Should infrastructure clouds be priced entirely on performance? An EC2 case study

John O'Loughlin* and Lee Gillam

Department of Computer Science,
University of Surrey,
Guildford, Surrey, GU2 7XH, UK
Email: john.oloughlin@surrey.ac.uk
Email: l.gillam@surrey.ac.uk
*Corresponding author

Abstract: The increasing number of public clouds, the large and varied range of VMs they offer, and the provider specific terminology used for describing performance characteristics, makes price/performance comparisons difficult. Large performance variation of identically priced instances can lead to clouds being described as 'unreliable' and 'unpredictable'. In this paper, we suggest that instances might be considered mispriced with respect to their deliverable performance – even when provider supplied performance ratings are taken into account. We demonstrate how CPU model determines instance performance, show associations between instance classes and sets of CPU models, and determine class-to-model performance characteristics. We show that pricing based on CPU models may significantly reduce, but not eliminate, price/performance variation. We further show that CPU model distribution differs across different AZs and so it may be possible to obtain better price/performance in some AZs by determining proportions of models found per AZ. However, the resources obtained in an AZ are account dependent, displays random variation and is subject to abrupt change.

Keywords: cloud computing; virtual machines; performance; pricing; probability; brokers.

Reference to this paper should be made as follows: O'Loughlin, J. and Gillam, L. (2014) 'Should infrastructure clouds be priced entirely on performance? An EC2 case study', *Int. J. Big Data Intelligence*, Vol. 1, No. 4, pp.215–229.

Biographical notes: John O'Loughlin is the Service Delivery Team Leader within the Faculty of Engineering and Physical Sciences at the University of Surrey. He is also a part-time doctoral student within the Department of Computing with research interests in cloud computing, cloud economics and virtualisation. He installed and maintained private infrastructure as a service (IaaS) clouds based on OpenStack, for use within teaching and research in the department. He has guest lectured on the topics of virtualisation and the OpenStack system for the MSc course in Cloud Computing within the department.

Lee Gillam is currently a Senior Lecturer in the Department of Computing. His research interests include computational terminology, information extraction and grid computing. He has been responsible for software architectures for a number of systems developed for research projects supported by the EU's IT research and development programmes – TRANSTERM, POINTER, INTERVAL, ACE, SALT, GIDA, and PI on the eContent project LIRICS – and the UK EPSRC and ESRC – SAFE-DIS, SOCIS and FINGRID.

This paper is a revised and expanded version of a paper entitled 'Performance prediction for public infrastructure clouds: an EC2 case study' presented at the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2013), Bristol, UK, 2–5 December 2013.

1 Introduction

Cloud computing is a continuation of the theme of migrating from on-premise IT to the delivery of IT resources by service providers. Infrastructure as a service (IaaS) clouds offer computers, as might previously have been part of a core internally-offered infrastructure, in the form of virtual machines (VMs), attachable storage, and configurable networks. The flexibility of obtaining resources on-demand with a 'pay as you use' pricing model is driving some uptake of cloud computing, albeit with various reluctance in certain quarters. This paper addresses one area of reluctance – performance. In particular, we focus on performance in IaaS clouds, and when we use the term cloud in the remainder of this paper we are referring to IaaS unless otherwise stated.

For many, Amazon's elastic compute cloud (EC2) is the de facto standard for IaaS clouds; and as such is the focus of this study. EC2 uses the Xen hypervisor to abstract physical hardware into the VMs they offer. However, the physical infrastructure of EC2 is heterogeneous and so VMs of the same specification could be running on different hardware. Indeed, Amazon state – 'EC2 is built on commodity hardware, over time there may be several different types of physical hardware underlying EC2 instances' ('Amazon EC2 FAQs', http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it). Amazon's VM type descriptions do not identify specific hardware (except in a small number of specialised cases which we discuss later), such as CPU models, describing instead the quantities – for example, '2 vCPUs'. Knowing the level of achievable performance that such cloud resources will provide, a priori, is difficult and cannot be composed into a request. And yet this knowledge, essentially of resource performance, is essential for making price/performance comparisons both across instance types in the same cloud and between types in different clouds.

Cloud providers have taken to crafting abbreviations to rate performance, though arguably only in meaningful ways to themselves. The ratings used by EC2, and others are an attempt to provide, or at least to suggest, that a homogeneous level of performance can be obtained from a heterogeneous environment ('Amazon EC2 FAQs', http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it). Such an undertaking would appear to be difficult when one considers how the process of abstracting a physical machine into multiple VMs works. From Popek and Goldberg (1974), we know that one of the properties a hypervisor must have is that "a statistically significant fraction of machine instructions execute without VMM intervention". This is known as the performance property, and from it we can readily infer that VM performance is a function of the underlying hardware, and in particular that different instances of the same type will perform differently when running on different hardware. The pertinent question is of course, how different?

A number of authors have identified performance variation on EC2. This has mainly been as a side effect of comparing EC2 to local systems or for testing HPC suitability rather than investigating performance on EC2 per se. These papers tend towards preferring internal systems, and to demonstrate support for this selection they run benchmarks on a relatively small number of instances across a limited number of availability zones in EC2. As such, results rarely account for the full extent of performance variation of an instance type, or ex7plain the causes of it, and also miss the fact that the variation is different in different AZs in EC2 due to heterogeneity.

The aim of this paper is to understand how instance compute performance varies by the physical resources available to EC2 customers, estimate how resources are distributed across EC2, understand how they are allocated to customers and consider whether pricing should be related to performance. As such, we:

1 quantify the range of performance variation across all non-specialised instance types

2 identify CPU model as cause of performance variation

3 identify the sets of CPU models associated with instance classes

4 quantify the performance of each CPU model with respect to our benchmark

5 show that instances types of the same class may be allocated different resources when running in the same AZ

6 we demonstrate how EC2 AZ mappings affect the range of performance a customer can obtain from a region.

The remainder of this paper is structured as follows: Sections 2 and 3 provide an overview of EC2 followed by a discussion of our methodology for measuring instance performance. In Sections 4 to 7, we present benchmarking results of 1,297 instances across six regions and 14 availability zones. Section 8 examines how the value of an ECU varies across instance classes. In Sections 9 and 10, we look at the performance properties of CPUs (with respect to our benchmark) and how their distribution across EC2 varies. In Sections 11 to 13, we examine how resources are allocated to customers and the impact that this has on price/performance a user may obtain. We review and compare related work in Section 14, with conclusions and future work presented in Section 15.

2 EC2 background information

Amazon's EC2 global infrastructure ['Global infrastructure', (2 July 2013), http://aws.amazon.com/about-aws/globalinfrastructure/] is divided into eight regions (excluding the GovCloud region), which we can think of as 'sub' clouds. When using the EC2 API to manage instances a region must be specified – with US East being the default if no region is set. Similarly, when using the web interface, a user must first connect to one of the regions before they can launch new instances or manage existing ones. A user's credentials are the same across all regions, however, resources such as SSH keys and security groups (firewall settings for instances) are solely contained within a region. A user is required to setup these resources for each region they wish to use.

Regions consist of AZs, which are isolated from each other and have their own networking and power infrastructure – and so should provide an element of redundancy: failure of one AZ should not affect instances running in another. The largest and first publically available region is US-East N. Virginia, which has 5 AZs; whilst the newest region is Asia-Pacific Sydney with 2 AZs. In total, there are 23 AZs across EC2, however not all of these are

available to customers. Both US East N. Virginia and US West N. California restrict the number of AZs they make available to new users. One of the authors has 2 AWS accounts, one of which can launch instances in 4 US East N. Virginia AZs whilst the other, in line with all new EC2 accounts, can only use 3.

In Table 1, we list all the regions, their locations, API endpoint names and the number of AZs in each one.

