of=sb-io-test bs=1M count=1k conv=fdatasync* is used to evaluate the performance of Disk IO. It will copy in the file *sb-io-tests*, 1M x 1k NULL (/dev/zero).

It gives as results, the record-in, record-out, the size copied, the time spent to copy data, and the copy speed.

3) ioping: By the same principle of the command *ping*, which evaluate the network latency, we have the command *ioping* used to measure the monitor I/O latency. However, it does many request, we can limit the number of request by the parameter *-c number_of_requests*. It gives as results the minimum, the average, the maximum time of all requests done.

4) Redis: Redis is an open source, in-memory advanced key-value store with optional persistence to disk. Redis command line interface $redis-cli$ is used to analyze the Redis metrics and access the actual data. Redis includes the redis-benchmark utility which is used to evaluate the performance of memory using simulates running commands done by N clients at the same time sending M total queries. Basic syntax of redis benchmark is shown below:

```
redis-benchmark [option] [option value]
```

Instance type	# virtual CPU	Memory (GiB)	Instance storage	EBS-Optimized Available	Network performance
t2.micro	1	1	EBS Only	No	Low to moderate
t2.small	1	2	EBS Only	No	Low to moderate
t2.medium	2	4	EBS Only	No	Low to moderate
c3.large	2	3.75	2 x 16 (SSD)	No	Moderate
c3.xlarge	4	7.5	2 x 40 (SSD)	Yes	Moderate

Table I: Characteristics of each instance of our case study

When Redis-benchmark is run, lots of tet with specific definitions are done as following:

- PING-INLINE and PING-BULK: This command is often used to test if a connection is still alive, or to measure latency.
- SET: It Set key to hold the string value. If key already holds a value, it is overwritten, regardless of its type. Any previous time to live associated with the key is discarded on successful SET operation.
- GET: It Get the value of key. If the key does not exist the special value nil is returned. An error is returned if the value stored at key is not a string, because GET only handles string values.
- INCR: This comment increments the number stored at key by one. If the key does not exist, it is set to 0 before performing the operation. An error is returned if the key contains a value of the wrong type or contains a string that can not be represented as integer. This operation is limited to 64 bit signed integers.
- LPUSH: This comment insert all the specified values at the head of the list stored at key. If key does not exist, it is created as empty list before performing the push operations. When key holds a value that is not a list, an error is returned. It is possible to push multiple elements using a single command call just specifying multiple arguments at the end of the command. Elements are inserted one after the other to the head of the list, from the leftmost element to the rightmost element.
- LPOP: It removes and returns the first element of the list stored at key.
- SADD: Add the specified members to the set stored at key. Specified members that are already a member of this set are ignored. If key does not exist, a new set is created before adding the specified members. An error is returned when the value stored at key is not a set.
- SPOP:This command removes and returns a random element from the set value stored at key.
- LRANGE: Returns the specified elements of the list stored at key. The offsets start and stop are zero-based indexes, with 0 being the first element of the list (the head of the list), 1 being the next element and so on. These offsets can also be negative numbers indicating offsets starting at the end of the list. For example, -1 is the last element of the list, -2 the penultimate, and so on. Note that if you have a list of numbers from 0 to 100, LRANGE list 0 10 will return 11 elements, that is, the rightmost item is included.
- MSET: MSET replaces existing values with new values, just as regular SET. See MSETNX if you don't want to overwrite existing values. MSET is atomic, so all given keys are set at once. It is not possible for clients to see that some of the keys were updated while others are unchanged.

5) Dbench: Dbench is a Filesystem benchmark which generates load patterns, but without requiring a lab of Windows load generators to run. It is used to calculate the performance of Disk and Network Troughput. Dbench is a utility to benchmark a system based on client workload profiles.

Dbench takes only one parameter on the command line, which is the number of processes (clients) to start. Issuing the command dbench 30 for example, creates 30 parallel processes of dbench. All processes are started at the same time and each of them runs the same workload. The workload for each dbench process is specified by a client.txt configuration file in the working (testing) directory. It consists of a mixture of file system operations executed by each dbench process. dbench runs with n parallel processes and delivers only one value as a result. The resulting value is an average throughput of the file system operations described in client.txt and measured in megabytes per second.

