UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JONATHAS GABRIEL DIPP HARB

Performance Evaluation of an Uncheatable Benchmark for Cloud Systems

Trabalho de graduação realizado em convênio de dupla diplomação com a Technische Universität Berlin.

Orientador: Prof. Dr. Marcus Ritt

Porto Alegre
2014

RESUMO ESTENDIDO

1 COMPUTAÇÃO EM NUVEM

A Computação em Nuvem vem ganhando cada vez mais atenção devido aos benefícios econômicos que ela traz por disponibilizar recursos computacionais para uso através da rede. Este paradigma evita a necessidade de comprar e manter hardware e software, uma vez que um determinado consumidor pode alocar e utilizar recursos sob um método de cobrança de “pague conforme o uso”. A Computação em Nuvem pode ser definida como um modelo para habilitar acesso conveniente e conforme a demanda, através da rede, a um ambiente compartilhado de recursos computacionais configuráveis, tal como servidores, aplicações e serviços. Para que este paradigma seja mais bem entendido, alguns conceitos básicos devem ser introduzidos. A “*Nuvem*” refere á um ambiente de tecnologia da informação que contém recursos computacionais que estão disponíveis remotamente, através da rede. Um *provedor* refere à entidade que provê recursos computacionais na nuvem. Um *consumidor* refere àquele que utiliza recursos computacionais na nuvem.

Um dos modelos de serviço mais ofertados na nuvem é a infraestrutura (do inglês infrastructure-as-a-service, IaaS), que provê a consumidores acesso a ambientes completos onde o nível de controle é o maior dentre o de todos os outros serviços. Neste modelo, a infraestrutura corresponde a ambientes virtualizados que são disponibilizados como Máquinas Virtuais (MV). MVs podem ser vistas como partições logicas isoladas de um hardware físico que atuam isoladamente. A criação e manipulação de MVs é responsabilidade do *Hipervisor*. No contexto da Computação em Nuvem, consumidores pagam por MVS que são colocadas à disposição para uso. Apesar de terem total controle sobre suas MVs, consumidores não têm controle algum sobre a estrutura responsável pelo provimento de tais recursos, ficando esta sob total controle dos provedores. Como para os provedores mais recursos significam mais clientes e maior lucro, a motivação ao não cumprimento do que foi acordado pode ser grande. Afirmar que o serviço está sendo provido corretamente pode ser um problema para consumidores. Uma solução para tal problema é avaliar os recursos providos através do uso de benchmarks resistentes à adulteração, que evitam que seus resultados sejam alterados por provedores mal-intencionados. Entretanto, o processo de benchmarking consome recursos e isso significa despesas para o consumidor. No geral, é necessário decidir se o esforço do processo de benchmarking na nuvem vale a pena ou não. Deste fato originou-se a proposta deste trabalho, cujo objetivo é concluir sobre o impacto e desempenho de um benchmark resistente à adulteração em um sistema em nuvem através da elaboração de um experimento.

2 BENCHMARKING NA NUVEM

Em computação, benchmarking é o ato de executar um programa a fim de avaliar o desempenho de um objeto. Um objeto pode ser tanto um computador (ou uma parte específica como o processador) quanto um serviço ou recurso disponibilizado na nuvem. Programas de benchmark têm propriedades que permitem que se conclua seguramente acerca do desempenho dos objetos avaliados. No contexto de Computação em Nuvem, benchmarking ainda é algo novo, principalmente devido ao fato de que abordagens de benchmarking tradicionais requerem total controle e completa informação sobre o objeto em questão. Como dito anteriormente, na nuvem apenas os provedores possuem esse privilégio e pouco é disponibilizado aos consumidores, o que torna a tarefa complicada. Para os fins deste trabalho, o processo de benchmarking na nuvem tem como objetivo a avaliação de componentes que de fato estão sob controle do provedor, como a infraestrutura virtualizada. Devido ao fato de terem o controle da infraestrutura, provedores podem detectar que um programa de benchmark está sendo executado e, ao fim da execução, adulterar o resultado para passar uma imagem falsa do sistema. Portanto, a propriedade de um benchmark de ser resistente à adulteração se torna fundamental.

No contexto do cenário descrito até agora, Falk Köppe desenvolveu em seu trabalho um benchmark resistente à adulteração para sistemas na nuvem, chamado aqui de *POW* [20]. O POW baseia-se em um sistema de desafio/resposta que avalia o desempenho do processador. Nesses tipos de sistemas, desafios são enviados ao objeto testado para que o mesmo calcule a solução e retorne a resposta. Uma propriedade é que o objeto testado não tem outra escolha senão a de calcular o que for necessário, sem atalhos. Para os fins deste trabalho, assumiu-se que de fato o POW benchmark é resistente à adulteração. Apesar de ter-se teoricamente resolvido o problema da confiança na nuvem, a execução de um benchmark consome recursos e avaliar o quão impactante ele pode ser para um sistema em nuvem é, como já mencionado, motivação para este trabalho.

3 DESEMPENHO DE UM BENCHMARK RESISTENTE À ADULTERAÇÃO NA NUVEM

O ponto inicial da elaboração do experimento foi a definição do objetivo, dos cenários, das ferramentas e da metodologia de coleta e análise de dados. O objetivo do experimento é avaliar o impacto do POW benchmark desenvolvido em [20] em um sistema na nuvem bem como avaliar a precisão de seus resultados em diferentes situações. Dois cenários foram definidos, o primeiro correspondendo a uma configuração na nuvem típica, onde não há processo de benchmarking; Neste cenário as aplicações do consumidor executam no ambiente sem interferência de outras aplicações. O segundo cenário compreende a execução da mesma aplicação do consumidor, porém adiciona a execução do benchmark em diferentes situações: uma onde a execução é constante ao longo do tempo e outra onde o benchmark atua como um serviço em plano de fundo, com um fluxo de execução intervalado. As ferramentas escolhidas, necessárias à execução do experimento, foram um hipervisor, responsável pela criação e gerenciamento de máquinas virtuais que simularam a infraestrutura na nuvem, e uma ferramenta que simulou típicas aplicações de usuários, a fim de aplicar cargas de trabalho relevantes. As máquinas virtuais criadas possuíam configurações idênticas, exceto pelo fato de que uma possuía um núcleo de processamento e a outra possuía dois núcleos. A diferença no número de núcleos tem o intuito de observar diferenças nos resultados, uma vez que o POW benchmark foi desenvolvido apenas para máquinas de um núcleo. A ferramenta de simulação de aplicação de usuário escolhida foi a *Phoronix Test Suite.* A Phoronix é uma plataforma de benchmarks que compreende diferentes testes para diferentes objetos, incluindo processador. A escolha de uma plataforma de benchmarks é conveniente, pois com benchmarks é possível de se aplicar cargas de trabalho reais em domínios específicos, tal como o dos processadores.

