# Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds

Zhonghong Ou, *Member*, *IEEE*, Hao Zhuang, *Member*, *IEEE*, Andrey Lukyanenko, *Member*, *IEEE*, Jukka K. Nurminen, *Member*, *IEEE*, Pan Hui, *Member*, *IEEE*, Vladimir Mazalov, *Member*, *IEEE*, and Antti Ylä-Jääski, *Member*, *IEEE*

**Abstract**—Public cloud platforms might start with homogeneous hardware; nevertheless, because of inevitable hardware upgrades, or adding more capacity, the initial homogeneous platform will gradually evolve into heterogeneous as time passes by. The consequent performance heterogeneity is of concern to cloud users. In this paper, we evaluate performance variations from hardware heterogeneity and scheduling mechanisms of public clouds. Amazon Elastic Compute Cloud (Amazon EC2) and Rackspace Cloud are used as the representatives because of their relatively long record and wide usage among small and medium enterprises (SMEs). A comprehensive set of microbenchmarks and application-level macrobenchmarks have been used to investigate performance variation. Several major contributions have been made. First, we find out that heterogeneous hardware is a commonality among the relatively long-lasting cloud platforms, although the level of heterogeneity varies. Second, we observe that heterogeneous hardware is the primary culprit of performance variation of cloud platforms. Third, we discover that varied CPU acquisition percentages and different virtual machine scheduling mechanisms exacerbate the performance variation problem, especially for network related operations. Finally, based on the observations, we propose cost-saving approaches and analyze Nash equilibrium from cloud user perspective. By using a simple "trial-and-better" approach, i.e., keep good-performing instances and discard bad-performing instances, cloud users can achieve up to 30 percent cost saving.

**Index Terms**—Hardware heterogeneity, VM scheduling mechanism, performance variation, cloud computing, Amazon EC2

## 1 Introduction

As an industry-driven initiative, cloud computing gains a great deal of attention from various parties, including academia, industry, and policy makers, in the past few years. Several essential characteristics make cloud computing attractive for enterprises, for example, on-demand self-service, resource pooling, and rapid elasticity [1]. A range of vendors have provided various cloud services, for example, Amazon Elastic Compute cloud (Amazon EC2) [2], Rackspace cloud [3], Google Compute Engine [4], and Microsoft Azure [5].

Nevertheless, as times passes by, the likely homogenous platform at the beginning will gradually evolve into heterogeneous. Heterogeneity may originate from both hardware and software. From hardware perspective, server upgrades, adding more capacity, and network devices (switches and routers) replacement all contribute to enlarging heterogeneity; from software perspective, virtual machine (VM) schedulers evolution, and network topology changes of data centers both exacerbate the level of heterogeneity. Our previous work [6] represents one of the first studies exploring hardware heterogeneity in public clouds. It demonstrates that hardware heterogeneity exists in public cloud platforms, specifically in Amazon EC2, and contributes primarily to performance variations.

In this paper, based on our previous work [6], we take a step further to investigate the impact of hardware diversity and underlying scheduling mechanisms on the performance of public clouds. Amazon EC2 and Rackspace cloud platforms are taken as the examples. We also evaluated Microsoft Azure and Google Compute Engine cloud offerings. However, because of their relatively short period of time for service provision,[1] these two platforms have been using homogeneous hardware to date, which are of no interest to this paper. The motivation of this work is as follows:

1. Amazon EC2 and Rackspace cloud platforms represent the earliest cloud platforms that provide infrastructure-as-a-service (IaaS) to the public. They both have been introduced for a relatively long period of time (since 2006).[2] According to three-year hardware life cycle, in general, these two platforms should have experienced several generations of hardware upgrades. Thus, hardware heterogeneity

- *Z. Ou, A. Lukyanenko, J.K. Nurminen, and A. Ylä-Jääski are with the Department of Computer Science and Engineering, Aalto University, PO Box 15400, Konemiehentie 2, Espoo, FI-00076, Finland. E-mail: {zhonghong.ou, andrey.lukyanenko, jukka.nurminen, antti.yla-jaaski}@aalto.fi.*
- *H. Zhuang is with Distributed Information Systems Laboratory (LSIR), School for Computer and Communication Science, EPFL, BC 118 Station 14 CH-1015, Lausanne, Switzerland. E-mail: hao.zhuang@epfl.ch.*
- *P. Hui is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, China, and Telekom Innovation Laboratories, Ernst-Reuter-Platz 7, Berlin, Germany. E-mail: panhui@cse.ust.hk.*
- *V. Mazalov is with the Institute of Applied Mathematical Research, KRC of RAS, Russia. E-mail: vmazalov@krc.karelia.ru.*

*Manuscript received 25 Feb. 2013; revised 25 Aug. 2013; accepted 23 Oct. 2013; published online 31 Oct. 2013.*
*Recommended for acceptance by M. Demirbas.*
*For information on obtaining reprints of this article, please send e-mail to: tcc@computer.org, and reference IEEECS Log Number TCC-2013-02-0035.*
*Digital Object Identifier no. 10.1109/TCC.2013.12.*

1. Microsoft Azure was commercially available in 2010 and Google Compute Engine was introduced in 2012.

2. The tagline of Rackspace cloud service has been changed several times, and the tagline of Rackspace cloud was introduced in 2009.

and the consequent performance variation should be noticeable to cloud users.

2. Amazon EC2 and Rackspace cloud utilize customized versions of Xen hypervisor [7] to provision Linux-based VMs.[3] Previous work found out that different VM scheduling mechanisms in Xen pose varied impact on the performance of the VMs running atop [8]. Thus, there exists a high probability that different scheduling mechanisms also impact the instances of the public clouds.

We explore the two phenomena mentioned above in this paper. To that end, longitudinal measurements have been conducted. For Amazon EC2 platform, measurements from two periods of time have been conducted, one in 2011 and the other in 2012. For Rackspace cloud servers, the measurements are conducted in 2012. During the measurements, several interesting results have been observed that might affect the instance selection process for cloud users:

1. We observe that hardware diversity is a commonality between the two public clouds, although the level of heterogeneity varies. For Amazon EC2, hardware heterogeneity exists among the same instance type within the same data center; while for Rackspace, hardware heterogeneity only exists among different data centers.
2. We find out that hardware diversity is the primary source of performance discrepancy. Depending on the specific subsystem, performance variation ranges from 20 percent for CPU to 268 percent for memory, which all contribute to the overall application performance variation.
3. We discover that different VM scheduling mechanisms are used in Amazon EC2 within the same instance type. It is another major cause for performance diversity and unpredictability of the instances, especially for network related operations, for example, network throughput and network latency.
4. Depending on the specific cloud platform, we propose cost-saving approaches for cloud users to optimize their cloud usage. By using a simple "trial-and-better" approach, i.e., selecting better-performing instances to complete the same task, cloud users can acquire 16.7-30 percent cost saving. It is always beneficial for cloud users to use the "trial-and-better" approach if they intend to stay in the cloud for a relatively long period of time, for example, more than 10 hours.

The observations can be viewed from two aspects:

1. From technical perspective, as hardware upgrades are long lasting and inevitable procedures for data centers, more considerations should be taken in building a homogeneous cloud platform from heterogeneous hardware. This problem is challenging provided that new generations of hardware usually bring improvements from all subsystems, including CPU, memory, and hard disk. Adjusting one subsystem might result in imbalance from the other subsystems. Therefore, a holistic design is fundamental to achieve performance homogeneity.
2. From cost perspective, cloud users can utilize pricing inefficiency of cloud providers to optimize their cost of cloud usage. However, when a large portion of users start to use the "trial-and-better" strategy, this might reduce the achievable benefits from the approach. When to start using the strategy is a challenging game-theoretic problem. Furthermore, from the viewpoints of cloud providers, this "trial-and-better" behavior results in worse-performing instances never been used; thus, it causes significant resource wastage. How to detect this activity and introduce efficient approaches to prevent it requires both technical and business considerations.

The rest of the paper is structured as follows: Section 2 presents related literature. Section 3 details the environment setup. Section 4 describes the microbenchmark measurements, followed by application-level benchmarks in Section 5. Section 6 presents the potential cost saving approaches for cloud users. In Section 7, we conclude the paper.

## 2 Related Work

The existing research efforts can be divided into three categories: high-performance computing (HPC) oriented, system performance comparison, and exploiting heterogeneity in the cloud.

*HPC-oriented*. Walker [9] studied the performance of Amazon EC2 high-performance cluster compute instances. The study showed that there exists a performance gap between the EC2 provisioned cluster and local traditional scientific cluster. Jackson et al. [10] presented a comprehensive evaluation comparing conventional HPC platforms to Amazon EC2. Later on, Zhai et al. [11] compared Amazon EC2-based platform with typical local cluster and supercomputer options. They revealed that the high latency of Amazon EC2 cluster compute instances for small messages results in performance degradation for applications.