It can also take the following options:

- -c client.txt : Use this as the full path name of the client.txt file (the default is $/usr/share/dbench/client.txt$).
- -s: Use synchronous file IO on all file operations.
- -t TIME: set the runtime of the benchmark in seconds (default 600)
- -D DIR: set the base directory to run the filesystem operations in
- -x: enable xattr support, simulating the xattr operations Samba4 would need to perform to run the load
- -S: Use synchronous IO for all directory operations (unlink, rmdir, mkdir and rename). The tbench program takes a number, which indicates the number of clients to run simultaneously, and a server name: $tbench_srv$ should be invoked on that server before invoking tbench. tbench can also take the following options:
- -c loadfile: Use this as the full path name of the client.txt file (the default is $/usr/share/dbench/client.txt$).
- -T option[,...]: This sets the socket options for the connection to the server.

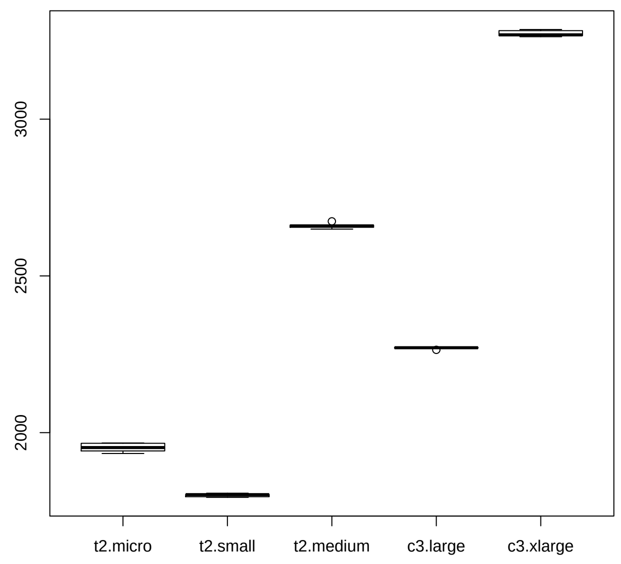


Figure 1: UnixBench results

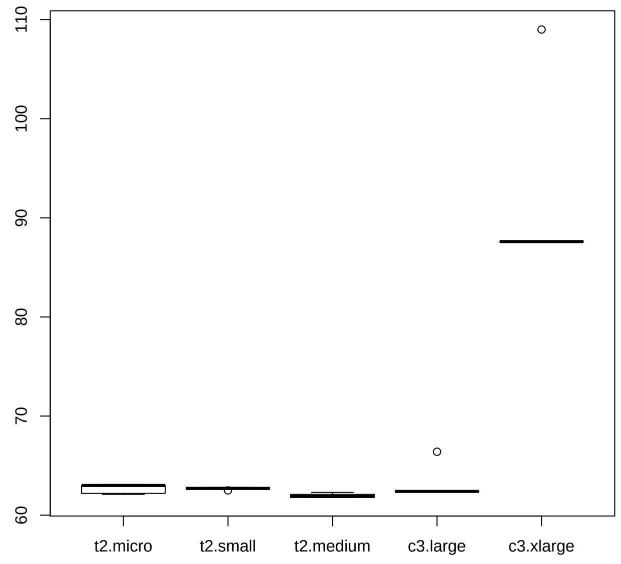


Figure 2: IO benchmark results

III. Approach

In this section, we provide the approach used to extract data. While the first step consists of creating the instances, the second one consists of computing data.

We have created the first instance of type *t2.micro*, we have also created a script to compute data. Therefore, to create another instance, we have used the image of the first one. Hence, we have created the types of instances consisting of *t2.small*, *t2.medium*, *c3.large*, and *c3.xlarge*.

To compute data, we have used the benchmarks described above. Since the performance of Cloud instances change over time, because there are many instances deployed in the same hardware, we execute each one five times.

We have extracted data in csv form, as described in the Figure .... .