Com os cenários e as ferramentas definidas, foi especificada a metodologia utilizada no experimento para a coleta e analise de dados. Primeiramente, foi considerada a avaliação apenas em picos de processamento. Logo, a plataforma Phoronix utilizava praticamente toda a capacidade de processamento das MVs para simular os picos. O POW benchmark coletava medidas apenas nos picos de processamento e em intervalos de tempo predeterminados. O intervalo era um parâmetro em Milissegundos que deveria ser respeitado até a coleta da próxima medida. Note que um parâmetro de zero Milissegundo indica uma execução constante, e qualquer valor maior do que zero torna o POW num serviço em plano de fundo. A métrica coletada pelo POW era o tempo necessário para a resolução do desafio. A Phoronix executava um conjunto de testes de processador por dez vezes a fim de prover resultados precisos. A Phoronix foi executada nos dois cenários, a fim de possibilitar a comparação de seus resultados com e sem o impacto da execução do POW. O POW, por sua vez, foi executado apenas no segundo cenário, impactando a plataforma Phoronix, em diferentes situações, onde cada situação corresponde a um intervalo de tempo.

Para avaliar o impacto do POW benchmark no sistema, foi realizada uma comparação dos resultados obtidos pela plataforma Phoronix nos dois cenários. A diferença dos resultados permitiu obter de fato o impacto resultante. Para avaliar a precisão fornecida pelo POW nas diferentes situações, foi utilizado o coeficiente de variação. O coeficiente de variação é uma medida de dispersão que é utilizada para estimar o desvio padrão expresso como porcentagem da média. Uma vantagem do uso deste coeficiente é que permite a comparação de diferentes conjuntos de dados, como os das diferentes máquinas virtuais. É importante relembrar que todo o processo descrito até agora foi realizado para ambas as máquinas virtuais, pois a plataforma Phoronix se beneficia de núcleos extras e, portanto, sua execução se torna mais rápida e por consequência o pico de processamento é menor. Como o POW benchmark apenas coletava medidas nos picos de processamento, não se beneficiando de núcleos extras, é evidente que na máquina virtual de dois núcleos tinha-se um tempo menor para a coleta.

4 ANÁLISE DOS RESULTADOS

O primeiro ponto a ser relatado foi o tempo de execução da plataforma Phoronix. Como o esperado, levou-se 4 horas para a execução completa na máquina virtual de dois núcleos e 7 horas na de núcleo único. Isto significa que o POW benchmark teve significativamente menos tempo para a coleta de suas medidas no primeiro caso. Ao final da série de execuções, foi obtida uma quantidade de dados significativa (cerca de 4 GB) para análise. O primeiro resultado a se destacar é que de fato o POW benchmark produz impactos bem significativos que variam conforme a escolha do intervalo de tempo de espera. O gráfico na figura 5.1 mostra o impacto médio nas duas máquinas virtuais.

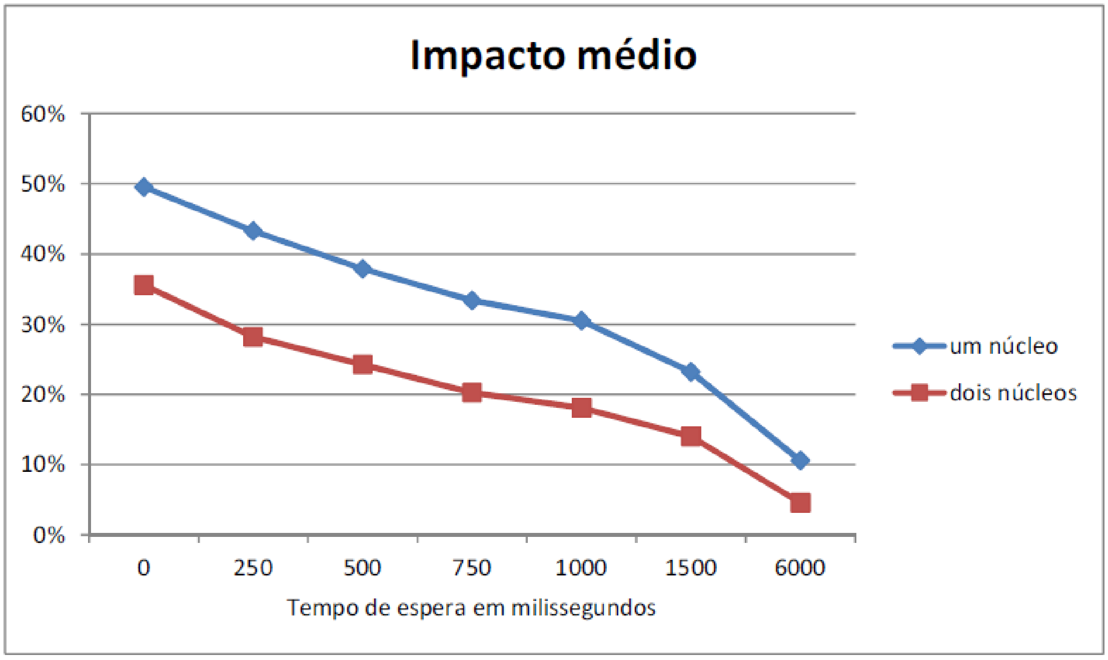


Figura 5.1 – Impacto médio nas duas máquinas virtuais.