*System performance comparison*. Li et al. [12] developed a performance and cost comparator, i.e., CloudCmp, to measure various cloud services. Later on, they designed CloudProphet [13] to predict the end-to-end response time of an on-premise web application if migrated to a cloud. Lenk et al. [14] found out that the performance indicators provided by IaaS providers are not sufficient to compare different IaaS offerings. Wang and Ng [15] presented a measurement study on the impact of virtualization on network performance of EC2 platform. Schad et al. [16] analyzed performance variance of EC2 from different perspectives, including CPU and network performance. Barker and Shenoy [17] used a combination of microbenchmarks and two real-world latency sensitive applications for experimental evaluation.

*Exploiting heterogeneity in the cloud*. Suneja et al. [18] proposed to use graphics processing unit (GPU) acceleration to speed up cloud management tasks in virtual machine monitor (VMM). Lee et al. [19] introduced a scheduling mechanism in the cloud that takes into consideration

3. Hereafter, we use VM and instance interchangeably.

heterogeneity of the underlying platform and workloads. Through mathematical modeling, Yeo and Lee [20] found out that to achieve optimal performance, performance variation among a heterogeneous cloud infrastructure should be no larger than three times. Samih et al. [21] proposed an autonomous collaborative memory system that manages cluster memory dynamically. The key contribution behind the system is to dynamically detect nodes that have excess memory capacity (i.e., memory servers) and to provide means for nodes that are running short on space (i.e., memory clients) to swap their evicted kernel pages to the memory servers. Such a system proved to be feasible as it provides significant performance improvement to cloud clusters (up to $3\times$).

The work mentioned above focused on analyzing performance behavior of cloud platforms rather than investigating the underlying causes of various performance behavior. The latter is the focus of this work and clearly differentiates it from the previous ones. The most relevant work is from Farley et al. [22]. Similar to our previous work [6], Farley et al. also found out that hardware heterogeneity exists in Amazon EC2 standard small (*m1.small*) instance, which results in performance variations. Nevertheless, several differences exist to differentiate these two studies. First, only EC2 *m1.small* instance was covered in [22], which makes their results lack of generality, as small instances are likely used for testing purpose rather than for product-level provision because of their limited capacity; while this work covers large instance and other more powerful instances to provide a more generic investigation. Second, Farley et al. [22] provided rigorous simulation and experiment result from the strategy of exploiting heterogeneity, while this work investigates the case where every user exploits the heterogeneity using game-theoretic analysis. Furthermore, analysis of [22] is based on one-week long measurements, which makes their results, to a large extent, reveal temporary phenomenon rather than long-lasting behavior of clouds.

Compared to our previous work [6], this paper makes several differences:

1. in our previous study, we only took into consideration different processor models from EC2 on analyzing performance variation; in this work, both EC2 and Rackspace cloud are covered, and differentiation from other subsystems (memory, disk, CPU acquisition percentage) is also taken into account;
2. the impact of different scheduling mechanisms on performance variation is added;
3. utilizing open-source Xen hypervisor [7], a local environment is set up to assist understanding the underlying VM scheduling mechanisms; and
4. game-theoretic and Nash equilibrium analysis is added.

## 3 Environment Setup

In this section, we describe the environment setup and the benchmarks we used for the experiments. Ninety-five percent confidence intervals are used throughout the paper, where appropriate. If not otherwise stated, results illustrated in the figures are aggregated from 20 different instances.

### 3.1 Environment Setup

#### 3.1.1 Amazon EC2 Platform

For detecting the hardware configurations of Amazon EC2 instances, we cover most of the available instance types. Then, as a focusing and detailed research, we select the standard large instance type, denoted as *m1.large*, as the representative. Meanwhile, we carry out experiments on standard small (*m1.small*), standard xlarge (*m1.xlarge*), high-CPU medium (*c1.medium*), and high-CPU xlarge (*c1.xlarge*) instances, to supplement the evaluation. Most of the instances selected are located in us-east-1c availability zone of US East (Virginia) region. A self-created 64-bit Amazon EC2 Machine Image (AMI) with Cent OS 5.6 distribution is used to eliminate differentiation from operating systems (OS).

#### 3.1.2 Rackspace Cloud Platform

Similar to EC2 platform, we cover all the available instance types from Rackspace cloud. Both Dallas and Chicago data centers (regions) are covered. It is noteworthy that Dallas is a relatively new region in comparison with Chicago. Rackspace cloud servers differentiate instances by memory size, ranging from 512 MB to 30 GB. Then corresponding hard disk capacity and different number of virtual CPUs are configured with the instances. As a focusing research, we select the 4-GB memory instance as it provides similar capability as the EC2 *m1.large* instance type.

#### 3.1.3 Local Environment

To explore the underlying scheduling mechanisms of EC2, we set up a local platform based on Xen hypervisor to emulate the characteristics of EC2 instances. In the local environment, two servers are used to set up different versions of Xen hypervisor. The two servers have identical configurations: HP ProLiant BL280c G6 Server Blade with one Quad-Core Intel Xeon E5640 2,667-MHz processor, 8-GB DDR3 1,333-MHz memory, one HPNC362i dual port Gigabit server adapter. The hypervisors for the two platforms are Xen 3.4.3 (based on simple earliest deadline first, i.e., SEDF scheduler [23]), and Xen 4.1.1 (Credit scheduler [24]). Dom0 uses CentOS 5.6 distribution, as in Amazon instances. Varied number of VMs (each with different number of virtual CPUs) are running within the two machines.

### 3.2 Measurement Metrics and Tools

#### 3.2.1 Hardware Configuration

We acquire hardware information of EC2 and Rackspace cloud instances by using *cpuid* command, a nontrapping instruction that can be used in user mode without triggering trap to the underlying processor. Thus, the hypervisor could not capture the instruction and return modified results. Furthermore, we run *cat/proc/cpuinfo* command to verify the results from *cpuid*. The CPU models acquired from both approaches are identical.

For Amazon EC2 platform, we collect hardware information within two periods of time. One period is from

TABLE 1
Microbenchmark Tools

| |
|---|
| CPU performance: UnixBench [25] |
| VM scheduling: CPUBench |
| Memory performance: RAMspeed [26] |
| Disk performance: Bonnie++ [27] |
| Network throughput: TCPBench, UDPBench |

April through July in 2011; the other one is from January through May in 2012. For each period, we collect hardware information of 200 instances for each instance type, covering all availability zones in the US East (Northern Virginia) region. To make the instances under test as representative as possible, we measure instances at different time of a day, and during different days of a week. Furthermore, a smaller number of instances from other regions (e.g., US West, and EU) are also tested to confirm the existence of hardware heterogeneity among different regions. Note that the exact percentage of each type of hardware is of no significance, as the focus of this work is to reveal the existence of hardware heterogeneity in public clouds and analyze the consequent performance variation. For Rackspace cloud server platform, the hardware information is collected from June through October 2012.

#### 3.2.2 Microbenchmarks

We conduct a series of microbenchmark measurements to evaluate various aspects of the instances. The complete list is shown in Table 1, with supplementary description in the following texts where appropriate. The benchmark tool is the only process running on the instance when we conduct the measurements. In certain measurements, for example, CPU performance, two benchmark processes are used concurrently to measure the maximum capability of the instances, as *m1.large* and 4-GB instances possess two virtual cores.

*VM scheduling*. To investigate CPU sharing and understand the underlying scheduling mechanisms, we develop a CPU microbenchmark, *CPUBench*, to measure the CPU acquisition percentage of instances. The CPUBench records system time by making *gettimeofday()* system calls consecutively for one million times, the same approach as employed in [15]. In a regular call, the system call has a resolution in the order of microseconds. When the VM is scheduled off, the gap in the acquired system time is in the order of milliseconds. By analyzing the CPU running time and waiting time intervals, we can calculate the CPU acquisition percentage of the instance, and further derive the VM scheduling mechanisms of the underlying hypervisor. In VMs with multiple virtual CPUs, multiple *CPUBench* are run in separate processes to make full use of CPU capability.

*Network throughput*. We develop two fine-grained micro-benchmarks to measure TCP and UDP throughput, which we refer to as *TCPBench* and *UDPBench*. Each microbenchmark has a client and a server component. Both components are located in the same availability zone (or region). The server side of TCPBench calculates TCP throughput upon receiving per 256-KB data, while the UDPBench calculates throughput upon receiving per 128-KB data. To prevent potential bottleneck from the client side, the client components of TCPBench and UDPBench are deployed on powerful instances.