IV. Results

In this section, we provide the benchmarks results.

A. CPU performances

Results. The averages of UnixBensh results for each instance are: 1952.12 for the instance *t2.micro*, 1800.18 for the instance *t2.small*, 2660.04 for the instance *t2.medium*, 2269.7 for the instance *c3.large*, and 3273.46 for the instance *c3.xlarge*.

As highlighted by Figure 1. The minimum value of *UnixBench* benchmark results for the instance *t2.micro* is *1933.5*, whereas the maximum is *1967*. The minimum for the instance *t2.small* is *1793.6* and the maximum is *1806.8*. *t2.medium* provides as a minimum result of the same benchmark *2649.2*, where the maximum is *2674.1*. The minimum value of *c3.large* is *2264.2* where the maximum is *2271.8*. *c3.xlarge* provides a minimum of *3263.1* and a maximum of *c3.xlarge*.

The smallest value of *UnixBench* benchmark is provided by the instance *t2.small*, and the maximum is provided by the instance *c3.xlarge*.

As provided by Figure 1, the order of instances by the *UnixBench* benchmark is as follow: *t2.small, t2.micro, c3.large, t2.medium, c3.xlarge*.

Discussion. We observe that we have three categories of instances in Figure 1. We can classify the instances in three main clusters. While the first one is represented by the instances *t2.micro* and *t2.small*, which have one virtual processor. The second category is presented by *t2.medium* and *c3.large*, which contains two virtual processor. The third category contains the instance *c3.xlarge*, it contains 4 virtual CPU. Hence, the difference between the three category is due to the number of virtual processor.

Moreover, There is a difference between the instances *t2.micro* and *t2.small*. By analyzing the benchmark results, we have found that the main difference is in the test *File Copy 4096 bufsize 8000 maxblocks*, which consists of copying a large size, which gives the important different score (around 5910.3 for the instance *t2.micro* and 4998.4 for the instance *t2.small*).

The difference between *t2.medium* and *c3.large* is principally due to the test *Dhrystone 2 using register variables*. It tests the operations on the strings. In this test, *t2.medium* has a score of 5449.3, whereas *c3.large* has a score of 3408.7.

t2.micro	t2.small	t2.medium	c3.large	c3.xlarge
386 us	385 us	409 us	285 us	242 us
424 us	394 us	365 us	273 us	241 us
423 us	364 us	303 us	287 us	233 us
404 us	394 us	328 us	269 us	224 us
433 us	380 us	298 us	280 us	250 us

Table II: IOPS min results

t2.micro	t2.small	t2.medium	c3.large	c3.xlarge
469	480	457	372	336
473	447	475	329	352
515	487	452	359	357
495	475	518	380	321
500	483	460	357	312

Table III: IOPS average results

t2.micro	t2.small	t2.medium	c3.large	c3.xlarge
57	81	28	36	39
33	43	84	32	50
43	88	66	39	75
57	58	165	43	39
48	52	79	34	26

Table V: IOPS MDev results

t2.micro	t2.small	t2.medium	c3.large	c3.xlarge
196.735	232.123	255.073	287.458	354.273
196.843	231.064	259.235	287.348	350.399
197.842	232.38	256.889	287.161	353.942
197.558	230.762	255.986	286.702	353.547
196.474	231.347	255.956	46.8487	355.52

Table VI: DBench benchmark results (MB/sec)

B. IO performances

Results. The averages of IO Benchmark results for each instance are: 62.68 for the instance *t2.micro*, 62.7 for the instance *t2.small*, 61.98 for the instance *t2.medium*, 63.2 for the instance *c3.large*, and 91.88 for the instance *c3.xlarge*

As highlighted by Figure 2. The minimum value of *IO* benchmark results for the instance *t2.micro* is *62.1 MB/s*, whereas the maximum is *63.1 MB/s*. The minimum for the instance *t2.small* is *62.5 MB/s* and the maximum is *62.8 MB/s*. *t2.medium* provides as a minimum result of the same benchmark *61.8 MB/s*, where the maximum is *62.3 MB/s*. The minimum value of *c3.large* is *62.4 MB/s* where the maximum is *66.4 MB/s*. *c3.xlarge* provides a minimum of *87.6 MB/s* and a maximum of *109 MB/s*.