Visivelmente o impacto é maior quando o tempo de espera é zero. Conforme o tempo de espera é incrementado, maior se torna o intervalo entre a obtenção de medidas e menor se torna o impacto. É possível observar, com o comportamento da curva, que a tendência é que o impacto seja mínimo, tendendo a zero, em intervalos de tempo muito grandes. A diferença observada comparando-se as duas máquinas virtuais se dá pelo fato de que como a plataforma Phoronix se beneficia do núcleo extra e o POW benchmark não, o impacto resultando acaba sendo menor quando da utilização de núcleos extras.

A segunda parte da análise consistiu em concluir sobre a precisão dos resultados obtidos pelo POW benchmark. É importante ressaltar que o POW benchmark coletou um número diferente de medidas nas diferentes escolhas de intervalos de tempo e o intuito era verificar o comportamento em tais situações. O gráfico na figura 5.2 mostra o coeficiente de variação da precisão para ambas as máquinas virtuais.

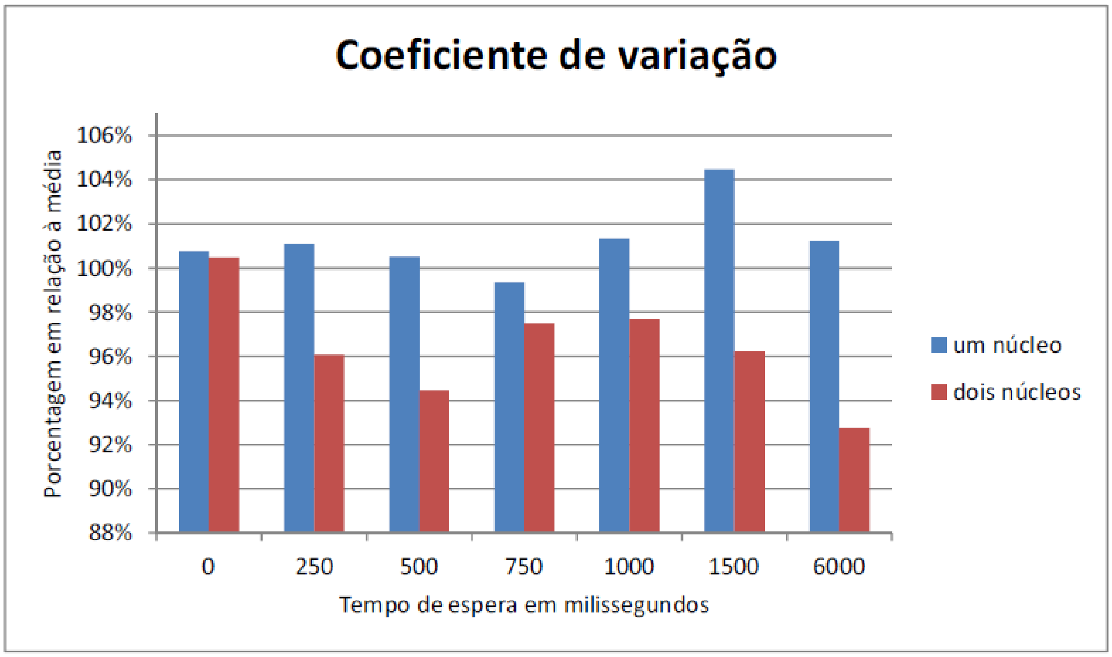


Figura 5.2 – Coeficiente de variação da precisão para as duas máquinas virtuais.

A primeira observação feita é que os resultados acabaram sendo diferentes dos esperados. Foi suposto que o maior número de medidas coletadas em intervalos menores de espera fosse resultar em uma precisão maior. Isto, entretanto, não ocorreu. O que de fato verificou-se foi que não há uma tendência clara de comportamento. Entretanto, é possível de se observar, por exemplo, que o coeficiente de variação pouco muda nas duas máquinas virtuais. Especialmente na máquina de um núcleo os valores são bem próximos, mesmo com intervalos de tempo bem distantes. Uma possível explicação para a falta de uma tendência é um fator de aleatoriedade presente no POW benchmark, melhor explicado em [20]. A pouca diferença nos coeficientes de variação para diferentes situações é explicada pelo fato de que, para todas as situações testadas neste experimento, o número de medidas coletadas pelo POW benchmark superou o número de medidas mínimo estipulado em [20] para obter uma precisão adequada. Este número superior de medidas só foi possível de ser obtido devido ao longo período dos picos de processamento. Um resultado importante disso é que tanto uma escolha por um intervalo menor quanto por um intervalo maior resulta em uma precisão de certa forma constante, porém a diferença no impacto é significativa. O pico de processamento considerado neste experimento foi suficientemente longo para permitir uma escolha de intervalo maior, diminuindo o impacto e mantendo a precisão.

5 CONCLUSÃO

O objetivo deste trabalho foi avaliar o desempenho de um benchmark resistente à adulteração desenvolvido para sistemas em nuvem, o que se acredita ter sido atingido com o experimento proposto e os resultados apresentados. Esta tese guiou o leitor através dos conceitos de Computação em Nuvem, benchmarking, e a relação entre eles. A motivação que originou este trabalho também foi detalhada: a falta de informações acerca do impacto causado por benchmarks resistentes à adulteração na nuvem.

Um experimento foi elaborado a fim de verificar o impacto de um benchmark especifico que atende aos requisitos de ser resistente à adulteração e de ser desenvolvido para atuar na nuvem. A capacidade de prover resultados precisos bem como manter tal precisão em diferentes situações também foi avaliada. O experimento, que simulou um ambiente em nuvem e também cargas de trabalho que representassem aplicações reais de consumidores, consistiu em testar o benchmark atuando tanto continuamente quanto como um serviço em plano de fundo, dividindo o trabalho ao longo do tempo para minimizar os impactos. Observou-se que para este experimento específico uma divisão do trabalho em tempos mais longos não comprometeu a precisão dos resultados, porém diminuiu significativamente o impacto. Por fim, este trabalho mostrou resultados que podem representar a configuração de cenários gerais na nuvem.

