Exploiting Hardware Heterogeneity within the Same Instance Type of Amazon EC2

Zhonghong Ou[†], Hao Zhuang[†], Jukka K. Nurminen[†], Antti Ylä-Jääski[†], *Pan Hui*[‡]
[†]*Aalto University, Finland;* [‡]*Deutsch Telekom Laboratories, Germany*

Abstract

Cloud computing providers might start with near-homogeneous hardware environment. Over time, the homogeneous environment will most likely evolve into heterogeneous one because of possible upgrades and replacement of outdated hardware. In turn, the hardware heterogeneity will result into performance variation. In this paper, we look into the hardware heterogeneity and the corresponding performance variation within the same instance type of Amazon Elastic Compute Cloud (Amazon EC2). Standard large instance is selected as the example. We find out that there exist three different sub-types of hardware configuration in the standard large instance. Through a set of detailed micro-benchmark and application-level benchmark measurements, we observe that the performance variation within the same sub-type of instance is relatively small, whilst the variation between different sub-types can be up to 60%. By selecting better-performing instances to complete the same task, end-users of Amazon EC2 platform can achieve up to 30% cost saving.

1 Introduction

Cloud computing attracts a significant amount of attention from industry, academia, and media because of its on-demand, pay-as-you-go, etc, characteristics. As a representative and one of the most widely adopted public cloud platforms, Amazon Elastic Compute Cloud (Amazon EC2) has been used for a host of small and medium-sized enterprises (SMEs) for various usages. Amazon EC2 was introduced in 2006, and supports a wide arrange of instance types. Naturally, these different types of instances are likely hosted by heterogeneous hardware. Over time, because of hardware upgrade and replacement, it would be interesting to investigate the following issues:

(1) Does the same type of instance utilize homogeneous or heterogeneous hardware configuration?

(2) If heterogeneous hardware is used, what is the resulting performance variation?

In this paper, we try to answer the aforementioned two questions by utilizing the standard large instance type, i.e. *m1.large*. Similar results are observed for the other types of instances within the same standard family, including small (*m1.small*), and extra large (*m1.xlarge*) instances. Our contributions are as follows:

(1) We observe that within the same instance type, Amazon EC2 uses heterogeneous hardware to host the instances.

(2) The variation of the same sub-type of instances, i.e. hosted by identical hardware, is relatively small, whilst the variation among different sub-types of instances , i.e. hosted by heterogeneous hardware, can reach up to 60%.

(3) Compared with taking the random instances assigned by Amazon EC2 platform, by selecting better-performing instances to complete the same task, EC2 users can acquire up to 30% of cost saving.

The rest of the paper is structured as follows. In Section 2, we present background and related literature of Amazon EC2 study. Section 3 details the micro-benchmark measurements and application-level benchmarks. Section 4 analyzes the potential cost saving for EC2 end-users. In Section 5 we conclude the paper and present ideas for future work.

2 Related Work

Several studies have been conducted to analyze various aspects of Amazon EC2. Garfinkel [4] conducted a measurement study of various Amazon Web Services (AWS) to evaluate the feasibility and cost of moving a large-scale research application from localized server to Amazon offering. Palankar et al. [8] performed measurements focusing on Amazon S3 to testify its ability to provide stable storage support for large-scale scientific

computation application. Walker [12] studied the performance of Amazon EC2 high-performance cluster compute instances against a locally configured equivalent processors cluster, and showed that there exists a performance gap between the EC2 provisioned cluster and local traditional scientific cluster. Wang et al. [13] presented a measurement study on the impact of virtualization on Amazon EC2 platform. Their findings indicated that virtualization causes instability and variation to network throughput and packet delay. Li et al. [7] developed a performance and cost comparator, i.e. CloudCmp, to measure cloud services from different cloud providers. Their study demonstrated that there was no single winner who outperformed the other counterparts in all aspects of its cloud service offerings. Cooper et al. [2] developed Yahoo! Cloud Serving Benchmark (YCSB) framework to facilitate performance comparison. Barker et al. [1] analyzed the impact of virtualization on the performance of latency sensitive applications in the cloud.