Table 1 EC2 regions

Geographical area	*Location*	*API end-point*	*AZs*
US	N. Virginia	us-east-1	5
	N. California	us-west-1	3
	Oregon	us-west-2	3
Asia Pacific	Singapore	ap-southeast-1	2
	Tokyo	ap-northeast-1	3
	Sydney	ap-southeast-2	2
Europe	Dublin	eu-west-1	3
South America	San Paulo	sa-east-1	2

EC2 offers VMs in various sizes known as instance types. An instance type description specifies the number of vCPUs, the amount of RAM and local storage. By an instance, we simply understand a running VM of a given type. Instance types are grouped together into instance classes; 4 of which we refer to as standard (or non-specialised) instance classes. Instance types in the same class have a similar ratio of RAM to number of vCPU cores. In total, there are 11 standard instance types and these are available in all AZs and to all users. There are six instances types which we consider to be specialised as they have some or all of the following hardware: SSD, 10 Gb networking and GPUs. We do not consider specialised instance types in this paper.

In addition to quantities of resource, instance type descriptions include performance descriptors which are intended to give the user an indication of expected performance. However, these tend to be somewhat vague, for example the m1.small is described as having 'moderate' I/O. For the compute capacity of an instance Amazon use a rating called the EC2 compute unit (ECU). The ECU is defined as follows: "Equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor". The ECU rating is per core, so the total rating for an instance type is given by: number of cores multiplied by ECU core rating. For example the m1.xlarge instance type has 4 vCPUs at 2 ECU per core so is rated at 8 ECUs. When we use the term 'rated at' we are referring to the per core rating. We detail standard instance types in Table 2.

Describing expected performance in terms of reference machines appears to be a growing trend, for example, Google describe their Google Compute Engine Unit (GCEU) ('Google Cloud Platform', https://cloud.google.com/pricing/compute-engine) as follows: 'We chose 2.75 GCEUs to represent the minimum power of one logical core (a hardware hyper-thread) on our Sandy Bridge platform'. Similarly, HP define their HP Cloud Compute Unit (HPCCU) ('HP Cloud Pricing', https://www.hpcloud.com/pricing) as: '6.5 CCUs are roughly equivalent to the minimum power of one logical core (a hardware hyper-thread) of an Intel(R) 2012 Xeon(R) 2.60 GHz CPU'.

Table 2 Non-specialised EC2 instance types

Class	*Instance type*	*RAM (GB)*	*vCPU*	*ECU per core*
First generation standard	m1.small	1.7	1	1
	m1.medium	3.75	1	2
	m1.large	7.5	2	2
	m1.xlarge	15	4	2
High CPU	c1.medium	1.7	2	2.5
	c1.xlarge	7	8	2.5
High memory	m2.xlarge	17.1	2	3.25
	m2.2xlarge	34.2	4	3.25
	m2.4xlarge	68.4	8	3.25
Second generation	m3.xlarge	15	4	3.25
	m3.2xlarge	30	8	3.25

Amazon, in common with other providers, does not publish detailed information about how their ECU is measured, and how much variation there is in the measurement. Instead, they simply state: "EC2 uses a variety of measures to provide each instance with a consistent and predictable amount of CPU". Publishing variation information would allow customers to gauge if the performance their instances are delivering is in line with what they may expect.

Instances are made available in three pricing models: *reserved instances*, *on-demand instances* and *spot instances*. With reserved instances users pay an upfront fee and obtain a reduced hourly charge for the instance. On-demand instances are perhaps the most familiar as they are charged solely on a per hour basis with no upfront costs. Spot instances allow users to bid for unused capacity at prices lower than on-demand instances. In order to manage costs, we use spot instances in our experiments. Spot prices do vary, and we have observed occasions when spot prices were similar to on-demand prices and indeed some occasions when spot prices significantly exceeded on-demand prices.

Amazon state that ('Amazon EC2 FAQs', http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it) spot instances 'perform exactly the same as on-demand instances'. However, we cannot really know if the resources obtained here reflect what we would have obtained in the on-demand market without running an appropriate test alongside.

EC2 offers a generic service level agreement (SLA) for all customers. The only performance metric for is for service availability – not for compute performance. SLAs not offering a guarantee of performance perhaps would be less problematic if instance performance was consistent – however, we show in Sections 4 to 7 this is not

the case. In Section 10, we consider how CPU performance data may be used to add performance related metrics to SLAs.

3 Measuring compute capacity

From Table 2, we see that EC2 offers instances with cores rated at 1, 2, 2.5 and 3.25 ECUs, respectively. However, how do these ratings relate to actual application performance? For example, should a user expect their application to run in half the time on a 2 ECU instance as compared to a 1 ECU instance? Whilst Amazon (and others) is defining their compute rating in terms of references machines we would argue that application performance (typically execution time) is the most intuitive and easy to understand measure of performance for users. For our experiments, the metric we chose for measuring compute performance is *application execution time*.

Benchmarks are the standard way to measure and compare the compute performance of computer systems. There is a wide range of benchmarks available, some of which have been specifically written, whilst others are 'real world' benchmarks applications. Specifically written benchmarks typically attempt to mimic a particular class of applications. Synthetic benchmarks, such as Whetstone, attempt to mimic statistical CPU usage, whilst kernel benchmarks, such as NAS and Linpack, mimic the main computations performed. Due to limitations of purpose written benchmarks the trend is now towards benchmarks that are similar to (or actually are) applications that users will use in their 'real work'. Such benchmarks are usually CPU bound tasks; this approach is typified by the SPEC CPU2006 ('SPEC CPU2006', http://www.spec.org/cpu2006/), a commercially available suite of integer and floating point benchmarks. The suite contains, for example, CPU integer benchmarks that perform the following tasks: compressions, audio encoding, video transcoding and compilation. The floating point suite contains benchmarks that perform speech recognition, weather modelling and fluid dynamics.

From the definition of the ECU, it is not clear if it is equally representative of integer and floating point performance, or if it is skewed in favour of one or the other. Although we do not consider such matters further here, we note that in some domains, such as scientific computing, this is likely to be an important consideration.

In order to measure compute performance of EC2 instance types we chose to use a benchmark based on bzip2, a compression tool found on Linux/UNIX systems. Bzip2 is a CPU bound task, although clearly there is some input and output (I/O) involved as the input file needs to be read and the compressed file is written back. During a bzip2 compression, we ran *top* in batch mode to capture the CPU state at 5 s intervals. From this, we observed that the CPU spends 97% to 98% of its time in the user state, i.e., running bzip2, whilst being in an I/O wait state (when bzip2 blocks) for only 2% to 3%. From this, we can infer that whilst there is some dependency on disk speed the time taken to complete the compression is predominately determined by CPU speed, i.e., it is a CPU bound task. Bzip2 uses the Burrows Wheeler Transform ('bzip2', http://www.bzip.org/1.0.5/bzip2-manual-1.0.5.html#intro) to compress files in blocks with the default block size being 900 KB. By compressing in blocks, even files larger than available RAM do not cause swapping when being compressed. This is confirmed by running *vmstat* at 5 s intervals to capture the number of bytes being swapped.

The bzip2 benchmark is part of the OpenBenchmarking.org toolkit and the Standard Performance Evaluation Corporation (SPEC) CPU benchmark toolkit, although they use a slightly modified version which sends its output to the device /dev/null to minimise I/O. For the input file to bzip2, we used an Ubuntu 10.04 AMD desktop ISO file (696MB), and timed the compression. In each instance, the benchmark was run 3 times and we take the average. Before running bzip2 the input file, ubuntu.iso was copied to a file called test.iso and bzip2 was run on test.iso. As all I/O on a Linux system goes through the disk cache in RAM (and remains there until reclaimed) this helps to reduce disk reads. We refer to the above procedure as 'the standard benchmark'.

We can determine the CPU model an instance is running on by examining the file/proc/cpuinfo. This file is populated via the *cupid* instruction which is non-trapping on the Xen hypervisor. On clouds that use the KVM hypervisor, such as GCE, this instruction does trap and the hypervisor can be configured to export a variety of models to the guest ('Guest CPU Models', https://access.redhat.com/site/documentation/en/Red_Hat_Enterprise_Linux/6/html/Virtualization_Getting_Started_Guide/para-CPU_Models.html). As such /proc/cpuinfo cannot be used to determine physical CPU model.

Our performance investigations are as follows: We begin by benchmarking first generation standard instance types. We divide this into:

a m1.small instance type

b m1.medium, m1.large and m1.xlarge.