TABLE 2
Hardware Configuration of EC2 Instances

| Instance | CPU(Alias) | GHz | %(2011) | %(2012) |
|---|---|---|---|---|
| *m1.small* | **E5645** | **2.0** | 3% | 30% |
| | E5430 | 2.66 | 34% | 38% |
| | E5507 | 2.26 | 45% | 12% |
| | 2218HE | 2.6 | 18% | 20% |
| *m1.large* | **E5645(A1)** | **2.0** | 5% | 42% |
| | E5430(A2) | 2.66 | 29% | 17% |
| | E5507(A3) | 2.26 | 58% | 40% |
| | 2218HE | 2.6 | 4% | 1% |
| | 270 | 2.0 | 4% | - |
| *m1.xlarge* | **E5645** | **2.0** | 40% | 48% |
| | E5430 | 2.66 | 27% | 46% |
| | E5507 | 2.26 | 31% | 6% |
| | 270 | 2.0 | 2% | - |

#### 3.2.3 Application Performance Benchmark

We measure the web server performance of various instances using Httperf [28] benchmark tool. Apache web server is selected for serving static and dynamic requests. For static measurement, we send a HTTP request to fetch a file from the web server, and different file sizes are used. Dynamic HTTP request is used to fully utilize CPU capability. 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, dynamic web test is more CPU bound rather than network bound. Once again, a powerful instance from the same zone is used as the client.

## 4 Microbenchmark Measurements

In this section, we first analyze the hardware configuration of Amazon EC2 and Rackspace cloud servers. Then, we utilize several microbenchmark tools to evaluate the performance of various instances from different perspectives.

### 4.1 Hardware Configurations

As mentioned in Section 3.2.1, we use nontrapping *cpuid* instruction to acquire the underlying hardware information. The hardware configurations of various instance types from Amazon EC2 are listed in Table 2. It is noteworthy that we only list the *standard* instance family in Table 2. *High-CPU* instance family demonstrates similar behavior, i.e., multiple processor models are used within the same instance type. Note that Amazon introduced a new *standard* instance type (i.e., *m1.medium*) in the beginning of 2012. However, as we are more interested in hardware evolution of EC2 platform, we exclude *m1.medium* instance. Processor models for the other types of instances are relatively uniform, which are of no interest to this paper.

Hardware configurations of Rackspace cloud servers are listed in Table 3. As illustrated, Rackspace provides relatively coarse-grained instances, without further categorizing instances into different families. Furthermore, the hardware within the same region demonstrates less diversity compared with EC2 instances. For example,

TABLE 3
Hardware Configuration of Rackspace

| Instance (Memory) | Dallas | | Chicago | |
|---|---|---|---|---|
| | CPU(Alias) | GHz | CPU(Alias) | GHz |
| *512 MB*<br>*1 GB*<br>*2 GB* | 4170 | 2.1 | 4170 | 2.1 |
| *4 GB*<br>*8 GB*<br>*15 GB*<br>*30 GB* | 4170 (R1) | 2.1 | 2374 (R2) | 2.2 |

within the Dallas region, all the instances are hosted by AMD Opteron 4170 HE model; while within the Chicago region, two processor models are used, 4170 HE (512 MB, 1 GB, and 2 GB), and 2374 HE (4, 8, 15, and 30 GB). Recall that the Dallas region is a newly built data center, thus, its server machines are newer than the Chicago region. We can think of Rackspace cloud as a special case of EC2 platform, which provides *standard* instance family solely.

The processor models in Tables 2 and 3 starting by digits are from AMD Opteron series, while the rest are from Intel Xeon series. From the tables, several observations can be made:

1. Hardware heterogeneity is a commonality in EC2 and Rackspace cloud, although the level of heterogeneity varies. EC2 illustrates broad hardware diversity (intrainstance type and interinstance type), while Rackspace demonstrates heterogeneity only at the level of different regions. Newer processor models are replacing older ones progressively.
2. Same types of processor models are used within the same instance family. For example, E5645, E5430, and E5507 are used across the three different types within the standard instance family (*m1*) of EC2. On the one hand, this phenomenon can increase resource pooling, for example, *m1.small* instance can colocate with *m1.large* instance on the same physical server. On the other hand, hardware heterogeneity within the same subtype likely results in performance variation, which we will analyze in the subsequent sections.
3. When we collect information from EC2, we notice that the probability of a specific type of processor significantly varies in different availability zones. In one zone, we can acquire 95 percent of *m1.large* instances hosted by E5645 machines; while in another zone, the percentage of E5645 instances is as low as 10 percent. We believe that the availability zone with 95 percent E5645 instances is a newly built data center. This observation also explains performance of the same instance type varies significantly among different availability zones, which was revealed in previous work [16].
4. Furthermore, one more phenomenon is observed regarding E5645 processor model, which is bolded in Table 2. E5645 processor has a default clock speed of 2.4 GHz and can obtain a max turbo frequency of 2.67 GHz. However, during our experiments, we find out that the clock speed of E5645 processor is limited to 2.0 GHz. We will give an explanation in Section 4.3.1.

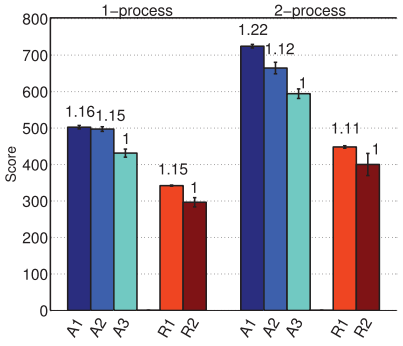



Fig. 1. UnixBench score, for one and two processes. A1-A3 instances are from Amazon EC2, while R1-R2 instances are from Rackspace.

Now, we are aware that EC2 and Rackspace utilize diversified CPU models to provision the same type of instance. The interesting question to ask is whether the heterogeneous hardware leads to diversified performance. Without loss of generality, we select *m1.large* instance from EC2 and 4-GB memory instance from Rackspace for focusing experiments in the subsequent sections. The aliases of these instances are sorted by their respective CPU capability, from powerful to weak. A1 and R1 stands for the most powerful instance from EC2 *m1.large* instances and Rackspace 4-GB instances, respectively. Apart from that because the AMD processor models (including 2218HE and 270) from EC2 account for too low percentage to be representative, we ignore them in the following analysis.

### 4.2 CPU Performance

From the previous section, we know that diversified processor models are used within EC2 *m1.large* instance and Rackspace 4-GB instance (between different regions). It is interesting to know how much performance variation the instances hosted by these processor models will present. We use UnixBench [25] to measure CPU performance, the results are depicted in Fig. 1. Note that we are more interested in comparing performance variations within the same cloud platform. Thus, we take the weakest instance as the baseline for each platform, i.e., A3 for EC2 *m1.large* instances (A1, A2, and A3) and R2 for Rackspace 4-GB instances (R1 and R2). Several findings can be made from Fig. 1:

1. The difference within the same subtype of instance is relatively small. For example, in both the 1-process and 2-process measurements, the gap between the upper and lower bounds of A1, A2, and A3 instances is small, which is a good indicator of small variation.
2. The differences between different subtypes are significant. If one process is running, A1 and A2 have comparable performance, while they are approximately $1.15\times$ of A3. When two processes are running, the performance variation in times is $1.22\times$ and $1.12\times$ for A1 and A2, respectively, against A3. The difference between Rackspace instances has similar trend.

TABLE 4
CPU Acquisition Percentage of Instances

| CPU | 1-proc (%) | 2-proc (%) | Emulation |
|---|---|---|---|
| A1 | 99.9 | (99.9, 99.6) | Credit (256, 200) |
| A2 | 75.7 | (75.4, 74.9) | SEDF (30, 40, 0) |
| **A2-s** | **99.4** | **(74.6, 74.8)** | **Credit (256, 150)** |
| A3 | 72.5 | (71.1, 71.9) | SEDF (150, 210, 0) |
| R1 | 99.9 | (98.1, 98.5) | Credit (512, 200) |
| R2 | 99.5 | (97.3, 95.7) | Credit (512, 190) |

Furthermore, it can be noticed that the differences of A1, A2, and A3 instances from UnixBench measurements are less than expected from PassMark benchmark. Calculated from PassMark CPU benchmark score [29], the difference between A1 and A3 instances should be $(6{,}898 - 3{,}632) / 3{,}632 = 89.92\%$. This discrepancy leads us to analyze CPU acquisition percentage and the underlying VM scheduling mechanisms.

### 4.3 CPU Acquisition and Scheduling

We use *CPUBench* (cf., Section 3.2.2) to analyze CPU acquisition percentage of the instances. By CPU acquisition percentage, we mean how much CPU processing time an instance can acquire from the hosting physical server.

#### 4.3.1 CPU Acquisition

The detailed CPU acquisition percentages for various instances are listed in Table 4. The notation *percent* means the percentage of the underlying physical CPU. To provide comparison, CPU acquisition percentage of EC2 *m1.small* instances is also measured, which is 50.6 percent (A1), 38.4 percent (A2), and 43.4 percent (A3).

Not surprisingly, *m1.small* instances (except A1) can acquire approximately 40 percent of CPU processing time, while *m1.large* instances (except A1 and A2-s) can acquire 75 percent of CPU time each for the dual-process CPUBench experiments. This is consistent with the advertised configurations of *m1.small* instances (one virtual core with one EC2 Compute Unit, *ECU*) and *m1.large* (two virtual cores with two *ECUs* each) instances.[4] For Rackspace, 4-GB instance type, one-process can acquire close to 100 percent CPU acquisition percentage; while for dual-process CPUBench, CPU acquisition percentage for each process varies between 95 and 99 percent, which we believe is because of admin overhead.