Furthermore, in exploiting heterogeneity in the cloud, there exist several studies. Suneja et al. [10] proposed to use Graphics Processing Uint (GPU) acceleration to speed up cloud management tasks in Virtual Machine Monitor (VMM). Lee et al. [6] introduced a scheduling mechanism in the cloud that takes into consideration heterogeneity of the underlying platform and workloads. Through mathematical modeling, Yeo et al. [14] found out that in order to achieve optimal performance, the performance variation among a heterogeneous cloud infrastructure should be no larger than three times. To the best of our knowledge, there is no work focusing on exploiting the heterogeneity within the same instance type of Amazon EC2, which motivates our work in this paper.

3 Micro-benchmark

In this section, we first analyze the hardware configuration of Amazon EC2. Then we utilize several micro-benchmark tools to evaluate the performance of various sub-types of instances. Specifically, standard large instance (*m1.large*) is selected as the representative for performance evaluation.

3.1 Hardware Configurations of EC2

We acquire the hardware information of Amazon EC2 instances by using *cpuid* command, a non-trapping instruction that can be used in user mode without triggering trap to the underlying processor. Thus, the hypervisor does not capture the instruction and return modified results. Furthermore, we run *cat /proc/cpuinfo* command to verify the results from *cpuid*. The CPU models from both sources are identical, and the results are listed in Table 1. It is noteworthy that we only list the *standard* instance family in Table 1. Diversified hardware is also used in *high-CPU* instance family (*c1.medium* and *c1.xlarge*). We exclude them due to space limit. Furthermore, the high-memory instances use identical Intel X5550 processors, and the cluster compute and cluster GPU instances both use Intel Xeon X5570 processors.

Table 1: Hardware configuration

Instance type	CPU model	%(2011)	%(2012)
m1.small	E5507	45%	12%
	E5430	34%	38%
	E5645	3%	30%
	2218HE	18%	20%
m1.large	E5507	58%	40%
	E5430	29%	17%
	E5645	5%	42%
	2218HE	4%	1%
	270	4%	-
m1.xlarge	E5507	31%	6%
	E5430	27%	46%
	E5645	40%	48%
	270	2%	-

We collected hardware information within two periods of time to investigate the hardware changes from hardware upgrade or replacement. One period is from April through July in 2011; the other one is from January through March in 2012. For each period, we collect hardware information of 200 instances for each instance type, covering all availability zones in the US (Virginia) east region. The percentage of each CPU model is shown in "%(2011)" and "%(2012)" columns, respectively. The "2218HE" and "270" models are from AMD Opteron series, whilst the rest are from Intel Xeon series. From Table 1, it is clearly shown that newer processor models are replacing older ones gradually, whilst the older ones are likely used for smaller instances in the same instance family. For example, in *m1.large* instance, the AMD Opteron 270 (released in 2005) processor that was found in 2011 is no longer accessible in 2012, whilst the Intel Xeon E5645 (released Q1'10) CPU model is more frequently accessible in 2012 than in 2011. This trend is similar in all standard (including *m1.small*, *m1.large*, and *m1.xlarge*) and high-CPU (including *c1.medium* and *c1.xlarge*) instances.

Furthermore, we notice that the probability of a specific type of processor, e.g. E5645, significantly varies in different availability zones. In one availability zone, we can acquire 95% of instances hosted by E5645 machines, whilst in another zone, the probability of E5645 instances is as low as 10%. We conjecture that the availability zone with 95% of E5645 machines is a newly built

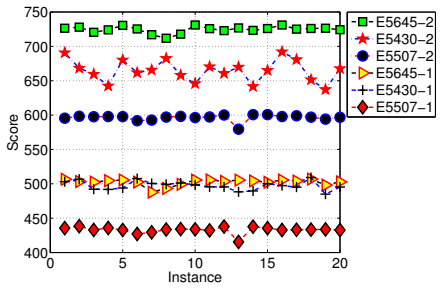

Figure 1: UnixBench score, one and two processes

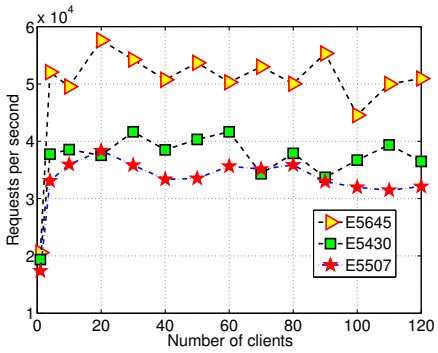

Figure 2: Redis SET operation

data center within the US east region. The interesting question to ask is whether the heterogeneous hardware configuration within the same instance type leads to diversified performance. We select the *m1.large* instance as the example to evaluate performance because this instance has a relatively large amount of memory and can be used in various general applications.