The smallest value of *IO* benchmark is provided by the instance *t2.medium*, and the maximum is provided by the instance *c3.xlarge*.

As provided by Figure 2, the order of instances by the *IO* benchmark is as follow: *t2.medium, c3.large, t2.small, t2.micro, c3.xlarge*.

Discussion. From the comparison provided in Figure 2, the instance *c3.xlarge* has the highest speed of reading/Writing on disk. We refer these results to the I/O Performances, where the instance *c3.xlarge* has ”Moderate/500Mbps”, and the other instances have ”Low to moderate” or ”Moderate”.

C. IOPS performances

Table II, Table III, Table IV, and Table V provide the results of the benchmark used to evalute the IOPS performance. While we have used the benchmark ioping to evaluate the performance of IOPS of each instance, the benchmark gives four different results, the first one corresponds to minimum time in the results of one execution, which highlighted in II. The second one corresponds to the average, provided by Table III. The third one represents the maximum time shown by Table IV. Whereas, the third one presented by Table V is MDev.

t2.micro	t2.small	t2.medium	c3.large	c3.xlarge
568	620	497	419	393
532	504	652	391	453
594	665	559	406	544
615	593	967	423	379
565	574	592	401	353

Table IV: IOPS Max results

As provided in Figure 3a, each instance has its own minimum time to provide an IOPS result, the instances are ordered as follow: *c3.xlarge, c3.large, t2.medium, t2.small, and t2.micro*.

As provided in Figure 3b, each instance has its own average time to provide an IOPS result, the instances are ordered as follow: *c3.xlarge, c3.large, t2.medium, t2.small, and t2.micro*.

As provided in Figure 3c, each instance has its own maximum time to provide an IOPS result, the instances are ordered as follow: *c3.xlarge, c3.large, t2.micro, t2.small, and t2.medium. t2.micro, t2.small, and t2.medium* have approximately the same value as a mediam.

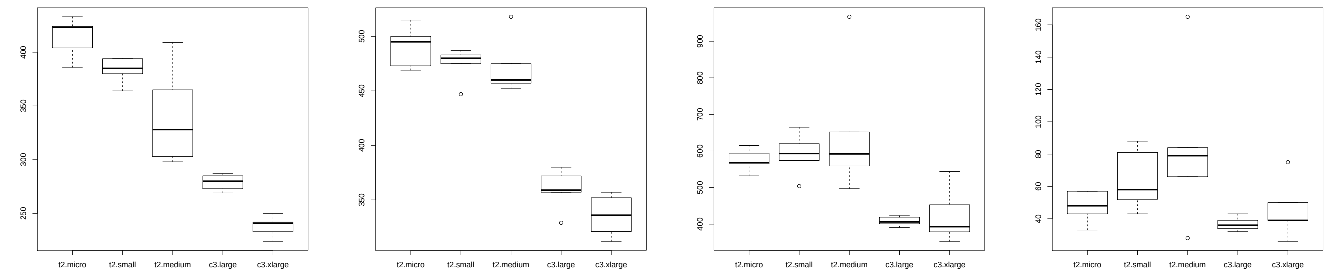
As provided in Figure 3d, each instance has its own MDev value, the instances are ordered as follow: *c3.xlarge, c3.large, t2.micro, t2.small, and t2.medium*.

D. Disk performances

The averages of DBench results for each instance are: 197.0904 MB/sec for the instance *t2.micro*, 231.5352 MB/sec for the instance *t2.small*, 256.6278 MB/sec for the instance *t2.medium*, 239.10354 MB/sec for the instance *c3.large*, and 353.5362 MB/sec for the instance *c3.xlarge*. The results are provided in Table VI

As provided by Figure 4, the order of instances by the disk performance is as follow: *t2.micro, t2.small, t2.medium, c3.large, and c3.xlarge*.