There exist two *m1.large* instances behaving completely differently from the others, i.e., A2-s (A2 special) and A1. A2-s instance can acquire close to 100 percent of CPU acquisition percentage for single process, while 75 percent each for dual-process. However, the A2-s instances account for only a small percentage, less than 1 percent of overall. Thus, we leave A2-s for discussion in Section 4.3.3. A1 instances can acquire close to 100 percent CPU acquisition percentage for both single and dual process. Several conjectures can be made to explain the irregularity of A1 instances:

1. Fifty percent (for *m1.small*) and 100 percent (for *m1.large*) CPU acquisition percentages are easier to manage and likely more efficient than the 40 and 75 percent CPU allocation. For example, one physical core (without hyperthreading) is able to host two *m1.small* instances with 50 percent CPU acquisition percentage, while two physical cores (without hyperthreading) together can host one *m1.large* instance with 100 percent CPU acquisition percentage. In the case of 40 percent CPU utilization, one physical core can host two small instances. However, the remaining 20 percent CPU time has to be combined with another physical core or will otherwise be wasted. Rackspace cloud also uses simple and uniform close to 100 percent CPU utilization for all its instances.
2. Amazon intends to increase the overall performance of its instances within certain extent. Amazon is criticized for lagging behind Moore's law because EC2 instances today present the same performance level as several years back. Thus, it makes sense to improve the overall performance by utilizing newer hardware and higher CPU acquisition percentage. However, the improvement should be within certain extent to avoid discernible performance gap for end users. Thus, the frequency of A1 processors are limited to 2.0 GHz rather than the default 2.4 GHz, which presents an explanation to the phenomenon we observed in Section 4.1 (cf., Table 2).

It is also worth noting that the CPU performance variation among various *m1.large* instances, as shown in Section 4.2, is approximately within 20 percent. This variation is consistent with the EC2 statement,[4] which implies an estimated 20 percent difference.

4. One *ECU* provides the equivalent CPU capacity of a 1.0-1.2-GHz 2007 Opteron or 2007 Xeon processor [2].

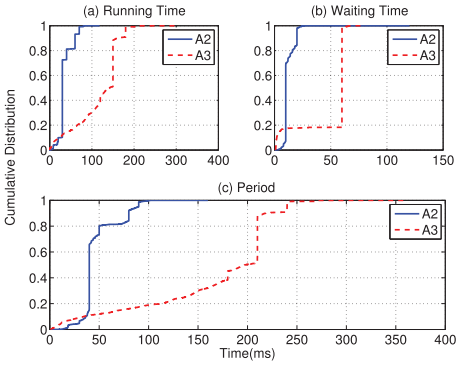



Fig. 2. Running time, waiting time, and periods of A2, A3 instances (single process).

#### 4.3.2 VM Scheduling

Being aware that A2 and A3 instances are sharing CPU resources with other instances, we analyze the VM scheduling mechanisms in this section. We use CPUBench, as mentioned in Section 3.2.2, to analyze the *running time*, *waiting time*, and *periods* of A2 and A3 instances. A *period* is a complete cycle that is the summation of running time and waiting time. The cumulative distribution functions (CDFs) of A2 and A3 instances are depicted in Fig. 2. Note that because the A1, R1, and R2 instances acquire close to 100 percent CPU acquisition percentage, it can be thought of as running all the time without any waiting period.

From Fig. 2, it can be seen that A2 behaves more regularly than A3 instance. Approximately 70 percent of time, the running time, waiting time, and period of A2 instances is 30, 10, and 40 ms, respectively, which turns into $30/40 = 75\%$ CPU acquisition percentage. A3 instances demonstrate less regularity. Still, for around 40 percent of time, the running time and period is 150 and 210 ms, respectively. The waiting time of A3 instances shows the most regularity, i.e., over 80 percent of time, the waiting time of A3 instances is 60 ms.

The observation we can make from the VM scheduling of various *m1.large* instances is: Amazon EC2 uses different VM scheduling mechanisms for its instances. For A2 instances, it uses "short-and-fast" mechanism, while for A3 instances, it uses "long-and-slow" approach. The "short-and-fast" mechanism increases the overhead of VM switching, but it is beneficial for latency sensitive applications. The "long-and-slow" mechanism has benefits in lowering VM switching overhead; however, it harms the responsiveness of applications. The impact of scheduling mechanisms on the networking performance will be analyzed in Section 4.4.

#### 4.3.3 Discussion

Recall in Section 4.3.1 that there exist two different types of A2 instances, one is regular (A2) and the other one is special (*A2-s*). To further investigate the irregularity of A2 instances, we build a local environment (cf. Section 3.1.3) to emulate various behaviors of the instances. Two well-used schedulers from Xen community are selected in this paper, i.e., simple earliest deadline first scheduler and Credit scheduler. The detailed parameters for the emulation are listed in Table 4, *Emulation* column. Refer to [8] for detailed meaning of the parameters.

As shown in Table 4, we are able to successfully emulate the EC2 and Rackspace instances. Nevertheless, for EC2 instances, two different schedulers have to be used, i.e., SEDF and Credit. With the SEDF scheduler, we are not able to achieve the desired CPU acquisition percentage for *A2-s* instance. With the Credit scheduler, we are not able to emulate the A2 and A3 instances. This observation indicates the existence of different scheduling mechanisms of Amazon EC2, which supplements the conclusion we reach in Section 4.3.2. Different VM scheduling mechanisms contribute to performance variations within the same instance type, especially for network related operations.

### 4.4 Network Performance

As EC2 utilizes different VM scheduling mechanisms (cf. Sections 4.3.2 and 4.3.3) and varied CPU acquisition percentages (cf. Section 4.3.1), we analyze its impact on network performance in this section. Note that besides TCP throughput, UDP throughput and network latency experiments are conducted. The other two experiments demonstrate similar trends as TCP throughput, thus, they are omitted because of space.

We use TCPBench to measure TCP throughput of the instances. The results are depicted in Fig. 3. Note that the tests are run for 30 seconds, but 1-second curve is sufficient to represent the general trends for 30 seconds. The average TCP throughput for these instances is: 880 (A1), 883 (A2), 875 (A3), 248 (R1), and 243 Mbps (R2). Several tendencies can be observed:

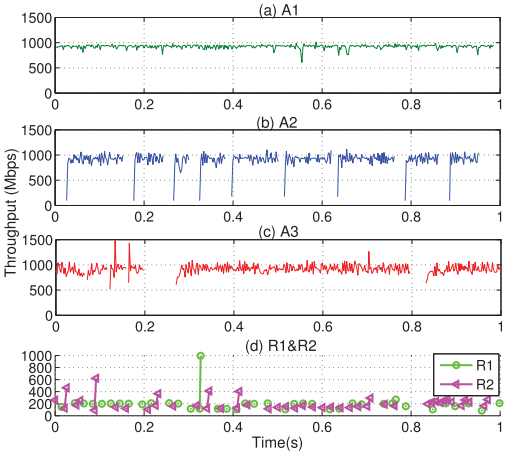



Fig. 3. TCP throughput of various instances. (a) A1 instance; (b) A2 instance; (c) A3 instance; and (d) R1 & R2 instances.

1. As expected, A1 demonstrates the most stable TCP throughput among all EC2 instances. It can be explained by the fact that A1 instance acquires close to 100 percent CPU acquisition percentage. On the other hand, both A2 and A3 instances present intermittent transmission behaviors. Nevertheless, compared with A3, A2 instance illustrates shorter and more frequent intermissions. Recall from Section 4.3.2 that the scheduling intervals for A2 are much shorter (40 ms) than for A3 instances (210 ms) (cf. Fig. 2). Therefore, A2 instances are more frequently interrupted for shorter periods of time because of VM scheduling. In general, EC2 instances can acquire close to 900-Mbps internal TCP throughput, and no traffic control policies are observed.
2. R1 and R2 instances both provide close to 200-Mbps internal TCP throughput, consistent with Rackspace promises [3]. Dissimilar to EC2, which fully utilizes its network capacity, Rackspace employs traffic control policies toward its instances, both internally and externally. Thus, traffic flows from Rackspace are frequently interrupted. In Fig. 3d, two dots without a line connecting means the gap is larger than 10 ms. Compared with R2, R1 instance is more stable.

The performance behavior of TCP throughput from EC2 instances supplements the conclusion we made in Section 4.3.2 that different VM scheduling mechanisms pose a significant effect on networking performance. As for Rackspace instances, traffic control policies dominate the overall networking behavior.

### 4.5 Memory Performance

From the sections above, we are aware that the diversified processor models have a significant impact on the CPU performance. We continue to investigate the impact of memory on the performance of the instances. To that end, RAMspeed is used, and the results are depicted in Fig. 4.