3.2 Micro-benchmarks

We use several micro-benchmark tools to measure the performance of *m1.large* instance, including UnixBench [11] to measure the CPU, Redis [9] to measure the memory, and Dbench [3] to measure the disk subsystems. To provide apples-to-apples comparison, we use the same Amazon Machine Image with CentOS5.6 in all the instances tested. The benchmark is the only process running when we conduct the measurements.

CPU performance: UnixBench [11] utilizes multiple tests to measure various aspects of the system's performance, primarily CPU's performance. The test results are compared to the baseline system to produce an index value. The entire set of index values are then combined to make a composite index for the system. To measure the likely diversity of instances from the same hardware configuration, we choose 20 instances from each subtype of instance, i.e. E5507, E5430, and E5645. The results of the UnixBench benchmark are shown in Fig. 1. The figure clearly demonstrates that the differences amongst the same sub-type of instances, e.g. E5507, is small, whilst the differences between different sub-types are significant. If one process is running, E5430 and E5645 are comparable in terms of performance, whilst they are approximately 1.15 times of the performance of E5507. When two processes are running, E5645 outperforms E5430, whilst E5430 further outperforms E5507. The performance variation in times is 1.21, and 1.1 times for E5645, and E5430, respectively, wherein E5507 is taken as the baseline.

Memory performance: Redis [9] is an in-memory key-value store that has the benchmark utility to simulate multiple concurrent clients to send requests (e.g. SET, and GET) at the same time. In our measurements, we perform 100,000 requests and vary the number of concurrent clients. Random key is used to perform the operations. The detailed results from GET operations are depicted in Fig. 2. The results from other operation are similar to GET operation. Similar to Fig. 1, in memory operations, E5645 instances outperform E5430 and E5507 instances. The memory performance of E5645 is 1.5 times of that of E5507, whilst E5430 is 1.14 times of E5507.

Disk performance: The results from Dbench [3] show similar trends as the Unixbench, and Redis. E5645 instances can provide disk throughput 1.25 times as high as E5507 instances, whilst E5430 provides comparable disk throughput as E5507.

3.3 Application-level Benchmark

We use Httperf [5] to measure the Web server throughput. Dynamic HTTP request is used to make the processor busy. Dynamic request means after receiving a request from a client, the Web server performs a mathematical summation from 1 through 100, and then returns the result to the client. Thus, the dynamic Web test is CPU-bound rather than network-bound. To try to avoid potential bottleneck from client machine, we use a high-CPU medium instance from the same zone acting as the client. The Httperf throughput results are depicted in Fig. 3. The figure demonstrates that the advantages from separate subsystems, e.g. CPU, memory and disk, are accumulated at application-level, where E5645 is 1.6 times as efficient as E5507 and E5430 is 1.2 times as E5507.

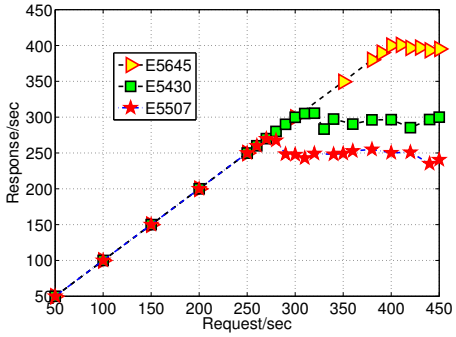

Figure 3: Httperf performance

Table 2: Notations

Notation	Definition
f	Hourly cost of an instance
h	Number of hours to run
m	Number of different instances
n	Number of instances needed with worst performance
p_i	Probability of instances hosted with a specific hardware
x_i	Performance variation compared to the baseline instance
C	The total cost

4 Cost Analysis

Now we are aware that there exists various hardware configuration in the same instance type. We analyze the potential cost saving by seeking for the best-performing instances in the same instance type. The worst-performing instance is used as the baseline, the other instances are x (no less than 1) times as fast as the baseline instance. We use the notations defined in Table 2.