Discussion. The main differences between the instances *t2.micro* and *t2.small* are the tests *Find*, which consists of finding a file in the disk. Where the second important difference we have found is the test *ReadX* related to the read file operations. Moreover, the important difference between *t2.small* and *t2.medium* is *Flush* test, which is a test related to the file operations. The same difference between *t2.medium* and *c3.large*. The main differences between *c3.xlarge* and *c3.large* are *ReadX* and *WriteX*, which are related to read/write on a file.



(a) Min (b) Average (c) Max (d) MDev

Figure 3: IOPS benchmark comparison of results (us)

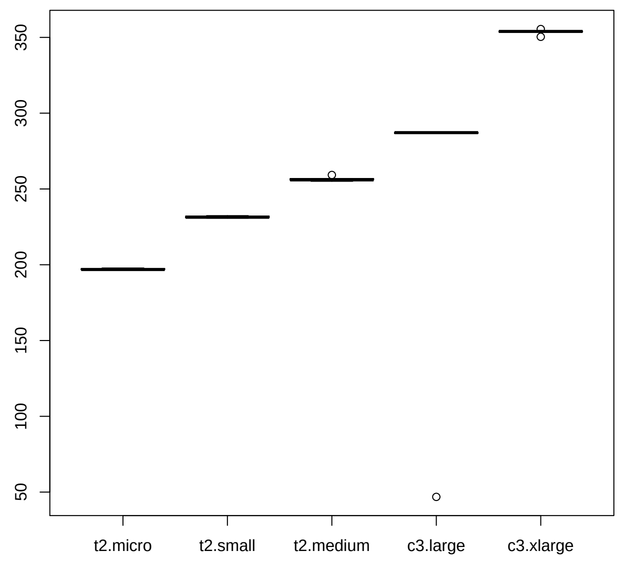


Figure 4: Difference between instances in Disk performances

t2.micro	t2.small	t2.medium	c3.large	c3.xlarge
4.63978	368.66	377.072	455.738	709.535
4.86367	236.93	268.571	294.85	337.944
4.86856	238.627	266.551	295.228	342.986
4.9142	239.147	269.105	298.469	327.806
4.88602	234.173	264.574	300.31	20.7406

Table VII: DBench benchmark results (MB/sec)

E. Network Throughput performances

The averages of DBench results for each instance are: 4.834446 for the instance *t2.micro*, 263.5074 for the instance *t2.small*, 289.1746 for the instance *t2.medium*, 328.919 for the instance *c3.large*, and 347.80232 for the instance *c3.xlarge*. The results are provided in Table VII

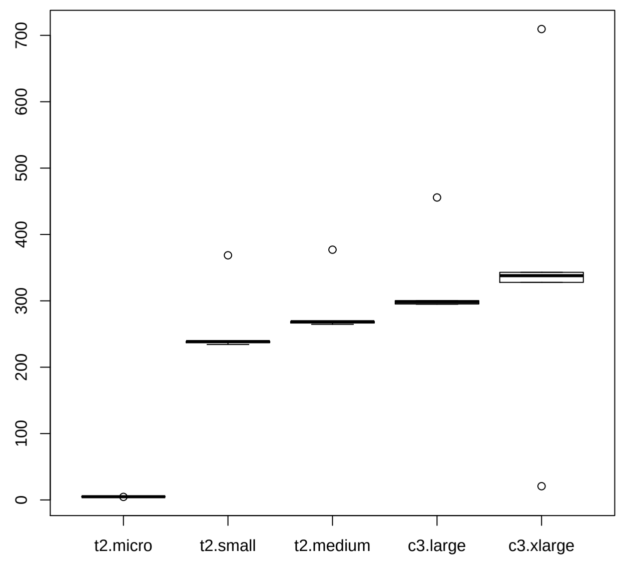


Figure 5: Difference between instances in Disk performances

As provided by Figure 5, the order of instances by the disk performance is as follow: *t2.micro, t2.small, t2.medium, c3.large, and c3.xlarge*.

V. Discussion

VI. Comparison of Results

VII. Conclusion

Acknowledgment

TODO