From the figure, it can be seen that A1 instance performs the best, A2 instance functions the worst (thus taken as the

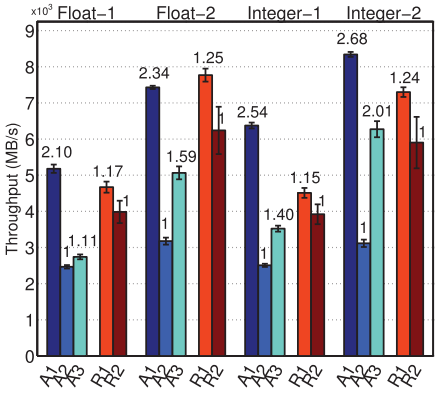



Fig. 4. Memory performance. Float- and Integer-stand for float and integer operations, and 1 and 2 stand for 1 and 2 processes, respectively.

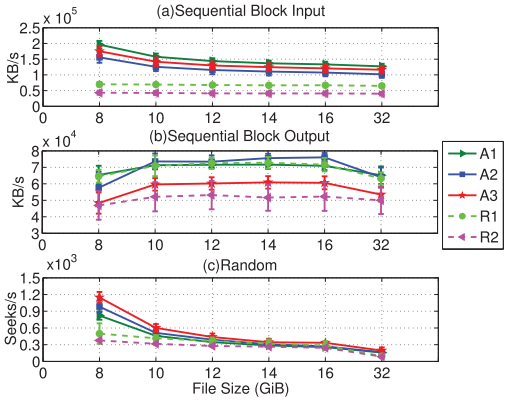



Fig. 5. Disk throughput. (a) Sequential block input; (b) sequential block output; (c) random operations.

baseline), while A3 sits in between. The difference between A1 and A2 instances in single-process integer operation, i.e., Integer-1, is $2.54\times$; while the difference in dual-process integer operation is $2.68\times$. The difference between A3 and A2 instances for integer operations is $1.40\times$ and $2.01\times$, for single and dual process, respectively. The float operations show relatively smaller difference than integer operations. However, the difference is still significant. For Rackspace instances, consistent with CPU performance, R1 outperforms R2 from memory perspective. The difference between the two instances, however, is smaller compared with EC2 instances.

At a first glance, the results from EC2 instances are counterintuitive, as A2 processor demonstrates better CPU performance than A3 processor. Recall from Section 4.1 that A1 and A3 processors were released in Q1'10, while A2 processor was released in Q4'07. Both A1 and A3 processors adopt Intel QuickPath Interconnect (QPI) technology, while A2 employs front-side bus (FSB) technology. QPI presents superior performance to FSB technology and has been replacing the latter gradually. Furthermore, A1 and A3 processors can support DDR3 memory; while A2 most likely uses DDR2 memory, as DDR3 was first in use in 2007. As for R1 and R2 instances, they have similar configurations for L2, L3 cache and memory type. Thus, their performance does not reveal large difference as EC2 instances, but still 25 percent difference is illustrated. As a summary, hardware diversity from memory type and architecture presents significant impact on memory performance variation, the difference can reach up to multiple times.

### 4.6 Disk Performance

Bonnie++ [27] is used to measure the hard drive and filesystem performance. Sequential block input (read), sequential block output (write), and random seek benchmarks are conducted, the results are shown in Fig. 5. Be aware that these measurements are more disk bound than CPU bound.

For EC2 instances, Fig. 5 shows that there exists no clear winner from all aspects. A1 performs slightly better than A2 and A3 in sequential block input operation (cf. Fig. 5a), while it functions the worst for random seek operation (cf. Fig. 5c). A3 instance is the winner in random seek operation, while it is the worst in sequential block output operation (cf. Fig. 5b). A2 instance functions the best in sequential block output operation, while the other two operations are worse than A1 and A3.

For Rackspace instances, the performance of disk is consistent with other components, for example, CPU and memory. R1 outstrips R2 from all three aspects. However, the difference from disk performance is larger than CPU and memory. As illustrated from Figs. 5a and 5b, the difference is $0.7/0.4 = 1.75\times$ for sequential block input, and $7/5 = 1.4\times$ for sequential block output operations. As mentioned before, Dallas (R1 instance) is a newly built data center, disk storage is also naturally newer than the Chicago (R2 instance) region. Thus, it is understandable that R1 outperforms R2 instances from every aspect we investigated.

The disk performance of EC2 instances is the most irregular one among the subsystems we investigated. It is difficult to investigate the real reasons for the irregularity. In general, disk performance is a complex issue that involves a number of aspects. We conjecture that the relative weak random seek performance of A1 instances is from the limited share of hard disk cache. Recall that the A1 processor has six physical cores and is hyperthread enabled. Thus, theoretically, one A1 processor is able to host six EC2 *m1.large* instances. The number for A2 and A3 processors is $4/(2 \cdot 0.75) = 2$, and $4/(2 \cdot 0.75) = 2$, respectively. Thus, to acquire the same allocation of disk buffer, the hard disk(s) for A1 processors should provision $3\times$ of disk buffer of A2 and A3.

One conclusion can be made: for EC2 instances, disk performance is less predictable than the performance of the other subsystems (e.g., CPU and memory), there is no clear winner for disk performance; for Rackspace instances, disk performance variation is bigger compared with other subsystems.

## 5 Application-Level Benchmark

Being aware of the varied performance from heterogeneous hardware subsystems, we analyze their accumulated effect on application performance. Two applications are selected, one is web server, and the other is in-memory database. Because the results from these two applications present similar trends, we only present web server throughput measurements.

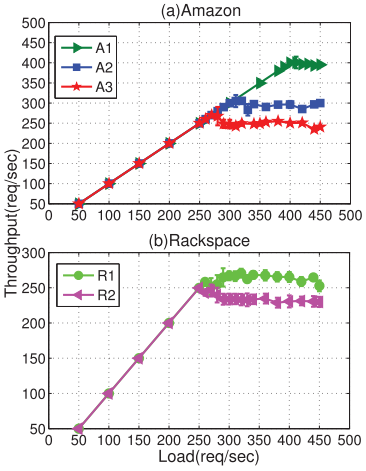



Fig. 6. Httperf performance for different instances. (a) Amazon instances; (b) Rackspace instances.

As stated in Section 3.2.3, both static and dynamic measurements have been conducted. Nevertheless, the static measurements are bound to network bandwidth, of which the instances have very close performance. The Httperf throughput for dynamic measurements is depicted in Fig. 6. Fig. 6a illustrates that the throughput of A1 and A2 is $1.6\times$ and $1.2\times$ of A3, respectively. This phenomenon implies that the strengths from several subsystems (CPU and network) are accumulated from application perspective. Recall from Section 4.2 that CPU performance of A1 and A2 is $1.2\times$ and $1.1\times$ of A3 instance. Thus, network performance of A1 and A2 instances poses a positive effect on their web server throughput. For Rackspace instances (R1 and R2), however, the web server throughput variation is consistent with CPU microbenchmark, i.e., 20 percent difference. This is mainly because the internal network throughput of Rackspace instances is limited to 200 Mbps; thus, CPU performance variation dominates the overall variation from application perspective.

## 6 Cost Optimization Analysis

We propose cost saving approaches, conduct game-theoretic analysis, and discuss Nash equilibrium in this section.

### 6.1 Trial-and-Better

Given a certain task, the "trial-and-better" approach works by seeking for better-performing instances from the same instance type to complete the task. A cloud user can take the following steps to fulfill the approach: 1) Apply for certain number of instances from the cloud (naturally, the instances acquired are a mixture of better-performing and worse-performing instances with certain percentage each); 2) Check the performance levels of the acquired instances, for example, checking *cpuid* information; 3) Keep the better-performing instances, discard the other ones, and then apply for new instances from the cloud. By iterating the aforementioned procedure for multiple rounds, the user will eventually get the desired number of better-performing instances. The rationale behind this approach is that from our several-month long measurements from Amazon EC2, instances are returned relatively randomly. Namely, every time after we terminate one instance and apply for a new one, the new machine is with randomized hardware.

TABLE 5
Notations and Their Meaning

| | Definition |
|---|---|
| $c$ | Hourly cost of an instance. |
| $T$ | Number of hours needed to complete the task. |
| $m$ | Number of different sub-types within the same type. |
| $q$ | Number of instances needed to complete the task with worst-performing instances. |
| $N$ | Total number of instances hosted by a cloud provider. |
| $N_i$ | Number of instances hosted by a specific hardware. |
| $p_i$ | Probability of an instance hosted by a specific hardware; note, $p_i = \frac{N_i}{N}$. |
| $\alpha_i$ | Stretch factor meaning how many additional instances a cloud user should apply besides the desired instances. |
| $x_i$ | Performance level (in times) compared with the baseline instance. |
| $X$ | A random variable stands for performance level. |
| $C$ | The total cost of completing the task. |

Furthermore, one common scenario in clouds is VM migration. From our measurements, VM migration does exist in public cloud platform, for example, Amazon EC2. Nevertheless, from our two periods of measurements, it occurs very rarely. This observation makes the "trial-and-better" approach work well in public clouds.