Technische Universität Berlin
Fakultät IV
Kommunikations- und Betriebssysteme

Bachelor's Thesis

Performance Evaluation of an Uncheatable Benchmark for Cloud Systems

Author:
Jonathas Gabriel Dipp Harb

Supervisor:
Dr.-Ing. Jörg Schneider

September 9, 2014

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 09. September 2014

____

Unterschrift

Abstract

Cloud Computing is gaining increasing attention because of the economic benefits it brings by making computing resources available over the network for use in a pay-as-you-go billing strategy. This avoids the need of purchasing and maintaining hardware and software. In the cloud, providers may have the incentive to cheat by providing less resources than agreed and charging consumers for the full amount of resources. One solution is to evaluate the provided resources through the use of uncheatable benchmarks, that are resistant to tampering. However, benchmarking is resource consuming and that means expenses to consumers that should then decide whether the effort is worth it or not.

This work presents an experiment that allows one to assess on the performance of an uncheatable benchmark for cloud systems and also gives an idea to consumers on how to proceed and on what to expect. The experiment is comprised by virtual machines that simulate typical cloud environments and also comprised by a tool that simulates a consumer's application. All components are carefully chosen to provide representative results. The findings show the impact caused by the benchmark as well as the variation in the precision of the results produced by such a benchmark.

Zusammenfassung

Cloud Computing gewinnt immer mehr Aufmerksamkeit wegen der wirtschaftlichen Vorteile, die es bringt. Cloud computing macht IT-Ressourcen über das Netzwerk für die Verwendung in eine Pay-as-you-go-Abrechnung verfügbar. Dies vermeidet die Notwendigkeit der Anschaffung und Wartung von Hardware und Software. In der Cloud können die Anbieter den Anreiz haben zu betrügen indem weniger Ressourcen als vereinbart zur Verfügung gestellt, aber vollständig abgerechnet werden. Eine Lösung ist, die bereitgestellten Ressource durch den Einsatz von fälschungssichere Leistungsmessungen zu bewerten. Allerdings verbrauchen Benchmarks Ressourcen, und das bedeutet Kosten für die Verbraucher, die dann entscheiden sollten, ob der Aufwand sich lohnt oder nicht.

Diese Bachelorarbeit stellt ein Experiment vor, das es erlaubt die Leistung eines uncheatable Benchmark für Cloud-Systeme zu bewerten und gibt den Verbrauchern einen Vorstellung davon, wie sie verfahren können und was sie erwartet. Das Experiment wird auf virtuellen Maschinen durchgeführt, die typische Cloud-Umgebungen zu simulieren. Die Werkzeuge, die die Anwendung eines Verbraucher simulieren, werden sorgfältig ausgewählt, um repräsentative Ergebnisse zu liefern. Die Ergebnisse zeigen die Auswirkungen die der Benchmark verursacht sowie seine Präzision.

Contents

Chapter 1

Introduction

Computing has become part of people's routine and may be delivered nowadays in different manners. Among them, computing delivered specifically as a service under the term *Cloud Computing* is gaining increasing attentions. Cloud computing has the potential to transform part of the IT industry by enabling the offering of computing resources over the network and bringing economic benefits. Users no longer need to purchase and maintain hardware and software resources, instead, they can make use of computing resources that are available in a pay-as-you-go billing strategy and pay only for the time they use such resources. In this paradigm, a service provider delivers the agreed amount of resources and at the same time starts to apply the correspondent bill, which depends on the time of use. The contract that stipulates the amount of resources to be provided as well as the obligations is called the Service Level Agreement (SLA). The SLA represents the trust relation between consumers and providers.

However, in cloud computing the possibility of cheating on the SLA causes consumers to want to monitor and evaluate performance of the resources, in order to check if what was agreed is being delivered. In order to achieve that, benchmark programs should be used. More precisely benchmarks that are resistant to tampering are preferred, as the provider may be able to sabotage results. However, benchmarking is resource consuming and in cloud computing it means expenses. This means that consumers may have a hard time deciding whether to benchmark or not the provided resources.

That said, this work wants to assess on the performance of a specific uncheatable benchmark for cloud systems in order to give an idea to consumers on how to proceed and on what to expect. To achieve that, an experiment comprised by two scenarios is proposed. The scenarios can be seen as real cloud configurations and the procedures as well as the tools and choices used to represent such scenarios are carefully chosen to provide representative results. In a nutshell, the experiment consists of running an application on virtual machines that represent the cloud environments. Such application is assumed to simulate a typical consumer's application. The next step is then to evaluate the virtual machines with the chosen uncheatable benchmark. With the experiment this work gathered data that allowed one to conclude on the impact caused by the uncheatable benchmark as well as to conclude on the precision of its results and on the important relation impact-precision.

The first step of this work is to explain the basis regarding cloud computing and its technologies. The SLA and the need for monitoring and evaluating cloud systems is explained and the points of interest for this work are touched in details. This step is described in the chapter *Cloud Computing*.

The second step is focused on computing benchmarking and details its goals as well as characteristics. In addition, this step consists of relating computing benchmarking with cloud computing under the term *cloud benchmarking*, which is explained in details from the point of view of this work. Furthermore, the so-called uncheatable benchmarks are presented and also the short theory behind them. The chapter that comprises this step is named *Benchmarking*.

Afterwards, the experiment itself is described and explained as well as everything that it comprises. This includes the specification of the scenarios as well as the reasons for the chosen tools that together form the experiment. Also in this step the methodology used to collect and analyze the data is detailed. The chapter that comprises all this information is named *Performance of an uncheatable benchmarks for cloud systems*.

Finally, the chapter named *Analysis* presents the findings originated from the data gathered on the experiment. It gives an idea on how consumers can proceed in general cases as well as on what they can expect from cloud benchmarking.

Chapter 2

Cloud Computing

This chapter presents cloud computing, the basis for this work. It aims to introduce the researched literature and also provide the knowledge needed to better understand the context and the motivation of this thesis.

2.1 Cloud Definition and Concepts

Over the years, the term cloud computing has been defined differently in several literatures, the idea of shifting infrastructure to the network has attracted many users to migrate to the cloud. Among some formal definitions proposed in the community, the one given by the US National Institute of Standards and Technology (NIST) includes widely accepted and used terms. The NIST defines cloud computing as "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., servers, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[1]. In addition, the cloud computing model is still composed by three service models, four deployment models, and five characteristics. The service models and the deployment models are to be approached in a dedicated section. The following characteristics are essential part in the definition of cloud computing. Basically, one can see cloud computing as a model comprised by:[1]

- On-demand self-service: consumers can access additional computing resources on demand without necessity of interaction with providers.
- Broad network access: computing resources are available over the Internet for any device to access through standardized mechanisms
- Resource pooling: the computer resources can be hosted anywhere and serve multiple consumers according to a demand model.