Given the same amount of task (computation, communication etc), with better-performing instances, the task can be completed with two alternatives: (1) smaller number of instances running for the same amount of time; (2) same number of instances running for shorter period of time. From the cost perspective, these two alternatives are the same. We take the first alternative as the example. The expected value of the performance of a random instance is defined as follows:

$$E(X) = \sum_{i=1}^{m} x_i * p_i \tag{1}$$

The total cost of completing the task, equivalent to $n*h$ hours' work, using random instances can be deduced as follows:

$$C_{random} = n*h*f/E(X) \tag{2}$$

If we aim to select the best-performing instances to complete the task, the cost of this optimized scenario is:

$$C_{opt} = n*h*f/x_{opt} \tag{3}$$

Furthermore, the "trial and error" testing process results in extra cost for the optimized scenario. As in Amazon EC2, the less than one hour usage is rounded up to and charged as one hour. Thus, the extra cost of finding n best-performing instances is:

$$C_{extra} = n*f/p_{opt} \tag{4}$$

Here we assume that the test of finding one fast instance takes no more than one hour and the jobs are relatively small to the population of available servers. As a matter of fact, we can simply request for one instance from Amazon, then inspect its *cpuid*. If the instance is not the best-performing one, we simply discard it and request for another one. The potential cost saving is:

$$C_{saving} = C_{random} - C_{opt} - C_{extra} \tag{5}$$

Put Eq. 1, Eq. 2, Eq. 3, and Eq. 4 in Eq. 5, we can deduce the following equation:

$$C_{saving} = (h/(\sum_{i=1}^{m} x_i * p_i) - h/x_{opt} - 1/p_{opt)} * n * f \tag{6}$$

Understandably, if one fast instance is able to acquire cost gain, the total cost gain achievable from multiple instances grows linearly with the number of instances. This is also applicable to the price of the instance.

Again, take the *m1.large* instance as the example. There are three different sub-types of instances, E5430, E5507, and E5645. The probability of each subtype of instance is 17%, 40%, and 42%, respectively. The unit cost of a regular *m1.large* instance (excluding reserved instances and spot instances) is \$0.34/hour. The worst-performing instance is E5507, thus it is taken as the baseline. On average, E5430 and E5645 is 1.1 and 1.4 times, respectively, as fast as E5507. Put all these values in Eq. 6, we can acquire the following equation:

$$C_{saving} = 0.34 * n * (0.1368 * h - 2.38) \tag{7}$$

In order to achieve cost saving, the requirement is $C_{saving} > 0$, then we can get the necessity: $h > 17.4$. That is to say, given the aforementioned probability of each subtype of instance and its respective performance, it starts to make sense from cost perspective to select E5645 instances to complete the task if the required time is larger than 18 hours.

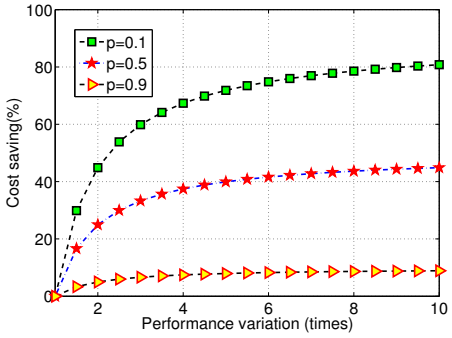

Figure 4: Cost saving analysis

If we have a task requires 100 E5507 comparable *m1.large* instances to complete in a year (24hours/day*365days/year=8760 hours), the potential cost saving for the whole year is $40664, a 16% cost saving in percentage.