Note that given a task, 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 a shorter period of time. From cost perspective, the two options are the same. We take the first alternative as the example for analysis. The notations are defined in Table 5.

Applying for an instance from a cloud randomly, the probability of the instance of a certain subtype is $p_i$, and the expected value of the performance level of the instance is defined as follows:

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

Given a task equivalent to $q \cdot T$ hours work using worst-performing instances, the total cost of completing the task using randomly allocated instances from the cloud, i.e., $C_{random}$, can be derived by the following equation:

$$C_{random} = q \cdot T \cdot c / E(X). \tag{2}$$

If we aim to select better-performing instances to complete the same task by utilizing the "trial-and-better" approach, then a smaller number of instances are needed. The cost of completing the real task (excluding the additional cost for selecting the better-performing instances) is

$$C_{opt} = q \cdot T \cdot c / x_{opt}. \tag{3}$$

Herein, $x_{opt}$ stands for the performance level of the better-performing instance. Furthermore, the "trial-and-better" process results in additional cost for acquiring the better-performing instances, which is

$$C_{extra} = q \cdot c / (p_{opt} \cdot x_{opt}), \tag{4}$$

wherein $p_{opt}$ denotes the probability of the better-performing instances in the overall instances.

Here, we assume that the process of finding the better-performing instances takes no more than 1 hour. As a matter of fact, the process of selecting the better-performing instances is very straightforward and fast. The process of acquiring instances from EC2 takes from 2 to 4 minutes from our measurements. We also assume that cloud users run the rented instances for a relatively long period of time, at least much longer than 1 hour. Thus, it makes the additional hours spent on selecting instances much worth, which we will demonstrate through the subsequent analysis. Furthermore, we assume that the number of instances required by a cloud user is relatively small compared to the population of available instances.

Thus, compared with the random scenario (taking the randomly allocated instances from the cloud platform to complete the task), the potential cost saving for the aforementioned optimized scenario can be calculated as follows:

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

Put (1), (2), (3), and (4) in (5), we can acquire the following equation:

$$C_{saving} = \left( T \Big/ \left( \sum_{i=1}^{m} x_i \cdot p_i \right) - T/x_{opt} - 1/(p_{opt} \cdot x_{opt}) \right) \cdot q \cdot c. \tag{6}$$

The cost saving in percentage is

$$C_{percent} = C_{saving}/C_{random}. \tag{7}$$

Take the EC2 *m1.large* instance as the example, three different subtypes (A1, A2, and A3) exist within this type. The probability of each subtype is 42 (A1), 17 (A2), and 40 percent (A3) (cf. *m1.large* instance in Table 2, column percent *(2012)*). We take the performance level of A1 and A2 instances as $1.6\times$ and $1.2\times$, respectively, against A3 instance (cf. Fig. 6). The unit cost of a regular on-demand *m1.large* instance is \$0.26/hour for Linux instance [2]. Put all these values in (6), we can acquire the following equation:

$$C_{saving} = 0.26 \cdot q \cdot (0.1587 \cdot T - 1.4881). \tag{8}$$

To achieve cost saving, the requirement is $C_{saving} > 0$, then we can get the necessity: $T > 9.4$. Namely, given the aforementioned probability of each instance and its respective performance level, it starts to make sense to select A1 instances to complete the task if the required running time is larger than 10 hours, which is not a demanding requirement.

Furthermore, recall from Section 4.1 that, for EC2 platform, varied processor models are not distributed uniformly among the availability zones, but rather different processor model dominates different availability zone. Thus, it is also interesting to analyze only two types of hardware configuration. To that end, the results of (7) with two subtypes are depicted in Fig. 7. Wherein $p$ stands for the probability of the better-performing instances, and $x$-axis stands for the performance level in times ($x_i$ in Table 5).

Understandably, if better-performing instances account for the majority of the overall instances, for example, $p = 0.9$ (cf. Fig. 7), without a selection process, the probability of acquiring a better-performing instance is very high. Thus,

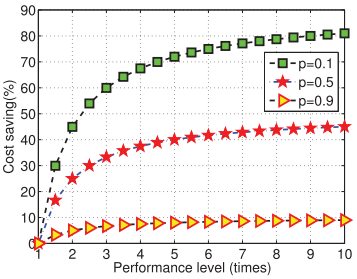



Fig. 7. Cost saving analysis. Different $p$ value stands for different percentage of better-performing instances.

the performance is close to the optimal situation with the selection process, and the cost saving achievable is small (less than 10 percent). However, as the better-performing instances account for increasingly smaller proportion of the overall instances, the cost saving achievable becomes significant. In the case of $p = 0.1$, if the better-performing instance is 10 times as efficient as the baseline (worse-performing) instance, the cost saving can reach up to 80 percent. Obviously, this is an unrealistic situation with all the efforts that Amazon contributes to make the same type of instance function closely. Nevertheless, from Sections 4 and 5, we know that 1.2-1.6× variation is highly possible. With $1.5\times$ variation, the achievable cost saving can still reach *30 percent* (cf. the square curve in Fig. 7 where the value of $x$-axis is 1.5, we can see the value of $y$-axis is approximately 30 percent), which is a significant saving. For Rackspace users, the decision-making process is straightforward, i.e., always preferring Dallas region to Chicago region. Let us assume on average R1 is $1.2\times$ of R2, then the achievable cost saving is still remarkable, *16.7 percent*.

### 6.2 Game-Theoretic Analysis

In the previous section, we analyzed the optimal behavior of one cloud user in the presence of regular behavior of others. It is obvious that if everyone starts to use the same strategy, the number of available better-performing instances will be reduced. To deal with this situation, we model a 2-player game-theoretic problem: two players intend to acquire $q_1$ and $q_2$ instances for time $T$. Two-player model is sufficient to investigate the interrelationship among individual cloud users. Because in the cloud, for a specific cloud user, he is not competing against any individual cloud user, but rather the other users as a whole. Furthermore, varied individual behaviors can be formulated by a single unified behavior. Thus, the whole cloud users can be modeled as a *small* individual user and a *large* combined user, from the perspective of occupied resources. Without loss of generality, we assume that there exist only two types of instances in the cloud ($N_1$ and $N_2$, and $x_1 > x_2$).

The gaming strategy for each cloud user is a *stretch factor* ($\alpha_i \geq 1$) representing how many additional instances the user should request from the cloud. Herein, $\alpha_i = 1$ stands for *regular behavior*, while $\alpha_i > 1$ stands for *selfish behavior*. *Regular behavior* means the cloud user takes whatever is offered from the cloud without any selection process. *Selfish behavior* means the cloud user applies for more than the desired number of instances from the cloud, and then takes the resources as much as possible from better-performing

instances acquired, while the rest (if any) from worse-performing instances. The "trial-and-better" approach mentioned in the previous section belongs to *selfish behavior*. Be noted that $\alpha_i \cdot q_i \leq N$.

Consider the scenario that one player comes first to the cloud, where all instances $N_1$ and $N_2$ are available for use, and then the other player comes (when some resources are already occupied by the first player). We assume that the probability of each player coming to the cloud first is equal (this can easily be extended to unequal case). Then, we can acquire the following utility function, which is defined as the performance divided by the cost.

**Theorem 1.** *The utility function ($u_i$) for player $i$ in such a game is*

$$u_i(\alpha_1, \alpha_2) = \frac{T \cdot (x_2 \cdot q_i + \frac{1}{2}(x_1 - x_2) \cdot v_i(\alpha_1, \alpha_2))}{c \cdot q_i \cdot (\alpha_i - 1 + T)}, \quad (9)$$

*where $i, j = 1, 2$ and*

$$\begin{aligned} v_i(\alpha_1, \alpha_2) = Min\Big\{ & q_i, \frac{N_1}{N} \cdot \alpha_i \cdot q_i \Big\} \\ & + Min\Big\{ q_i, N_1 - Min\Big[ q_j, \frac{N_1}{N} \cdot \alpha_j \cdot q_j \Big], \\ & \frac{N_1 - Min\{q_j, \frac{N_1}{N} \cdot \alpha_j \cdot q_j\}}{N - q_j} \cdot \alpha_i \cdot q_i \Big\}, i \neq j. \end{aligned} \quad (10)$$

Because of space limit, proofs of theorems are not presented in this paper. Given the utility function, each player will try every strategy out to maximize his own benefits, eventually this will result in Nash equilibrium.

### 6.3 Nash Equilibrium

We first analyze the potential states that lead to Nash equilibrium, which are defined as follows.

**Theorem 2.** *The candidates for Nash equilibrium from utility function (10) are the following points:*

$$\alpha_{1,2} = 1, \quad \alpha_{1,2} = \frac{N}{N_1}, \quad \alpha_i = \frac{N}{q_i}, i = 1, 2,$$

$$\alpha_i = \frac{N - q_j}{N_1 - q_j}, i = 1, 2, \quad \alpha_i = \frac{N - q_i}{q_j}, i = 1, 2, i \neq j.$$

Theorem 2 means that the potential candidates for Nash equilibrium are tightly related to the distribution (i.e., $q_i$) of instances. Meanwhile, it illustrates that the number of candidates is very limited. We then analyze the respective conditions under which *regular behavior* ($\alpha_i = 1$) and *selfish behavior* ($\alpha_i > 1$) are preferable. The condition leading to *regular behavior* is defined as follows.