- Rapid elasticity: computing resources can be elastically provided, so that the varying consumer demand can be satisfied at any time.

- Measured Service: computing resources used by consumers are monitored and measured, so that the pay-per-use model is possible.

The above definition of cloud computing as well as the characteristics given by the NIST aim to provide a baseline for further discussions regarding this paradigm. In fact, there are many opinions on what cloud computing can actually mean: from renting several virtual servers and loading applications on it to just storing huge amounts of data. As one can notice, a wide range of options could be related to this paradigm. That said, the definition given by the NIST is considered here as covering all the relevant meanings for this work. In addition to that, [2] refers to computing resources in the cloud as "often virtualized resources"; Yet to be approached, cloud computing incorporates virtualization as well as several others concepts that have already been established; By doing so, cloud computing can be seen as a new paradigm just in the way it promotes changes on developing, deploying, maintaining and paying for applications and underlying infrastructure[3]. In this new model of providing computing resources as a service through the Internet, it becomes important to have the knowledge about some basic concepts and cloud computing roles; such knowledge will help in the further understanding of the concepts presented in this work. Hereupon, [4] presents the following concepts:

- A cloud refers to a information technology (IT) environment designed for provisioning computing resources remotely, i.e. enabling computing resources through the Internet.

- A computing resource refers to physical and/or virtual IT entities, such as physical servers and software programs.

- A cloud service is any remotely accessible computing resource through a cloud.

- A cloud service consumer is a role that a software program incorporates while accessing a given cloud service.

In addition to the above concepts, [4] and [5] define the following roles:

- A cloud provider is an entity responsible for providing computing resources in a cloud.

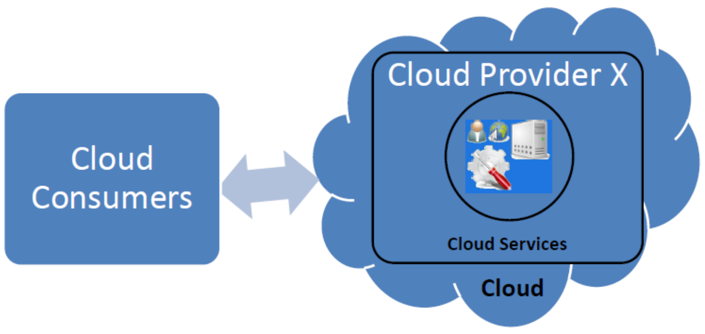


Figure 2.1: An example of a Cloud Interaction

- A cloud consumer is the final user (company and/or humans) that uses the available computing resource provided by a cloud provider through a contract/agreement. This agreement is referred as a cloud service level agreement (SLA), yet to be approached in a dedicated section.

To better assimilate some of the concepts and roles described above, a simple interaction of a cloud system is presented in the figure 2.1.

As mentioned before, cloud computing comprises different services models and different deployment models. The next subsections are dedicated to present such models that might be related to one or more of the above presented concepts and roles.

2.2 Cloud Service Models

The services provided in the cloud are classified into three main models, according to the abstraction level of the offered computing resources: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [6]. In general, the services models of the cloud providers are classified into one of the three main models, although specialized variations have emerged lately (e.g. Storage as a Service, Database as a Service, etc.)[7]. In addition, according to [8], the different levels of abstraction of computing resources can be viewed as an integrated architecture composed by layers, in simpler words, as a layered cloud stack. In this integrated architecture, computing resources that have similar level of abstraction belong to the same layer, furthermore, composability of services is possible. In this case, services of higher layers are composed from services of the underlying layers. One may have a view of the layered cloud stack by checking figure 2.2.

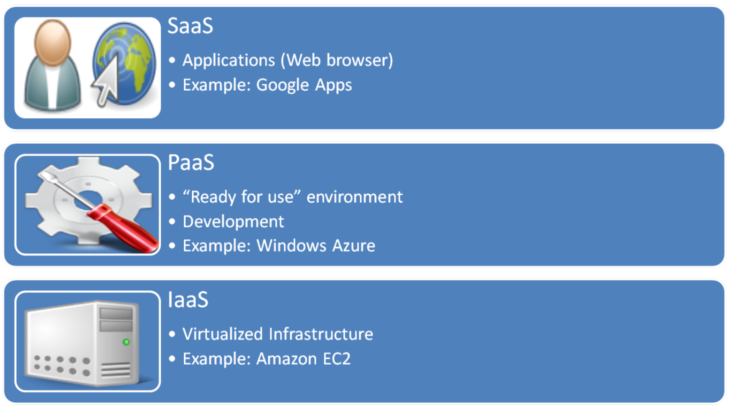


Figure 2.2: The cloud stack

2.2.1 Infrastructure as a Service

Infrastructure as a Service represents the service model that offers virtualized resources such as processing power, storage and networks. These virtualized resources compose the infrastructure and therefore this model is in the bottom of the cloud stack, providing the basis for the other delivery models. By incorporating virtualization to abstract the underlying hardware, cloud providers can manage multiple consumers in a single machine that therefore shares its processing power [9]. As can be noted, virtualization plays an important role (as mentioned in previous sections) and will be approached in a later subsection. In the IaaS model, the infrastructure allows on-demand provisioning of computing resources to the consumers, that is, whenever the consumers need more computing resources, the cloud provider is able to quickly provide such demands. Furthermore, the infrastructure is offered in a way that the consumers are able to deploy and run different kind of softwares (for instance, operating systems) freely, in other words, the consumers have a higher level of control compared to other service models and therefore they become responsible for the environment's usage [1]. The figure 2.3, adopted from the point of view of Microsoft Azure[1], illustrates the different levels of controls in cloud service models. Despite the control over such environment, the consumers do not manage the underlying infrastructure, which could raise questions as for example if the provided service (say, processing power, storage, and so on) corresponds to what has been agreed. According to [9], typical IaaS systems include features such as choice of pre-configured virtual machine and operating system (OS), choice of virtual-machines with specific pre-installed softwares, and ability to increase and decrease comput-

[1]http://azure.microsoft.com/en-us/services/virtual-machines/

ing resources in order to satisfy demands from softwares and applications. For the purposes of this thesis, Iaas is the delivery model that will be taken in account since it provides the whole infrastructure and allows high level of control of such infrastructure, what configures a basic scenario in which other service models could be stacked on top.