The reason for this division is that the m1.small is rated at 1 ECU whilst the others are rated at 2 ECU. We next benchmark high CPU instance types, c1.medium and c1.xlarge, rated at 2.5 ECU. This is followed by high memory instance types m2.xlarge, m2.2xlarge and m2.4xlarge, and second generation standard instances types, m3.xlarge and m3.2xlarge, all rated at 3.25 ECU. We have therefore covered all of the non-specialised instance types. The m1.small instance was benchmarked across 5 out of 8 EC2 regions and in all AZs within these regions. This wide ranging initial experiment allows us to measure how 1 ECU varies across a large portion of EC2.

Ideally we would have liked to run the same number of instances, in the same AZs, for other instance types as for the m1.small. In order to stay within budget, for each of the remaining instance type classes we benchmarked 200 instances across 2 regions (always including US East N. Virginia) and in all AZs that are open to us with a specific

user account within these regions – Section 10 discusses AZs in further detail. The number of instances run of a particular instance type within a class was primarily determined by cost constraints.

4 First generation standard instances

4.1 Performance of m1.small (1 ECU)

We benchmarked 540 m1.small instances across five regions (out of a possible 8) which included 2 US-based ones (US East N. Virginia and US East N. California), and 1 region each from Europe (EU Dublin), Asia-Pacific (Asia-Pacific Sydney) and South America (South America San Paulo). From these five regions, 13 AZs were available for use (with the account we were using) and we ran instances in all of them.

All 540 instances were backed by one of four different CPU models: Intel Xeon E5430, Intel Xeon E5-2650, Intel Xeon E5645 and Intel Xeon E5507. (We use only CPU model numbers from here on). Figure 1 shows a histogram of our results, with interval width of 5 s, together with some summary statistics in Table 3.

Figure 1 Histogram of benchmark results for m1.small instances

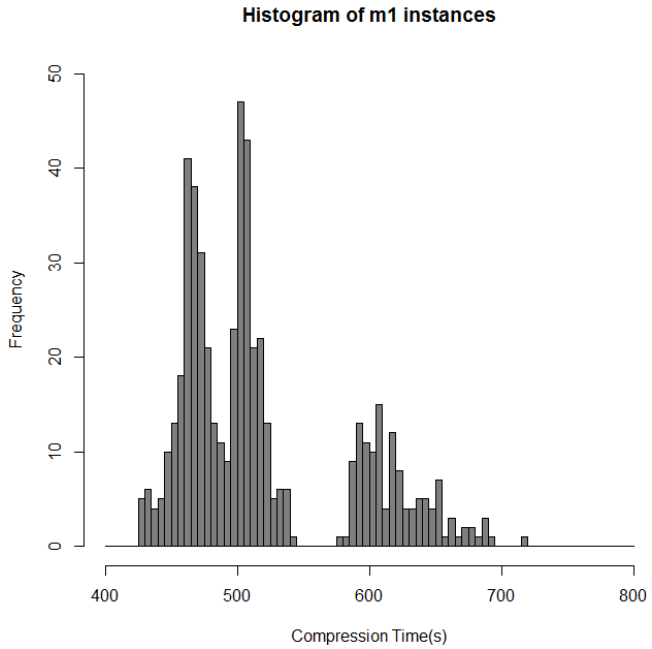


Table 3 Summary statistics for m1.small

Min(s)	*Max (s)*	*Median (s)*	*Mean (s)*	*Sd (s)*
425.35	715.72	501.88	517.76	63.24

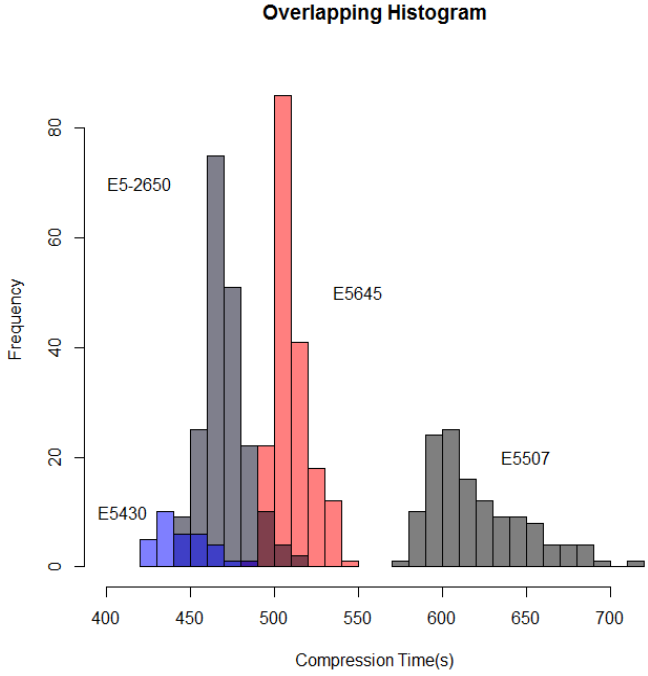
The distribution appears to be multi-modal, a mix of at least three uni-modal distributions. In Table 4, we present the results of the benchmarks by CPU model, followed by an overlapping histogram of the CPU model distributions in Figure 2.

Figure 2 shows that performance is largely determined by CPU model. There is little overlap between models, except between the E5430 and the E5-2650. More importantly, the worst performing instances (at the same price) were all backed by the E5507; the difference between the best performing E5507 and the worst of the other models (an E5645) is just under 25s. The E5-2650 and the E5645 are the models found most frequently, backing 397 of our 540 instances. The range of these models combined is [470.24, 543.8] but they only overlap in the range [495, 510] and this overlap covers fewer than 25 instances.

Table 4 Summary statistics by CPU model

Model	*Min (s)*	*Max (s)*	*Mean (s)*	*Sd (s)*
E5430	425.35	482.1	445.1	14.33
E5-2650	443.48	518.92	470.24	13.03
E5645	487.95	543.8	510.07	10.51
E5507	578.03	715.72	620.87	28.46

Figure 2 Histogram of benchmark results by CPU model (see online version for colours)



4.2 Performance of m1.medium, m1.large and m1.xlarge (2 ECU)

The remaining first generation standard instance types are m1.medium, m1.large and m1.xlarge, all of which have vCPU rated at 2 ECU. For cost management reasons, we limited our experiment to 200 instances: 80 m1.medium, 80 m1.large and 40 m1.xlarge, across 2 Regions, US East N. Virginia and US West Oregon. [Note: US West Oregon is included here due to having the lowest spot prices at the time]. These regions have 7 AZs but at the time of running us-west-2c were unavailable to our account. Unfortunately, due to a configuration error in the experiment, we were also unable to obtain the results from zone us-west-2a. We successfully collected benchmark

results from 157 instances: 78 m1.medium, 60 m1.large and 19 m1.xlarge.

All of our instances were backed by the same set of CPU models as the m1.small instance types, so we believe we have identified the set of the CPU models for all standard first generation instance types in six regions and 14 AZs across those. Clearly, the set of hardware platforms underlying instances classes will change over time, as we discuss in Section 13. However, the association is stable in the short-term.

As with the m1.small instance type, the distribution of results is multi-modal and is determined by CPU model, as shown by Figure 3 (histogram of results by CPU model). This means that, for instance, 2 m1.medium instances with different CPUs will show performance differences in line with Figure 2; whilst an m1.medium and an m1.xlarge backed by the same CPU will perform similarly and the performance can be predicted by the results presented in Table 5.