**Theorem 3.** *Assume the following condition takes place:*

$$T - 1 \leq \frac{x_2}{x_1 - x_2} \frac{N}{N_1}. \quad (11)$$

*Then, equilibrium is acquired at: $\alpha_i = 1, i = 1, 2$.*

Theorem 3 means that for users staying in the cloud for a relatively short period of time, it is preferable to take whatever is given by the cloud. This is straightforward to understand. We now analyze conditions that lead to *selfish behavior*, the following theorem is one possible form.

**Theorem 4.** *Assume that $q_1 < N_1$, $q_2 \geq N_1$, and the following condition is met:*

$$T - 1 \geq \frac{N}{N_1} \left[ \frac{2 \cdot x_2}{x_1 - x_2} + \frac{N_1 - q_1}{q_2} \right]. \quad (12)$$

*Then, the Nash equilibrium is acquired at*

$$\alpha_1 = \frac{N}{N_1}, \quad \alpha_2 = \frac{N}{q_2}.$$

Theorem 4 describes a scenario that is close to real life, where we have one *small* user (an individual user) and one *big* user (a combined user from a number of small users) in the cloud. The number of instances the *small* user applies is significantly less than the total number of better-performing instances in the cloud, i.e., $q_1 < N_1$; while the number of instances applied by the *big* player is larger than that, i.e., $q_2 \geq N_1$, which is to guarantee a reasonable usage of cloud resources. The received Nash equilibrium can be treated in terms of evolutionary stable strategy (ESS). It means that if all cloud users are using one strategy and an individual cloud user decides to change his strategy, then the strategy that the majority has adopted will be optimal. Theorem 4 states that if the cloud reveals performance variation within the same instance type, as long as the cloud user stays in the cloud for a long enough period of time, it is always beneficial to take *selfish behavior* $\alpha_i = \frac{N}{N_1}$, instead of the default *regular behavior* $\alpha_i = 1$.

*Numerical example*. Now consider a generic example: $x_1 = 1.5$, $x_2 = 1$, $N = 100{,}000$, $N_1 = 10{,}000$, $N_2 = 90{,}000$, $c = 0.26$, $T = 100$, $q_1 = 100$, $q_2 = 40{,}000$. It is close to what we acquired from EC2 cloud. The cloud utilization level is moderate ($(q_1 + q_2)/N = 40.1\%$) , and better-performing instances account for $N_1/N = 10\%$. To find Nash equilibrium, we need to consider combinations of possible strategies for both players and their utility:

| $\alpha_1$ | 1 | 1 | 10 | 10 |
|---|---|---|---|---|
| $\alpha_2$ | 1 | 2.5 | 1 | 2.5 |
| $u_1$ | 4.03846 | 3.94231 | 5.29287 | 4.41073 |
| $u_2$ | 4.03846 | 4.26274 | 4.0376 | 4.26061 |

Using the best response approach (the most favorable outcome for a cloud user taking the other users' strategies as a given), the Nash equilibrium is at point: $\alpha_1 = N/N_1 = 10$ for player I (*small*), and $\alpha_2 = N/q_2 = 2.5$ (an upper bound) for player II (*big*), where the cost saving is 9.22 and 5.5 percent, for players I and II, respectively. The optimal case for player I is at (10, 1), where its cost saving is $(5.29287 - 4.03846)/4.03846 = 31.06\%$ and player II is barely influenced. If player I stays with *regular behavior*, while player II takes *selfish behavior*, i.e., (1, 2.5), then the loss for player I is remarkable. Thus, considering the potential loss, the relative gain acquired by taking *selfish behavior* for player I is even larger, $(4.41073 - 3.94231)/3.94231 = 11.88\%$ (compared with 9.22 percent). For player II, compare *selfish behavior* ($\alpha_2 = 2.5$) with *regular behavior* ($\alpha_2 = 1$), the cost saving is around 5.5 percent, no matter what strategy play I takes. As a conclusion, in real-life cases, even though the exact cost

TABLE 6
Speedup and Overhead of "Trial-and-Better" Approach

| | Speedup | Overhead |
|---|---|---|
| Redis | 11.5% | 1.5% |
| Httperf | 28.8% | 1.8% |

saving varies according to opponents' strategies, it is always preferable to take *selfish behavior*. This applies to both *small* and *big* players. Nevertheless, *small* users can achieve significantly more cost saving than *big* ones.

### 6.4 Validation

To validate the proposed "trial-and-better" approach, we implement two applications in Amazon EC2. One application is in-memory database, for which we use Redis from YCSB [30]; the other one is web-based application, for which we use Httperf and use the same configuration as in Section 5. For each application, we conduct two sets of experiments, running 10 instances for 100 hours. One is to use the default configuration without any selection process; the other one is to use the "trial-and-better" approach, i.e., applying for instances from EC2 until acquiring 10 better-performing instances. Note in the latter case, the other worse-performing instances are terminated immediately after checking their CPUID information.

The speedup and overhead of the "trial-and-better" approach of these two applications, compared to the default setting, is listed in Table 6. The speedup is an accumulated throughput of the selected 10 better-performing instances compared against the randomly returned 10 instances. The overhead is the extra running hours required to acquire the 10 better-performing instances divided by the useful running hours. Our "trial-and-better" approach is straightforward to apply with. We simply check the CPUID information of the instance, keep the instances hosted by E5645 processor running and discard the other ones. This selecting process takes no more than 4 minutes. In a more realistic scenario, to select better-performing instances, cloud users can always utilize certain fast micro-benchmark tool rather than running the real application itself. Because EC2 charges by hour, the selection cost is rounded up to one full hour for those instances. It is worth noting that the overhead percentage decreases as the running time of the application increases because the overhead is amortized. Table 6 clearly demonstrates the benefit of the "trial-and-better" approach. Despite the extra overhead, the throughput boosted well justifies the extra cost of the selecting process. Depending on the application, throughput boost from the "trial-and-better" approach ranges from 11.5 percent for in-memory database application to close to 30 percent for web application, which is in line with our measurement in Section 5 and our analysis in Section 6.1. Furthermore, from our measurements, not only the throughput is boosted by the better-performing instances, but also the relative variation is reduced with these instances, which is of high significance to commercial deployments.

## 7 Conclusion and Future Work

We exploited hardware heterogeneity of public clouds in this paper. Amazon EC2 and Rackspace cloud are taken as two representatives for the analysis. Through longitudinal measurements, we found out that hardware heterogeneity is a commonality among the relatively long-lasting cloud providers. The level of heterogeneity, however, varies between different providers. Amazon EC2 uses diversified hardware within the same availability zone, while Rackspace demonstrates heterogeneity only between different regions. Hardware diversity serves as the primary culprit of performance variation. Because of widely diversified hardware, performance variation among EC2 instances is significant, varying from 20 percent for CPU performance to 268 percent for memory. On the other hand, Rackspace cloud utilizes similar hardware; thus, its performance variation is comparatively small, ranging from 15 percent for CPU and 75 percent for disk. Furthermore, CPU acquisition percentage and VM scheduling mechanisms exacerbates performance variation, especially in network-related operations. Finally, cost-saving approaches, game-theoretic analysis, and Nash equilibrium were discussed from cloud user perspective. By utilizing a simple "trial-and-better" approach, EC2 users can achieve up to 30 percent cost saving, which is verified by a real implementation in EC2 platform. We hope our work will spark a spectrum of research efforts from various aspects, for example, building more homogeneous platform from heterogeneous hardware, game-theoretic analysis (among cloud users, between cloud users and cloud providers). In the future, we will investigate the performance impact factors of disk operations in public clouds.

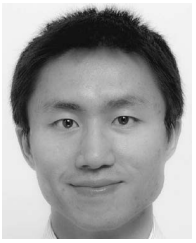
## Acknowledgments

The research work was 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] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Technical Report NIST Special Publication 800-145, The Nat'l Inst. of Standards and Technology (NIST), 2011.
[2] "Amazon EC2," https://aws.amazon.com/ec2/, 2013.
[3] "Rackspace," http://www.rackspace.com/cloud/, 2013.
[4] "Google Compute Engine," https://cloud.google.com/products/compute-engine, 2013.
[5] "Microsoft Azure," http://www.windowsazure.com/en-us/, 2013.
[6] Z. Ou, H. Zhuang, J.K. Nurminen, A. Ylä-Jääski, and P. Hui, "Exploiting Hardware Heterogeneity within the Same Instance Type of Amazon EC2," *Proc. Fourth USENIX Conf. Hot Topics in Cloud Ccomputing (HotCloud '12)*, pp. 1-5, 2012.
[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *Proc. ACM Symp. Operating Systems Principles (SOSP '03)*, pp. 164-177, 2003.
[8] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the Three CPU Schedulers in Xen," *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 35, no. 2, pp. 42-51, 2007.
[9] E. Walker, "Benchmarking Amazon EC2 for High-Performance Scientific Computing," *USENIX; login:*, vol. 33, no. 5, pp. 18-23, 2008.