2.2.2 Platform as a Service

The Platform as a Service model adds abstraction level to the IaaS by providing a sort of *middleware*, that is, an environment that is "ready for use", including already deployed and configured resources[9]. According to [6], this higher level of abstraction has the goal to make the cloud "easily programmable". Within such easily programmable environment, the consumers can quickly develop and deploy applications without having to worry about the underlying configured resources, in other words, they are not responsible for the environment's usage because such environment already provides everything they need for "programming" and therefore they are just responsible for their own applications. In addition, PaaS usually offers specialized services and interfaces for secure data access, authentication, etc.[6]

2.2.3 Software as a Service

Software as a Service represents the model in which applications running in a cloud are provided to the consumers as a service, often over the web [9]. In this model the consumers can access the applications from several Internet-connected devices. The abstraction here is at its highest if compared to the other two models and the consumers might have control just to some specific application settings. By using SaaS, one can avoid the concerns involving developing and testing by offering applications in the cloud, while the final users avoid the concerns involving maintenance [6]. In his work, [10] makes it clear how abstract the SaaS is for the consumer by stating that "the service provider handles all of the infrastructure, all of the application logic, all deployments, and everything pertaining to the delivery of the product or service".

2.3 Cloud Deployment Models

Not just cloud computing services are divided into different service models but they are also classified into four distinguished deployment models. Differently from the service models that are based on the abstraction of the of-

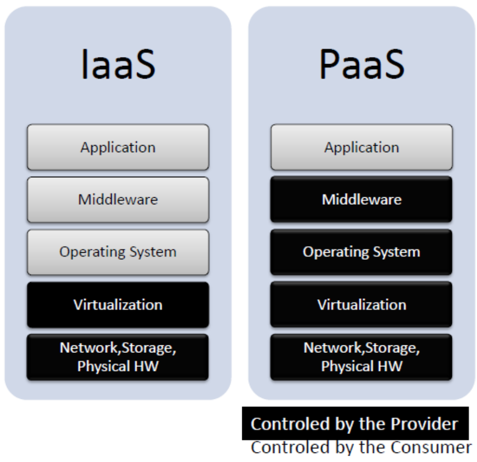


Figure 2.3: Levels of control in the Service Models, adopted from Microsoft Azure

fered computing resource, the deployment models represent the specific type of cloud environment in which the computing resources are offered. Such environment is classified in respect to factors such as ownership and access rights. That said, they are then classified into: public clouds, private clouds, hybrid clouds and community clouds [1].

Public clouds are computing resources that are open to any consumer. In this type of clouds, the cloud providers have the ownership over the computing resources in a degree that depends on the service model in question. As a consequence of such ownership, they are able to apply their own charging model. To the consumers it is possible to make use of such computing resources with respect to the provider's privacy and charging model. [10] As the providers have the ownership over the provided computing resources, the consumers have no option but rely on the providers to meet the SLA, this could be seen as one major drawback of public clouds. For the purposes of this thesis, public clouds are the ones that will be taken into account, since there is a strong relation between consumers and providers and such relation highly depends on SLAs. The fact that providers could deliver something different from what was agreed just to be capable of having more customer and therefore more profit makes SLAs crucial component of public clouds.

Private clouds are the types that are operated and owned by a single organization for internal use . In this case, companies can make use of the cloud computing technology to bring benefits such as a centralized environment of computing resources. According to [10], maximize and optimize existing resources, avoid security concerns and data transfer concerns constitute aspects that could lead to the use of private clouds.

Hybrid clouds are combination of both public and private clouds.The use of hybrid clouds is useful whenever an owner of a private cloud wants to expand its computing resources, like creating an extension of the existing ones. In this case, the clouds remain separated but are delimited by standardized technology. [1]

Community cloud refers to the type of cloud that is constructed and shared by several organizations. In this case, these organizations use the same built cloud infrastructure and therefore they are delimited by the same policy [10]. According to [1], Community clouds can be owned by one or more of the organizations in question, by a third party or even by a combination of them.

2.4 Virtualization and Cloud Computing

Virtualization is one key technology that enables cloud computing. According to [11], "Virtualization refers to the act of creating a virtual version of something, including but not limited to a virtual computer hardware platform, operating system, storage device, or computer network resources". Basically, this technology abstracts the computing resources and allows multiple systems to operate independently from each other into a single physical machine; Moreover, the abstraction above-mentioned hides the complexity of physical resources management and therefore allows scalability (which is essential part of cloud computing). One result could be that a given software stack (or specific operating system) could be deployed, undeployed and redeployed as many times as needed since it would not be tied to any physical machine. As one may notice, the concepts involving virtualization seem the very same concepts and ideas behind cloud computing, which reinforces virtualization's importance. In a very short description relating both, [12] states that virtualization is responsible for the abstraction of the computing resources while cloud computing is responsible for determining how such virtualized resources are allocated, delivered, and presented as service. Said that, one could now see cloud computing as a model that deals with the resulting hardware's manipulation promoted by virtualization.

Furthermore, the notion of "hardware's manipulation" leads to the concept of virtual machine (VM). According to [3], Virtual machines have become the standard unit of deployment when it comes to virtualization. A virtual machine can be seen as an isolated logical partition of a physical hardware, such virtual machine does not have knowledge of any kind of activity regarding the other VMs. Moreover, the hardware's manipulation that results on the creation of VMs is done by the hypervisor, also called vir-

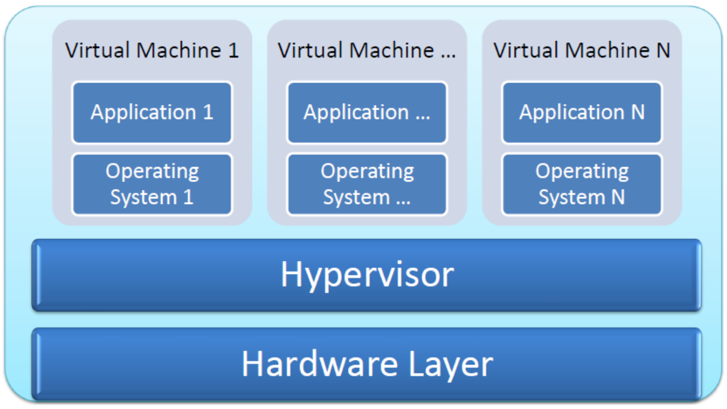


Figure 2.4: A typical Virtualization scenario

tual machine monitor (VMM) [13][14]. A typical structure that represents a virtualization scenario can be seen in figure 2.4, adopted from [13].