Figure 3 Histogram of benchmark results by CPU model (see online version for colours)

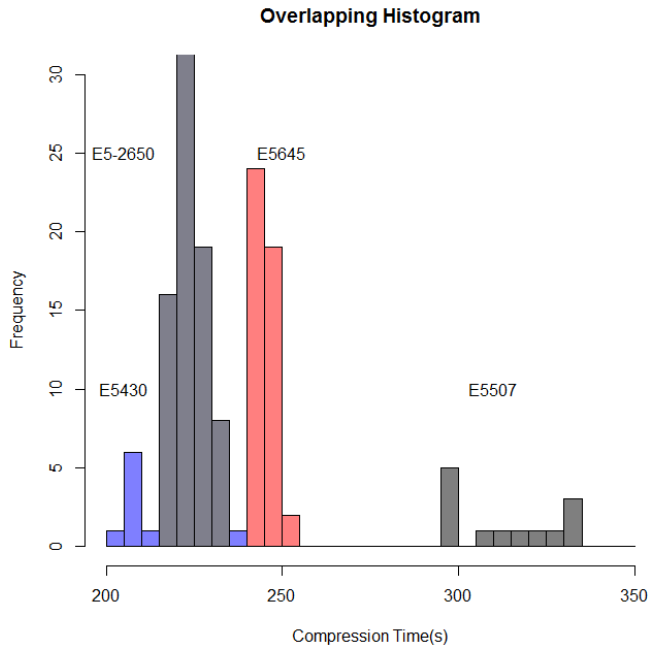


Table 5 Summary statistics for remaining first generation standard

Min (s)	*Max (s)*	*Median (s)*	*Mean (s)*	*Sd (s)*
204.71	332.02	226.79	236.23	25.98

We found no evidence to suggest that larger more expensive instance types are more likely to obtain better performing CPUs. For example, in the zone us-east-2 40 m1.large instances were backed by 33 E5-2650 and 7 E5645 whilst 19 m1.xlarge instances were backed by 10 E5-2650 and 9 E5645. We can say that different instance types in the same class provide more resources, but at the same level. This is in line with the VM descriptions and linear scaling of on-demand pricing, where, for example, an m1.large is twice the price of a m1.medium and has twice the RAM and number of vCPU cores. We discuss differences found proportions of CPU models backing different instances types within the same class further in Section 12.

Table 5 presents the summary statistics of all our instances and Table 6 show the breakdown by CPU model.

Table 6 Summary statistics by CPU model

Model	*Min (s)*	*Max (s)*	*Mean (s)*	*Sd (s)*
E5430	204.71	238.2	211.30	10.43
E5-2650	217.81	233.51	223.40	3.84
E5645	240.9	254.11	244.71	2.90
E5507	295.91	332.01	312.92	14.91

By comparing our 1 ECU to 2 ECU results, we see that our worst 2 ECU (332.02s) is only 93.33s better than our best 1 ECU (425.35). In this case, we might suggest that our 2 ECU core is only providing 1.28 ECUs. Similarly, we compare the worst performing 1 ECU (715.72 s) with our best performing 2 ECU (204.71s) and in this case the 2 ECU core is providing 3.5 ECUs. However, when considering a particular CPU model, we do see that performance does scale linearly, and 2 ECU is approximately twice as fast as 1 ECU. We also note the small coefficient of variation for a CPU model; and from this we could predict instance performance accurately if we knew the CPU model we were going to obtain before launching the instance. However, does an ECU scale with respect to the remaining non-specialised instance types with cores rated at 2.5 ECU and 3.25 ECU?

5 High CPU instance types

We next investigate the high CPU instance types, the c1.medium and c1.xlarge. These have 2 and 8 cores respectively rated at 2.5 ECU. The High CPU instance type was first made available in June 2008.

We benchmarked 200 high CPU instance types across two regions, US-East N. Virginia and Asia-Pacific Sydney, which offered 6 AZs, and we ran instances in all of them. We used Asia-Pacific Sydney instead of US West Oregon for our second region as we found spot prices to be lower there.

We found that both these instance types are backed by the following CPU models: E5-2650 (19), E5345 (4), E5410 (47) and E5506 (130). Note that the E5-2650 also backs first generation standard instances and is the only model we found that backed more than one class. As with the first generation standard instance types, the performance of a high CPU instance is determined by the CPU model. In Figure 4, we present a histogram of the results, followed by summary statistics for all instances which we then break down by CPU model.

Figure 4 Histogram of benchmark results by CPU model (see online version for colours)

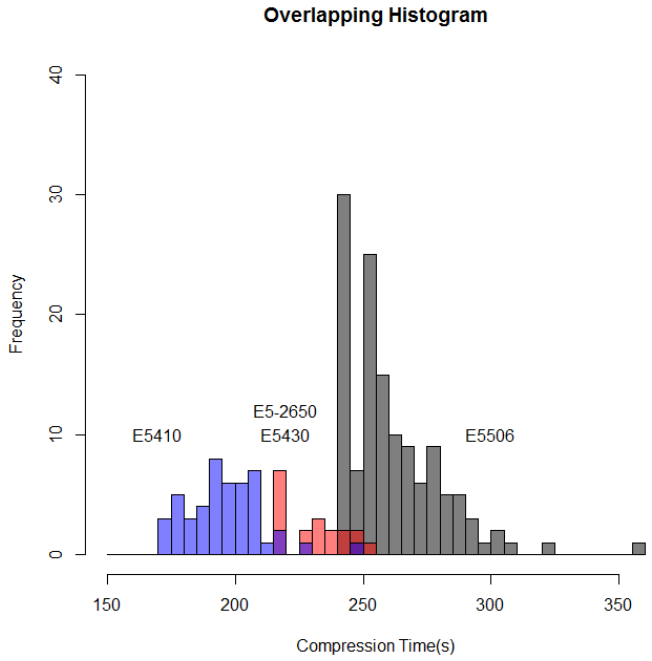


Table 7 Summary statistics for high CPU instance type

Min (s)	*Max (s)*	*Median (s)*	*Mean (s)*	*Sd (s)*
174.55	356.92	246.55	242.58	32.58

Table 8 Summary statistics by CPU model

Model	*Min (s)*	*Max (s)*	*Mean (s)*	*Sd (s)*
E5410	174.56	245.86	196.21	15.03
E5345	215.58	235.67	224.34	8.34
E5-2650	216.93	250.03	230.83	12.19
E5506	241.46	356.92	261.63	18.84

65% of our high CPU instances were backed by the E5506, our worst performing CPU, whilst our best performing CPU, the E5410, backed just 23.5% of our instances. On average, there is a 33% increase in the time taken to run our standard benchmark on an E5506 compared to an E5410. Later, we consider how 2.5 ECUs compares to 1 ECU.

6 High memory instances

The next group of non-specialised instance types is high memory: m2.xlarge, m2.2xlarge and m2.4xlarge. These instance types have 2, 4 and 8 vCPU, respectively, with 8.5 GB of RAM per vCPU. The vCPU cores are rated at 3.25 ECU. We benchmarked 200 high memory instance types across two regions US-East N Virginia and Asia-Pacific Sydney, which offered 5 AZs and we ran instances in all of them. Note that zone us-east-1b was unavailable whilst these benchmarks were carried out and so we could only use 5 AZs.

As with the previous instance types, the performance of high memory instances is determined by the CPU model. Instances were backed by just two models: E5-2665 (134) and the X5550 (66). In Figure 5, we present a histogram of the results, followed by summary statistics for all instances which we then break down by CPU model.

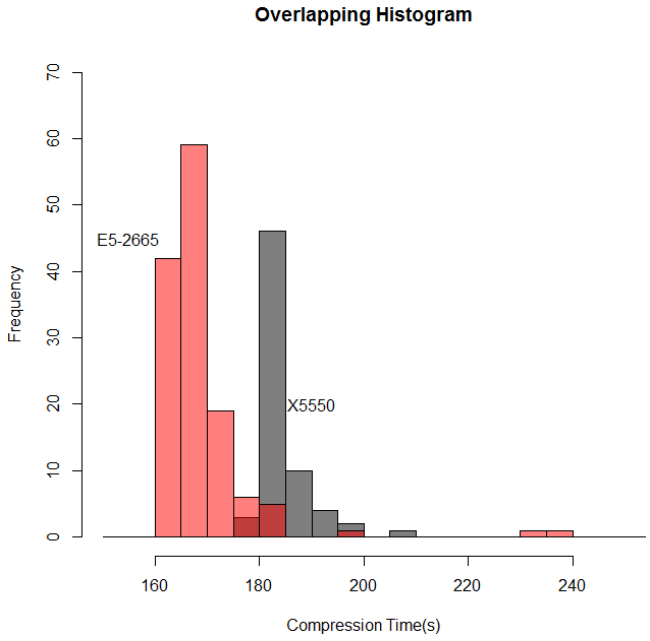
Table 9 Summary statistics for high memory instance type

Min (s)	*Max (s)*	*Median (s)*	*Mean (s)*	*Sd (s)*
163.06	235.87	169.79	174.19	11.10

Table 10 Summary statistics by CPU model

Model	*Min (s)*	*Max (s)*	*Mean (s)*	*Sd (s)*
E5-2665	163.06	235.87	169.07	9.69
X5550	179.75	207.86	184.59	4.67

Figure 5 Histogram of benchmark results by CPU model (see online version for colours)



7 Second generation standard instances

The second generation standard instances are m3.xlarge and m3.2xlarge with 4 and 8 cores, respectively. As with the first generation, they have 3.75 GB of RAM per vCPU, but with cores rated at 3.25 ECU. This instance class was first introduced in 2011 to 2012 in US-East N Virginia, and made available globally as of February 2013.