Recall from section 3.1 that different hardware is not distributed uniformly among all the availability zones, but rather in some zone one type of hardware dominates the whole zone, whist in another zone, another type of hardware dominates. Thus, it would also be interesting to analyze two types of hardware (e.g. E5507 and E5645) and investigate the maximum cost saving achievable. The result is depicted in Fig. 4, wherein p stands for the probability of the fast instances (e.g. E5645), and x-axis stands for the performance variation in times.

Understandably, if the fast instances account for the majority of the overall instances, e.g. $p = 0.9$, without a selection process, the probability of acquiring a fast instance is very high. Thus, the performance is close to the optimal situation with the selection process, and the cost saving achievable is trivial. However, as the fast instances account for less proportion of the overall instances, the cost saving achievable is becoming significant. In the case of $p = 0.1$, if the fast instance is 10 times as fast as the slow instance, the cost saving is as high as 80%. Obviously, this is an unrealistic situation with all the efforts Amazon contributes to make the same type of instances function closely. From section 3.2 and 3.3, we know that 1.2-1.6 times variation is highly possible. With 1.5 times variation, the achievable cost saving can reach 30%. For SMEs, which are the major customers of Amazon EC2 platform, this saving has a big impact.

5 Conclusions

In this paper, we investigated the hardware heterogeneity within the same instance type of Amazon EC2. Standard large instance (*m1.large*) was taken as the example. Through two periods of several-month measurements in 2011 and 2012, we found out that Amazon EC2 uses diversified hardware to host the same type of instance. The hardware diversity results in performance variation. In general, the variation between the fast instances and slow instances can reach 40%. In some applications, the variation can even approach up to 60%. By selecting fast instances within the same instance type, Amazon EC2 users can acquire up to 30% of cost saving, if the fast instances have a relatively low probability. In the future, we plan to investigate the scheduling mechanism and analyze its impact on the performance of Amazon EC2 instances.

6 Acknowledgments

The research conducted in this paper has been funded by the Finnish funding agency for technology and innovation (Tekes) in Massive Scale Machine-to-Machine Service (MAMMotH) project (Dnro 820/31/2011).

References

[1] BARKER, S., AND SHENOY, P. Empirical evaluation of latency-sensitive application performance in the cloud. *Proceedings of MMSys* (2010), 35–46.

[2] COOPER, B., SILBERSTEIN, A., TAM, E., RAMAKRISHNAN, R., AND SEARS, R. Benchmarking cloud serving systems with YCSB. *Proceedings of SoCC* (2010), 143–154.

[3] Dbench. `https://www.samba.org/ftp/tridge/dbench/`.

[4] GARFINKEL, S. L. An evaluation of Amazon's grid computing services: EC2, S3 and SQS. Tech. Rep. tR-08-07, Harvard University, 2007.

[5] Httperf. `http://www.hpl.hp.com/research/linux/httperf/`.

[6] LEE, G., CHUN, B., AND KATZ, R. H. Heterogeneity-aware resource allocation and scheduling in the cloud. *Proceedings of HotCloud* (2011), 1–5.

[7] LI, A., YANG, X., KANDULA, S., AND ZHANG, M. CloudCmp: comparing public cloud providers. *Proceedings of IMC* (2010), 1–14.

[8] PALANKAR, M., IAMNITCHI, A., RIPEANU, M., AND GARFINKEL, S. Amazon S3 for science grids: a viable solution? *Proceedings of the 2008 international workshop on Data-aware distributed computing* (2008), 55–64.

[9] Redis. `http://redis.io/`.

[10] SUNEJA, S., BARON, E., AND E. DE LARA, R. J. Accelerating the cloud with heterogeneous computing. *Proceedings of HotCloud* (2011), 1–5.

[11] Unixbench. `http://freecode.com/projects/unixbench`.

[12] WALKER, E. Benchmarking amazon EC2 for high-performance scientific computing. *USENIX ;login: 33*, 5 (2008), 18–23.

[13] WANG, G., AND NG, T. The impact of virtualization on network performance of amazon ec2 data center. *Proceedings of INFOCOM* (2010), 1–9.

[14] YEO, S., AND LEE, H. Using mathematical modeling in provisioning a heterogeneous cloud computing environment. *Computer 44*, 8 (2011), 55–62.