As can be noted, in the typical virtualization structure presented above the hypervisor controls directly the hardware and manage the access of several VMs to the resources. In the work of [13], this structure is called hardware-layer virtualization and allows more isolation and performance, this notion is also called in many literatures as full virtualization. Still according to [13], there are more virtualization approaches and among them is included paravirtualization, in which the VMs are aware of the virtualized environment; Moreover, they have to be modified in order to operate in such environment. As a result of paravirtualization the hypervisor becomes simple and the performance achievable by the VMs is higher, as if the environment were not virtualized in fact.

2.5 Cloud Service Level Agreements

In words of [15], a service level agreement is defined as a "negotiated contract between the customer and the service provider". SLAs aim at specifying exactly what kind of services are to be delivered to the consumer, leaving out of the question how such services are to be in fact provided. In his work, [16] states that SLAs are critical in cloud environments because the cloud providers have " responsibilities on behalf of the consumers". Such responsibilities come from the fact that in cloud environments the consumers have limited control and management rights over the computing resources. This delimitation requires contracts (SLAs) as a form of assurance that the provided service will be secure and reliable over the time and that the requirements will be fulfilled.

According to the work of [15], SLAs specifically aim at achieving determined objectives such as determining the customer's needs and providing an understandable framework that states services and costs as well as clarifies the responsibilities of both consumer and provider. As a way to achieve that, SLAs record terms of understanding in which provider and consumer are aware of. These terms are also referred as SLA's components and cover areas such as services, time, guarantees, credit and warranties.

Determining and recording the terms to build an SLA constitute important steps towards the SLA management process, however, these steps are related to the contract negotiation itself. As one may realize, another important step is to provide a mechanism in which SLAs could be monitored in real time, aiming at avoiding that providers cheat on the terms and also to measure the quality of the delivered service.[16] In order to monitors SLAs a set of metrics should be define, examples of typical metrics include CPU speed, memory and storage size.

2.5.1 The need for monitoring

As already mentioned, SLAs are of critical importance in cloud environments and should somehow be monitored as way of guarantee that they are being fulfilled. In order to so, a set of metrics should be defined, examples of typical metrics include CPU speed, memory and storage size. One example of its importance is the billing model, which might be in some cases really attached to the SLA's metrics and terms. From the providers perspective, as cloud services are mainly profit-based, more available resources means more potential consumers and therefore higher profit. That said, providers may have the incentive to violate the SLA by supplying consumers with less resources than agreed.

To solve this problem of SLA cheating, the consumers could then assume the monitoring process as their responsibilities, however, two factors can make such process really difficult: the first one is the lack of control over the underlying infrastructure, which limits consumers in running any kind of monitoring tool on the cloud; The second factor are the resources usage themselves, since adding extra processes for monitoring purposes could consume the very same resources that the user has paid for [17]. Furthermore, as the providers have the control over the underlying infrastructure, they could also identify such monitoring process and tamper the results. Tampered results would give the wrong notion to the consumer that the service is successfully being delivered as agreed. Another way of cheating on the SLA would be allocating more resources specifically when a given monitoring process is ongoing.

In summary, not only SLA should be monitored but this process might be consumer's responsibility. Moreover, providers might be able to cheat on SLA and tamper results of monitoring process. Bringing it all together, consumers have to find a way of applying some kind of uncheatable tool to assess the cloud environment and therefore if the SLA is being fulfilled or not.

2.6 Cloud Providers

Currently several offerings of different cloud service models are available for cloud consumers for use, examples include Amazon's Elastic Computing Cloud (EC2)[2], GoGRID[3], Microsoft's Azure platform[4] and Google's App Engine[5]. The service offered by these providers comprise IaaS, PaaS and SaaS. Also, despite the fact that nowadays these providers also make it possible to set up a private cloud, they can be mainly classified as public solutions since their services are "made available in a pay-as-you-go manner to the public" [18]. Furthermore, each provider may be related to more than one service model and for the purposes of this thesis just the IaaS options offered by such providers will be taken in account.

Amazon EC2 is perhaps the most famous cloud provider. In the work of [19], this service as well as GoGRID are considered as examples of "server-hosting" offers. In fact, both services allow the consumers to rent virtual machines and to freely configure, run and store whatever they want on them. These virtual machines are usually referred as servers and are comprised of "compute capacity", the amount of compute capacity used over the time basically defines the price of the service; In addition, the consumers can quickly scale the compute capacity to their demands.

Although both Amazon EC2 and GoGRID seem to offer really similar services (renting virtual machines), much difference can be found between them and all other providers when comparing the prices related to compute capacity as well as SLAs terms and additional features [18]. For example, Amazon EC2 works with "on-demand instance prices", where the instances are categorized into general purpose, compute optimization and memory optimization, among others. Within each category there are several choices of processing power, memory and storage capacity. Microsoft's Azure plat-

[2]http://aws.amazon.com/ec2/
[3]http://www.gogrid.com/
[4]http://azure.microsoft.com/en-us/
[5]https://cloud.google.com/products/app-engine/

form provides its “virtual machines” in similar categories: general purpose, compute intensive and memory intensive. Google’s App Engine also sticks to the same principle and provides “machines” classified into standard, high memory and high CPU. Differently, GoGRID offers “server pricing” that are classified from X-small until XXX-Large, each category differs from another in processing power, memory and storage capacity. In addition to the above-mentioned services, each provider offers different operating systems and additional features to complement the infrastructure, so the final scenario could be a mix of different service models.