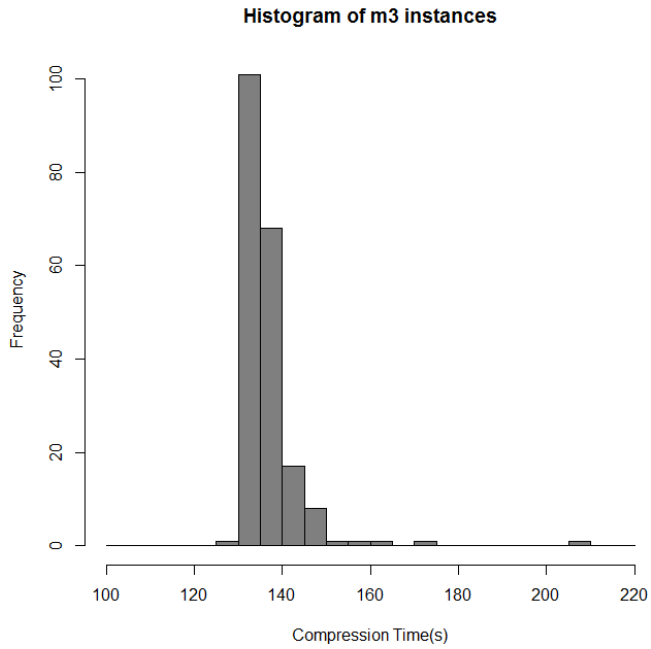
We benchmarked 200 second generation standard instance types across two regions US-East N Virginia and Asia-Pacific Sydney, which offered 5 AZs and we ran instances in all of them. Note that zone us-east-1b was unavailable whilst these benchmarks were carried out and so we could only use 5 AZs.

All the m3 instances we benchmarked were backed by just one processor model: E5-2670.

Table 11 Summary statistics for second generation standard instance type

Min (s)	*Max (s)*	*Median (s)*	*Mean (s)*	*Sd (s)*
129.99	206.27	134.89	135.85	7.65

Figure 6 Histogram of benchmark results



8 Comparisons of an ECU across instance classes

Despite the EC2 definition of an ECU as architecture independent, the previous sections have shown that the value of an ECU within an instance type class is determined by the physical CPU. We now consider how an ECU varies across instance type classes. In Table 12, we record the per vCPU ECU rating for each instance class, together with the mean and median benchmark values as previously determined. We can then use this to determine if, for example, an instance rated at 2.5 ECU will perform '2.5 times better' than instances rated at 1 ECU, i.e., the execution time will be 2.5 times faster.

Table 12 ECU across instance type classes

Instance class	*ECU*	*Median (s)*	*Mean (s)*
First Generation m1.small	1	501.88	514.01
First Generation others	2	226.79	236.23
High CPU	2.5	246.55	242.58
High Memory	3.25	169.79	174.19
Second Generation Standard Instances	3.25	134.89	135.85

For instance, types rated at 2 ECU this appears to scale linearly with respect to the statistics we calculate for the instance type rated at 1 ECU. This is perhaps to be expected as they all belong to the same class, first generation standard, and are all backed by the same set of CPU models. The high CPU instance types, rated at 2.5 ECU, performed worse on average then instances rated at 2 ECU.

Both the m3 range and the m2 range are rated at 3.25 ECU, and yet we can see that there is a 25% increase in the average time taken to run the standard benchmark on high memory instance types (m2 range) as compared to second generation standard instance type (m3 range). If we look at two similar sized instance types from these classes: m3.xlarge and m2.2xlarge, they both have 4 vCPUs rated at 3.25 ECU and 15 GB and 34.2 GB of RAM, respectively. The on-demand per hour cost in US East N. Virginia is $0.55 and $0.92. So, although the m2.2xlarge has an extra 19.2 GB of RAM this comes at a per hour cost of $0.37 and takes 25% longer to run the standard benchmark.

9 Performance properties for CPUs

On physical hardware, with no other tasks running, a CPU bound task produces repeatable results – results with negligible coefficient of variation and no outliers. This property is what makes them good candidates for benchmarks; and indeed deviation from the mean is an indication of a fault with a system. Therefore, the range of results for instances backed by the same CPU model is perhaps surprising. Except that infrastructure clouds are multi-tenanted, meaning that one physical server (or host) can host multiple VMs, and may indeed be doing so at the time of these tests.

The exact number of VMs running on a host depends on the number of cores the host has, the size of the VMs, and the density at which the provider wishes to run their cloud. By density, we mean the number of vCPUs to one physical core, and over-subscription is the term used when there are more vCPUs than physical cores.

A vCPU is a scheduling entity, and so if a provider runs, for example, 2 m1.xlarge and 2.medium instances on an 8 core server, there would be 10 vCPUs for 8 physical cores. In this case, each vCPU could expect to spend approximately 80% of its time in a run state. We do not know the density at which Amazon run EC2, and indeed it is possible, if not probable, that faster CPUs may have a higher density than slower ones. That is, the E5430 may run more m1.small instances than an E5507.

What is clear is that performance in a shared system differs from performance with exclusive access to resources. The potential for one VM to affect the performance of another is well known, and is often referred to as the 'noisy neighbour effect'. For example, the Xen hypervisor (used on EC2) uses a credit scheduler to manage guest CPU usage. However, CPU cycles doing I/O instructions on behalf of a guest, which trap to the hypervisor, are not accounted for. As such VMs with an I/O intensive workload may obtain a disproportionate amount of CPU time.

In a cloud we should therefore expect instances, even backed by the same CPU, to have a distribution of

performance results. What, if anything, can we infer from the shape of any such distribution?

Considering just the CPU models backing FGS instances, we can see from our histograms that they are positively skewed, and calculations confirm this. A longer tail to the right perhaps indicates we have seen more examples of degraded performance than of performance bursts, and so the majority of results 'bunch' to the left. For example, on the E5645 for FGS instances, the minimum value was 487.95 s and the 75th percentile is 515.87 s; the first 75% of results are within 27.92 s. The maximum value is 543.8 s and so the upper quartile range is 27.93 s.

The lack of performance bursting is likely explained by use of Xen CPU capping ('Xen', http://wiki.xen.org/wiki/Credit_Scheduler), and so even if cycles are available an instance cannot make use of them. Undoubtedly, this is a necessity to ensure a 'consistent' level of compute. The positive skew, or performance degradation, is quite possibly due to 'noisy neighbour' effects, as discussed above.

If we know the CPU model backing an instance, then the empirical data we have collected can be used to estimate the probability that the instance will run our benchmark within a given time. This is particularly useful for customers who require quality of service (QoS) performance metrics in SLAs. As an example of how performance metrics may be expressed, consider provisioned EBS volumes on EC2, which are described as: 'designed to deliver within 10% of the provisioned IOPS performance 99.9% of the time' ('Elastic Block Storage', http://aws.amazon.com/ebs/). Generally, we may express QoS for performance metrics in terms of an expected level, perhaps in a given range, to be met a given percentage of times.

As an example of how we may use CPU benchmarking as a performance metric, consider the second generation standard class. We know, since EC2 include it in the class description, that instances of these are, currently, always backed by the E5-2670 model. From our experimental data, we calculate the 90%, 95%, 99% percentiles as: 142.89 s, 147.28 s and 160.27 s. And so we could construct an SLA with the following terms: the instance will run the bzip2 benchmark in under 160.27 s 99% of the time. And so, from this, we begin to envisage how to formulate SLAs relating to workloads over extended durations.

However, until we – as a user of cloud systems – launch an instance, we will not know the hardware that backs it. We might use our results to say that, for example, an m1.medium instance will run our standard benchmark in 242.22 s on average, but if we are 'unlucky' and obtain an instance backed by an E5507 the average becomes 311.41 s.