[10] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman, and N. Wright, "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud," *Proc. IEEE Second Int'l Conf. Cloud Computing Technology and Science (CloudCom '10),* pp. 159-168, 2010.
[11] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen, "Cloud versus In-House Cluster: Evaluating Amazon Cluster Compute Instances for Running MPI Applications," *Proc. State of the Practice Reports (SC '11),* pp. 1-10, 2011.
[12] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," *Proc. 10th ACM SIGCOMM Conf. Internet Measurement (IMC '10),* pp. 1-14, 2010.
[13] A. Li, X. Zong, S. Kandula, X. Yang, and M. Zhang, "Cloudprophet: Towards Application Performance Prediction in Cloud," *Proc. ACM SIGCOMM '11 Conf.,* pp. 426-427, 2011.
[14] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann, "What Are You Paying for? Performance Benchmarking for Infrastructure-as-a-Service Offerings," *Proc. IEEE Int'l Conf. Cloud Computing (Cloud '11),* pp. 484-491, 2011.
[15] G. Wang and T. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," *Proc. IEEE INFOCOM '10,* pp. 1-9, Mar. 2010.
[16] J. Schad, J. Dittrich, and J.-A. Quianée-Ruiz, "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance," *Proc. VLDB Endowment,* vol. 3, pp. 460-471, Sept. 2010.
[17] S.K. Barker and P. Shenoy, "Empirical Evaluation of Latency Sensitive Application Performance in the Cloud," *Proc. First Ann. ACM SIGMM Conf. Multimedia Systems (MMSys '10),* pp. 35-46, 2010.
[18] S. Suneja, E. Baron, E. de Lara, and R. Johnson, "Accelerating the Cloud with Heterogeneous Computing," *Proc. Third USENIX Conf. Hot Topics in Cloud Computing (HotCloud '11),* pp. 1-5, 2011.
[19] G. Lee, B. Chun, and R.H. Katz, "Heterogeneity-Aware Resource Allocation and Scheduling in the Cloud," *Proc. Third USENIX Conf. Hot Topics in Cloud Computing (HotCloud '11),* pp. 1-5, 2011.
[20] S. Yeo and H. Lee, "Using Mathematical Modeling in Provisioning a Heterogeneous Cloud Computing Environment," *Computer,* vol. 44, no. 8, pp. 55-62, Aug. 2011.
[21] A. Samih, R. Wang, C. Maciocco, T.-Y.C. Tai, R. Duan, J. Duan, and Y. Solihin, "Evaluating Dynamics and Bottlenecks of Memory Collaboration in Cluster Systems," *Proc. IEEE/ACM 12th Int'l Symp. Cluster, Cloud and Grid Computing (CCGrid '12),* pp. 107-114, 2012.
[22] B. Farley, V. Varadarajan, K. Bowers, A. Juels, T. Ristenpart, and M. Swift, "More for Your Money: Exploiting Performance Heterogeneity in Public Clouds," *Proc. Third ACM Symp. Cloud Computing (SoCC '12),* pp. 1-14, 2012.
[23] I. Leslie, D. Mcauley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The Design and Implementation of an Operating System to Support Distributed Multimedia Applications," *IEEE J. Selected Areas in Comm.,* vol. 14, no. 7, pp. 1280-1297, Sept. 1996.
[24] "Credit-Based CPU Scheduler," http://wiki.xensource.com/xenwiki/CreditScheduler, 2013.
[25] "UnixBench," http://freecode.com/projects/unixbench, 2013.
[26] "RAMspeed," http://alasir.com/software/ramspeed/, 2013.
[27] "Bonnie++," http://www.coker.com.au/bonnie,++/, 2013.
[28] "Httperf," http://www.hpl.hp.com/research/linux/httperf/, 2013.
[29] "Passmark CPU Benchmarks," http://www.cpubenchmark.net/high, end cpus.html, 2013.
[30] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," *Proc. First ACM Symp. Cloud Computing (SoCC '10),* pp. 143-154, 2010.



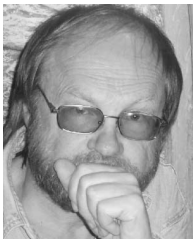
**Zhonghong Ou** received his PhD degree from the University of Oulu, Finland. He received the PhD degree in the field of structured peer-to-peer (P2P) networks. He is a postdoc researcher at the Department of Computer Science and Engineering, Aalto University, Finland since July 2010. From December 2009 to April 2010, he was a visiting scholar at Internet Real-Time (IRT) Lab at Columbia University. From March 2013 through August 2013, he is a visiting scholar at Intel Labs, Portland, United States. He has a wide spectrum of research interests. Recently, he is working on performance evaluation of virtualization and cloud computing platforms, large-scale machine-to-machine communications, energy optimization in cellular networks (3G and LTE). He is a member of the IEEE.

**Hao Zhuang** received the BSc degree in software engineering from Northeastern University, China in 2009, and the MS degree in Erasmus Mundus NordSecMob program specialized in security and mobile computing from Aalto University School of Science and Technology (TKK), Finland and Royal Institute of Technology (KTH), Sweden. He was provided a full scholarship for two years studying in his MS degree from the European Union. He is currently working toward the PhD degree with a topic of distributed system and cloud interoperability in Distributed Information Systems Laboratory (LSIR) at EPFL Lausanne, Switzerland. His major research interests include distributed computing, decentralized cloud computing, cloud interoperability and federation. He is a member of the IEEE.

**Andrey Lukyanenko** received the master's degrees from the University of Petrozavodsk in Russia and the University of Kuopio in Finland. In 2010, he received the PhD degree from the University of Helsinki. His thesis was on protocols for resource sharing in Internet/wireless environments. He worked on problems related to backoff protocols in IEEE802.11, security with host identity protocol (HIP), problems of denial-of-service attacks and reputations in peer-to-peer networks. He stayed in Helsinki to do a postdoctoral fellowship at the Aalto University on information-centric networking, data centers architecture and future Internet design. During his research, he uses game theory, queuing theory, and data analysis methods. He is a member of the IEEE.

**Jukka K. Nurminen** received the MSc degree in 1986 and the PhD degree in 2003 from Helsinki University of Technology. He started as a professor of computer science at Aalto University at the beginning of 2011. He has a strong industry background with almost 25-years experience of software research at Nokia Research Center. His experience ranges from mathematical modeling to expert systems, from network planning tools to solutions for mobile phones, and from R&D project management to tens of patented inventions. His main research interest are energy-efficient computing and communication, mobile peer-to-peer, distributed solutions for mobile devices, web communications. He is a member of the IEEE.

**Pan Hui** received the PhD degree from Computer Laboratory, University of Cambridge. He is currently a faculty member in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology, where he directs the HKUST-DT System and Media Lab. He also serves as a distinguished scientist of Telekom Innovation Laboratories (T-labs) Germany and an adjunct professor of social computing and networking at Aalto University Finland. He has published more than 100 research papers and has several granted and pending European patents. He has founded and chaired several IEEE/ACM conferences/workshops, and served on the technical program committee of numerous international conferences including IEEE Infocom, SECON, MASS, Globecom, WCNC, and WWW. He is a member of the IEEE.



**Vladimir Mazalov** received the PhD degree from the Faculty of Applied Mathematics, Leningrad University in 1979. After that he has mainly worked in research projects funded by the Russian Academy of Sciences, in 1980-1998 in Chita Institute of Natural Resources, East Siberia and, currently, in the Institute of Applied Mathematical Research, Karelian Research Center. He defended PhD thesis in Leningrad State University in 1981 and the second degree of Doctor of Sciences in Leningrad State University in 1991. His research interests are related to game theory and stochastic analysis and applications in behavioral biology, networking and economical systems. He is a research director of the Institute of Applied Mathematical Research, Karelian Research Center, Russian Academy of Sciences, and a professor of the Chair of Probability Theory in Petrozavodsk State University. He is a member of the IEEE.

**Antti Ylä-Jääski** received the PhD degree from ETH Zurich in 1993. He was at Nokia during 1994-2009 in several research and research management positions with focus on future Internet, mobile networks, applications, services and service architectures. He has been a professor for telecommunications software, Department of Computer Science and Engineering, Aalto University since 2004. He has supervised more than 200 master's thesis and 16 doctoral dissertations during his professorship. He has currently four ongoing research projects in the areas of Green ICT, mobile computing, services and service architectures: "Cloud Software," "Internet of Things," "Massive Scale Machine-to-Machine Service," and "Energy-Optimized Mobile Computing." He has published more than 60 international peer-reviewed journal and conference articles. He is a member of the IEEE.