10 CPU model distribution across EC2

As Amazon add new regions, add and expand zones in existing regions, and make hardware refreshes, the distribution of CPU models will increasingly differ from zone to zone. In Table 13, we list the proportion of CPU models backing the FGS instances in the AZs we tested. For example, in us-west-1b we found that 87% of our instances were backed by E5507. Without too much consideration, and with knowledge of CPU performance as outlined in Sections 4 to 7, a user would most likely avoid using us-west-1b and uswest-1c, if their region mapping were consistent with this one, due to the high proportion of E5507 models found there.

Table 13 CPU by AZ

Zone	*E5430*	*E5-2650*	*E5645*	*E5507*
us-east-1a	31%	0	25%	44%
us-east-1b	5%	59%	29%	7%
us-east-1c	0	47%	52%	1%
us-east-1d	18%	31%	44%	7%
us-west-1b	0	0	13%	87%
us-west-1c	8%	0	18%	74%
eu-west-1a	4%	75%	19%	2%
eu-west-1b	28%	0	44%	28%
eu-west-1c	4%	0	63%	33%
ap-southeast-2a	0	64%	36%	0
ap-southeast-2b	0	75%	25%	0
sa-east-1a	0	81%	19%	0
sa-east-1b	0	86%	14%	0
us-west-2b	0	73%	27%	0

The original experiments were focused on performance, benchmarking 697 FGS instances across 14 AZs. This allows us to be reasonably sure we have identified all CPU models associated with the FGS class, and to determine a performance distribution for each model with respect to our benchmark. However, the number of instances run per AZ is relatively low; just under 50 on average. As such there remains a number of questions on CPU distribution across AZs that need to be addressed, and which we address in the sections identified:

- How does AZ mapping affect the resources a user can obtain? (Section 9)
- Does the proportion of CPU models differ between instances types in the same class? (Section 10)
- Are proportions of CPU models found stable in the short-term? (Section 11)

To begin to answer these questions, new experiments were conducted in September 2013. These focused on the distribution of CPU models in 3 AZs for 2 Amazon accounts over a 3 week period. We refer to the Amazon accounts as Original, New and New2. The Original account is the one used to obtain the performance results above. In the sections below, we use the results from our new experiment to address the questions above.

11 AZ mappings

We have previously defined AZs as locations within a Region that are isolated from each other, with separate power and network connections; and interconnected with low latency networking. Using the word 'isolation' in the AZ definition strongly implies a degree of physical separation between them. From this, we can infer that Regions are, most likely, physically comprised of one or more distinct data centres (DC), if not, then AZs are physically housed under the same roof, and whilst they may have separate power and networking they would be all potentially be affected by fire or flooding in the DC. And so it is natural to ask if a region, such as US-EAST N. Virginia, which offers 5 AZs, us-east-1a, …, us-east-1e, is physically comprised of 5 DCs, say DC1, …, DC5, and whether the AZs names uniquely identify them. That is, for all customers, useast-1a identifies DC1, us-east-1b identifies DC2 and so on. If this were the case, then given knowledge of AZs with better performing resources, this could potentially lead to capacity problems with customers preferring better performing AZs over others. Perhaps because of this, AZs names do not identify the same location for all customers. This can be inferred from the EC2 statement: "…your availability zone us-east-1a might not be the same location as us-east-1a for another account".

To illustrate the differences in how the same AZ name maps to different locations for different customers, we ran 500 m1.medium instances in AZ us-west-1c using our original and new accounts, 250 instances per account, and recorded the CPU models each account obtained. In Table 13, we record the results. There are considerable differences in the resources obtained by the different accounts, and it is likely that the respective us-west-1c name maps to different locations. For the account new, 87% of its instances in us-west-1a are backed by the E5-2650 CPU; whilst the original account has 75% of its instances backed by the E5507. From Table 5, we know that (to the nearest second) the E5-2650 runs the benchmark in average time of 223s whilst the E5507 takes 313 s on average. Therefore, new is apparently able to obtain better price/performance from the AZ it sees as us-west-1c, than original.

If we assume that a location is a whole DC then for USEAST N. Virginia we would indeed have 5 DCs, and for each customer we would have a mapping from the names {us-east-1a, …, us-east-1e} to {DC1, …, DC5}. A new customer in this region is allocated 3 AZs out of a possible 5, and so this would give ten different possible combinations of locations a customer may have, that is, they may have use of DC1, DC2 and DC3 or it may be DC2, DC4 and DC5. This then leaves open the possibility for EC2 to remap a customer's AZ to a different DC, which may be useful in managing capacity. However, some resources, such as EBS volumes are AZ specific. And so any such remapping would have to ensure that all resources are replicated to the new DC.

As EC2 do not, currently, define what they mean by a location we can consider a number of possible architectures, in addition to the 'a location is a DC'. One possibility is that each DC is sub-divided into a number of logical data centres (LDC), and the AZ mapping maps a user account to a LDC. In this scheme, a DC expansion may either add additional hosts in LDC, or add a new LDC. Two customers, both with access to us-east-1a for example, may be mapped to either the same, or to a different LDC, but both within the DC comprising us-east-1a. This architecture allows EC2 to quickly remap a customer's AZ to a different LDC within the same DC, as it is not necessary to replicate the resources elsewhere. From the point of view of resources, recently added LDCs are likely to contain different resources than older ones.

We now examine the effect of restricting different users to different subsets of resources in a region. In the US-West N. California Region EC2 restricts a user to 2 AZs out of a possible 3 ('Global Infrastructure', http://aws.amazon.com/about-aws/globalinfrastructure/). For each of our three accounts, original, new and new2, we launched 80 m1.medium instances in all AZs they have access to, except in us-west-1c where we already have data from Table 13 (Note: data presented in Table 14 was obtained in the same time period).

Table 14 Names to AZ mappings in us-west-1c

Account	*AZ*	*E5430*	*E5-2650*	*E5645*	*E5507*
Original	us-west-1c	16%	0	9%	75%
New	us-west-1c	0	87%	13%	0

Table 15 Names to AZ mappings

Account	*AZ*	*E5430*	*E5-2650*	*E5645*	*E5507*
Original	us-west-1b	0	83%	4%	11%
Original	us-west-1c	16%	0%	9%	75%
New	us-west-1a	0	75%	25%	0
New	us-west-1c	0	87%	13%	0
New2	us-west-1a	0	93%	7%	0
New2	us-west-1b	0	81%	19%	0

From Table 14, us-west-1a for New account appears similar to us-west-1b for new2: (75%, 25%) and (81%, 19%) of E5-2650 and E5645, respectively. In a statistical test, using a 2 sample proportion test, with the null hypothesis that the proportions of E5-2650 are equal, produces a p-value of 0.44. Based on these samples we would not reject the null hypothesis. Also, us-west-1c for New is similar to us-east-1a for new2. However, the proportions found by original account are sufficiently different, from both new and new2, for us to reject a null hypothesis that the proportions are equal.

The AZs the new account can use are broadly similar in terms of the proportions of resources on offer. This should make both load balancing and high availability easier to manage. For example, the new account may prefer to use uswest-1c as its 'main' AZ, given the slightly higher proportions of E5-2650 available to it there, and only 'failing over' to uswest-1a when us-west-1c is unavailable. In a batch of ten instances in us-west-1c we would expect 9 E5-2560, whilst in us-west-1a we would expect 7 or 8. However, if we wanted to be 95% sure that we have 9 E5-2650 in us-west-1a, how many instances should we start? If we assume independence, then we have a Bernoulli trial, and we would need to start 15 instances to be 95% sure. Whether or not the probability of obtaining a particular model is independent of the previous models obtained depends on the scheduling algorithm being used. For example, an algorithm that attempts to co-locate users VMs on the same hosts would produce dependence. In practice, a user would need to empirically determine the distributions of the number of models returned per request, in order to be confident (to a given level) they have the required resources. Such undertakings may well be difficult or prohibitively expensive for some EC2 customers.

For the original account, the resources available in their AZs are very different, for example us-west-1c contains no E5-2650, whilst us-west-1b has 87%. As such they may well prefer to use us-west-1b as their main AZ, failing over to us-west-1c only when us-west-1b is unavailable. But us-west-1c is predominately comprised of the E5507 CPUs; and so any compute intensive task failing over to us-east-1c will either require more resource or take longer to run. For example, if running 20 m1.medium instances in us-west-1b we may expect, based on the percentages found, to have 17 E5-2650, 1 E5645 and 2 E5507; and failing over to us-west-1c they could expect 3 E5430, 2 E5645 and 15 E5507. If each instance was assigned the task of running our benchmark, then in us-west-1b, based on data in Table 5, this would take on average 4,600 s (to the nearest sec) with a worst case of 4,810 s. Whilst in us-west-1c we have an average of 6,443 s with a worst case of 6,867 s. Failing over results in an increase of 40% in the average time, and 43% increase in the worst case.

12 Differences in CPU distribution amongst instance types in same class

We have identified associations between sets of CPUs and instances classes. Clearly, the CPUs models backing a given class will change over time, but for the 6 month period we have been running experiments this has remained constant. This suggests medium term stability, and perhaps a desire on the part of EC2 to limit heterogeneity. The issue we consider in this section is whether or not there are differences in the resources obtained by instances types within the same class, in the same AZ. An understanding of the CPU distribution per AZ, potentially allows a user to choose the AZ that offers best performance.

To that end, using the new account, we ran 150 instances for each of 4 FGS instance types in AZ us-east-1a. The results are recorded in Table 16.

Table 16 FGS instances in us-east-1a

Type	*E5430*	*E5-2650*	*E5645*	*E5507*
m1.small	0	0	98%	2%
m1.medium	0	85%	15%	0
m1.large	0	67%	33%	0
m1.xlarge	0	85%	15%	0

Table 17 FGS instances in us-east-1a

Type	*E5430*	*E5-2650*	*E5645*	*E5507*
m1.small	0	0	97%	3%
m1.medium	0	91%	9%	0
m1.large	0	71%	29%	0

The most noticeable, and hard to understand, result of this experiment was the proportion of E5645 CPUs models backing m1.small instances as compared to the FGS2 instance types. Clearly, when sampling, there is a natural variation in the proportions we will obtain. Further, it would be reasonable to assume that the size of a VM may well affect the resources available to it in a given moment. For example, consider a host running an m1.xlarge and an m1.medium, and assume that 1 vCPU requires 1 physical core. Then, the host has three spare cores, so could run various combinations of m1.small, m1.medium and m1.large instances. However, to run an additional m1.xlarge would require over subscribing of the cores. How the size of an instance, in relation to its underlying physical host, effects how it is distributed is not yet something we have explored.

The second notable feature is the difference in resources that the m1.large instances obtain. A two sample proportion test rejects a null hypothesis that the m1.large CPU proportions are the same as the m1.medium, and so also the m1.xlarge. Under the current pricing scheme, there appears to be no financial incentives for EC2 to allocate resources differently to different instances types within the same class. We speculate that this may be an unintended side-affect of scheduling policies.

The experiment above was conducted on September 16. On the 4th of October (2013), and again using the new account, we started 100 instances (which is the maximum number of concurrent spot instances a user may run) of type m1.small, then m1.medium and finally m1.large in us-east-1a (m1.xlarge omitted for cost purposes). The results are recorded in Table 17.

The results of the second experiment are consistent with the findings of the first, in the sense that there is no evidence (at a 5% significance level) of any change in proportions. In a final experiment, on 8th October, we launched a 100 m1.small instances in us-east-1a, followed by 100 m1.mediums and we obtained 99 E5645 and 1 E5507, and 87 E5-2650 and 13 E5645, respectively.

In total, from 350 m1.small instances launched in us-east-1a over a three week period 98% were E5645 and 2% E5507, with no E5-2650, whilst for the FGS2 instances of the 750 launched approximately 80% were E5-2650 and 20% E5645. This is again supporting evidence for the view that the m1.small instances are being scheduled differently; either within the same set of resources or onto a different set.

Amazon do not, as far as we are aware release details of the scheduling polices that they have in place. One would assume they are designed to meet their own specific needs, be that minimising hosts in use so as to reduce power consumption, or to simplifying scheduling. There is little work that we are aware of, on how scheduling policies may affect the resources, and hence the ranges of price/performance a customer can obtain.

13 Consistency of proportions over time

In Section 11, we have already seen examples of consistency in the proportions obtained. Over a short period of time, we may expect the proportions of resources obtained to be affected by a number of things: an outage taking down a number of servers of a particular type, planned maintenance, and fluctuations in demand. However, this may well be offset by the scale of EC2. Clearly, we should expect changes over the longer term as regions, and AZ within them, grow. Indeed, the reasons that make public clouds heterogeneous will ensure that the resources we are able to obtain from them changes over time.

To further examine the consistency of the proportions of resources obtained we made a number of requests over a three week period for m1.medium instances in us-west-1c using the original account. We consider results from one request to the next to be inconsistent if we can reject the null hypothesis that the samples are drawn from the same population at 5% significance level. The results are recorded in Table 18.

Table 18 M1.medium instances in us-west-1c: 19/09-10/10

Date	*Sample size*	*E5430*	*E5-2650*	*E5645*	*E5507*
19/09	250	16%	0	9%	75%
29/09	80	20%	0	7%	73%
29/09	40	20%	0	10%	70%
29/09	40	15%	0	12%	73%
03/10	100	19%	0	8%	73%
03/10	100	15%	21%	12%	52%
03/10	99	10%	52%	8%	30%
08/10	100	0	82%	11%	7%
09/10	100	14%	0	14%	72%
10/10	100	21%	0	9%	70%

An abrupt change is evident in the resources being provided on 03/10. Although offering better performing CPUs (the E5-2650), such changes make predicting required levels of resource difficult. An auto-scaling application may well overprovision instances if they are basing the required number on past performance, which has an associated cost. On 09/10, and after, we obtain resources that are consistent with those obtained before 03/10.

It is open to question whether the E5-2650s are intended to be part of the resources we could obtain, and are simply often unavailable when we make our requests, or if our AZ mapping changed for during this period. That is, the location to which us-west-1c maps to was changed during this period.

14 Related work

Whilst we are not the only authors to be appraising cloud systems, the work presented here does help us to comment on related work, in some cases explaining the results of others, and in others commenting on why their work may require more rigorous follow-up. We take the opportunity to do so for a few such publications here.

In the influential paper '*Above the Clouds: A Berkeley View of Cloud Computing*' (Armbrust et al., 2008), the authors list performance unpredictability as their number 5 obstacle to cloud adoption. The unpredictability is a result of the variation they found when running the stream benchmark on 75 instances (instance type unspecified). The histogram they present is multi-modal and appears to be made up from three uni-modal distributions. The authors do not relate their results to potential differences in the underlying hardware. We conjecture that the variation found is due to CPU model differences, and from our own stream results, which we do not present in this paper; we find our distribution does indeed break down along these lines.

In Ward (2010), the author compares the performance of a local Private Cloud, based on Ubuntu Enterprise Cloud (UEC), with EC2. Section 1 states that "…we tested an EC2 virtual machine (VM) and a UEC VM of identical capacity against different criteria…". The authors appear to be using the term capacity to mean size and performance, and so assume that an m1.large on EC2 and an m1.large on UEC should provide the same level of performance, irrespective of underlying hardware. Whilst they are of the same 'size' the performance depends on the hardware characteristics of the compute node where the instance is running. This is a good example of the misunderstanding that can arise due to the lack of performance related information in machine descriptions.

In McGilvary et al. (2011), the authors studied performance and cost variability of EC2. They state that: "…the underlying processor of an instance can affect the performance of an instance of a user's job despite the purpose of ECU to obscure the differences". Although they identify hardware variation as a cause of performance variation their work does not extend to identifying all of the CPU models associated with each instance type, or quantifying instance type performance variation.

In Phillips et al. (2011), the authors state that the performance information provided by IaaS clouds is not sufficient to make a prediction on how an application will perform. They claim that micro benchmarks, based on the computational dwarf kernels, may offer better performance prediction than standard micro benchmarks. They run dwarf benchmarks across various instances on EC2 and on BonFIRE ('Infrastructure-Bonfire', http://www.bonfire-project.eu). Whilst benchmarking 10 m1.small instances they note that the scores fall into two statistically different performance classes and these classes were determined by CPU model. However, in the conclusions section they state 'some machines may have different clock speeds and cache size and even knowledge of this additional detail does not help in performance prediction'. However, we have shown that knowledge of CPU model can be used to predict our standard benchmark, and it is not unreasonable to assume we could perhaps, in future, use this to predict other CPU bound tasks that depend primarily on integer operations. This is something we intend to explore further in future.

In Ou et al. (2012), the authors identified different CPU models underlying the same instance types and then use this to estimate probability of obtaining a particular model as a method for optimising price/performance. The focus is on the US-East region alone; our work is more thorough in

considering a much larger range of regions, availability zones and instance types together with larger sample sizes. When formulating their cost model they also assume (without explicitly stating) that the level of performance provided by a given CPU is constant, and so only consider differences between models. We have shown this is not the case, indeed, we find (in a particular case) more variation in one model then they report between models. By considering the performance distribution of each model we find they are (in most cases) positively skewed, and in some cases we find maximum values (corresponding to time taken to complete a task) 3–4 standard deviations above the mean.

There are a number of papers investigating the suitability of EC2 for HPC (Akioka and Muroka, 2010; Evangelinos and Hill, 2008; Osterman et al., 2010), and they generally conclude that latency is too high on EC2 for MPI codes. Amazon does not yet offer dedicated low latency interconnects, however they do offer 10 GB Ethernet for cluster types, and such instances can be placed close together, as we have briefly discussed in Gillam et al. (in press).

15 Conclusions and future work

EC2 offers VMs in fixed sizes called instance types, the definition of which abstracts away underlying hardware details and to which is applied a computational rating called an EC2. This suggests homogeneous performance for heterogeneous hardware. However, as we have shown, the performance of instances of a given type is determined by the underlying CPU and differences in performance between two instances of the same type can be accounted for by the CPU model. However, whilst identifying CPU models associated with instances class can be useful to understand how price and performance may be related, it is not currently possible to specify the CPU model, or a desired level of performance (with respect to a given workload), in most requests. Consequently, it is not readily possible to predict the likely level of performance of an instance as this requires prediction of which CPU it will be backed by.

To address this problem, for each of the four classes of non-specialised instances – first generation standard, high CPU, high memory and second generation standard – we identified an associated set of underlying CPU models. The performance distribution of a CPU for a given class was then determined, and given knowledge of a CPU backing an instance we could use this data to determine the probability an instance obtains a particular level of performance. By estimating the proportion of each CPU model across zones, we can estimate likelihoods of obtaining a particular model in a given AZ. However, for this to be effective requires that:

1 the resources a user can obtain from a given AZ are consistent

2 the AZ mapping does not frequently change.

We would describe the results as relatively consistent; meaning that for a period of time the proportions of resources obtained are not so different as to be statistically different. And indeed, the association between an instance class and its set of CPU models appears stable, at least in the 6 month period covered by these experiments. Abrupt changes can occur though, leading to very different price/performance levels in an AZ. Further work on AZ modelling is required to understand the factors that influence the distribution of resources that a user can obtain. As already noted, scheduling is likely to be one such factor. Indeed, scheduling may well have some unintended side-affects such as a difference in the resources instances types within the same class may obtain.

Understanding the difference in CPU distribution can be beneficial as such differences can lead to difficulties for customers when either:

1 load balancing across AZs

2 having to fail over to another AZ.

In the largest regions, USEast N. Virginia and US-West N. California, new users are restricted to 3 and 2 AZs, respectively. A failure of one AZ can result in a user failing over to another which offers substantially different performance. This will result in either the user having to run more instances, and so essentially being additionally penalised for the zone failure, or potentially failing any SLAs they may have offered to their own customers for services on top of these instances. We have also shown that, due to the way EC2 maps AZ names to accounts, and limits the AZ a user has access to, different accounts obtain different resources. Hence, price/performance is dependent upon the account being used. It is open to question as to whether or not EC2 customers are aware of this.

We can suggest that some of the problems caused by performance variation could, in part, be solved by performance related pricing. Given that performance is related to the underlying CPU, simply pricing instances according the underlying hardware would account for the performance variation seen. However, there would still be some performance variation as a consequence of running VMs in a shared environment on technology that was not originally designed to be used in such a manner. Further, the variation is likely to be workload dependent and so different workloads on the same CPU may have different variations. Finer grained price/performance may be required by some customers, leading to a need for the pricing of workload specific SLAs. Until such a time as performance related pricing is used, it would be possible to exploit differential performance at the same price – for example, by a broker re-selling better performing instances to customers at higher prices to support more stringent SLAs.

References

'Amazon EC2 FAQs' [online] http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it (accessed 2 July 2013).

'bzip2' [online] http://www.bzip.org/1.0.5/bzip2-manual-1.0.5.html#intro (accessed 2 July 2013).

'Dwarf-Mine' [online] http://view.eecs.berkeley.edu/wiki/Dwarfs (accessed 2 July 2013).

'Elastic Block Storage' [online] http://aws.amazon.com/ebs/ (accessed 22 October 2013).

'Global Infrastructure' [online] http://aws.amazon.com/about-aws/globalinfrastructure/ (accessed 2 July 2013).

'Global Infrastructure' [online] http://aws.amazon.com/about-aws/globalinfrastructure/ (accessed 22 October 2013).

'Google Cloud Platform' [online] https://cloud.google.com/pricing/compute-engine (accessed 2 July 2013).

'Guest CPU Models' [online] https://access.redhat.com/site/documentation/en/Red_Hat_Enterprise_Linux/6/html/Virtualization_Getting_Started_Guide/para-CPU_Models.html (accessed 25 October 2013).

'HP Cloud Pricing' [online] https://www.hpcloud.com/pricing (accessed 2 July 2013).

'Infrastructure-Bonfire' [online] http://www.bonfire-project.eu (accessed 2 July 2013).

'SPEC CPU2006' [online] http://www.spec.org/cpu2006/ (accessed 2 July 2013).

'Xen' [online] http://wiki.xen.org/wiki/Credit_Scheduler (accessed 15 October 2013).

Akioka, S. and Muroka, Y. (2010) 'HPC benchmarks on Amazon EC2', in *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshop*, April, pp.1029–1034.

Armbrust, M. et al. (2008) *Above the Clouds: A Berkely View of Cloud Computing*, Technical Report EECS-2008-28, EECS Department, University of California, Berkeley.

Evangelinos, C. and Hill, C.N. (2008) 'Cloud computing for parallel scientific HPC applications: feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2', Presented at *Cloud Computing and its Applications 2008 (CCA-08)*, Chicago, IL, October.

Gillam, L., Li, B., O'Loughlin, J. and Tomar, A.P.S. (in press) 'Fair benchmarking for cloud computing systems', *Springer Open Journal of Cloud Computing*.

McGilvary, G., Barker, A., Atkinson, M.P. and Lloyd, A. (2011) 'Performance and cost variability of Amazon EC2', Presented at *AHM 2011*, York, September.

Osterman, S. et al. (2010) 'A performance analysis of EC2 cloud computing services for scientific computing', *Cloud Computing, Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, Vol. 34, pp.115–131.

Ou, Z., Zhuang, H., Nurminem, J.K., Yla-Jaaski, A. and Hui, P. (2012) 'Exploiting hardware heterogeneity within the same instance type of Amazon EC2', Presented at *4th USENIX Workshop on Hot Topics in Cloud Computing*, Boston, MA, June.

Phillips, S., Engen, V. and Papay, J. (2011) 'Snow white clouds and the seven dwarfs', in *Proceedings of the IEEE International Conference and Workshops on Cloud Computing Technology and Science*, November, pp.738–745.

Popek, G.J. and Goldberg, R.P. (1974) 'Formal requirements for virtualizable third generation architectures', *Communications of the ACM*, July, Vol. 17, No. 7, pp.412–421, ACM Press, New York, NY, USA.

Ward, J.S. (2010) 'A performance comparison of clouds Amazon EC2 and Ubuntu Enterpise Cloud', Presented at *SISCA DemoFest*, Edinburgh, November.