Regarding SLAs, the above-mentioned providers share some common terms as for example the guaranteed service uptime percentage of 99.9 % (GoGRID states 100%), if the service uptime is below expected, customers can request credits. Furthermore, GoGRID seems to be the only provider that really offers terms for performance of its service, the other providers either do not mention performance at all or slightly refer to it along with a long list of exclusions. By all means, one could infer that the providers focus on protect themselves by creating a wide range of exclusion terms, which delimits the scope of responsibilities in cases of failure; In addition to that, verifying if the provider meets the SLA’s terms is in most cases responsibility of the consumer. Nevertheless some providers offer their own monitoring tools or monitoring team[6][7][8][9].

2.6.1 The need for evaluation

The fact that some providers have similar approaches to offer their services does not mean that a comparison is possible. When looking for the most appropriate service, one may find a hard time equating virtual machines and their related prices; Moreover, each provider has its own SLA with peculiarities that might not be clear to consumers. In words of [19], “today’s cloud services differ among others by cost, performance, (...), SLA and programming language. System architects and developers are confronted with this variety of services and trade-offs.” Therefore, this scenario of “non-standardization” raises questions on how one could really verify which service fits better the needs. One solution for such questions would be benchmarking targeted services in order to evaluate and assess how well they fit such needs, therefore helping consumers to choose the right service. Benchmarking is a well-known approach to evaluate and compare performance of systems. That said, bench-

[6]http://aws.amazon.com/ec2/sla/
[7]http://azure.microsoft.com/en-us/support/legal/sla/
[8]https://developers.google.com/appengine/sla
[9]http://www.gogrid.com/legal/service-level-agreement-sla

marking could also assess whether the providers are really fulfilling the SLA's terms regarding resources and performance, working as a sort of monitoring mechanism as described in the previous section.

As one may notice, evaluating cloud services is a pertinent subject. Although benchmarking seems a simple solution that would solve the main problems, customers would have to trust that the benchmark's results are not, again, tampered by the provider. For example, Amazon does not offer any benchmarking tool but instead it offers its own monitoring tool called CloudWatch[10]. With such tool consumers can "collect and track metrics", but then the consumer must trust on "Amazon monitoring Amazon's services" and therefore consumers face the same problems described in section 2.5.1. In addition to this problem, benchmarking cloud services to collect and track metrics would require such services to be benchmarked with certain frequency. However, in relation to what was stated in 2.5.1, adding benchmarks as monitoring tools can consume a big parcel of the computing resources that the consumer is paying for. As a consequence, the two new questions that summarize the main problems faced are how benchmarking can avoid result's tampering and specially how would benchmarking impact the overall performance of the computing resources. Both questions are to be approached in the next section but specifically the second question is the one that this work aims to approach. The question regarding result's tampering is addressed in [20] and discussed in the next chapter.

This chapter provided the basis for cloud computing. In addition, the need for monitoring cloud services with respect to SLAs was elucidated as well as the need for evaluating the service provided. Also, benchmarking was cited as a possible solution for such needs.

[10]http://aws.amazon.com/cloudwatch/

Chapter 3

Benchmarking

This chapter aims to clarify the notion of benchmarking and point out the direction of this work. In order to do so, the basics about benchmarking and the relation with cloud computing are explained; Furthermore, a specific section is dedicated to explain the so-called *uncheatable benchmarks*.

3.1 Definition and Goals

The term *benchmarking* applies to many areas. In computing, a benchmark "is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object" [21]. Such object can be seen as computer hardware, a software, a database, or conveniently as a *system under test*. In the work of [22], a system under test is a collection of components that are needed in order to run a benchmark, it includes not only the component of interest but also extra ones. For instance, one given benchmark program can run against a computer CPU with the goal of stressing it and afterwards assessing how many floating point operations it can handle, in this case the component of interest in question is the CPU and it composes the system under test together with any other component that allowed such benchmark to run. Furthermore, not just benchmark programs assess performance but they aim at comparing the results against similar systems. Nowadays, given the variety of systems that can be tested, the varying workloads and the constant growing computing industry, a wide range of benchmarking tools are available aiming at comparing/evaluating systems. For instance, SPEC CPU2006* benchmark assesses systems regarding CPU, memory subsystem and compiler, SPEC virt$^{\text{TM}}$sc2013* assesses datacenter servers running in virtualized platforms regarding all components: hardware, virtualization platform, guest operating system and applications). The two above-mentioned benchmarks are well-known tools to compare systems simply regarding those specific aspects, say there are still extra components in the systems, then they should be evaluated with different tools.

Besides having the view of what benchmarking means in computing, it is important to have a view on how benchmarks actually work. According to [22], benchmarks create a "representative scenario for a given domain". Basically, they should define a set of rules on how such scenario should be created and on how the results should be obtained (through a choice of performance evaluation criteria and evaluation metrics). In a typical scenario-like assessment, several configurations and parameters can be settled in different forms to provide a comparison of scenarios and afterwards identify the best settings for a given system. Moreover, as the process of creating scenarios requires the participation of those extra components that compose the system under test, the benchmark should be able to isolate information from the component of interest from the extra components [22]. In a nutshell, benchmarks can be considered useful only if they are able to create a proper scenario that represents a system's expected behavior.

In order to reach the goal of assessing performance and comparing similar systems in a manner considered useful and satisfactory, not just traditional benchmarks should create a representative scenario but they should include desirable characteristics, some of them presented in [23] and [22]:

- Relevant – The benchmark should reflect the typical operation in the purposed problem domain. In addition, it should comprise meaningful and well defined metrics to bring important information about the problem domain.
- Repeatable – There should be confidence that the benchmark can run more then one time and deliver the same (at least very similar) results.
- Economical – The costs generated by running a benchmark should be affordable. In this sense, it might be of high importance for specific situations to conclude on the performance of a given benchmark to determine whether it worth being utilized or not.
- Secure – There should be confidence that the benchmark's result is not tampered, i.e., that the results in fact represent the system.

Still according to [23], there are more key characteristics that qualify a benchmark as "good" in fact. However, addressing extra characteristics such as relevant metrics and portability might run out of the scope of this work. Therefore, the characteristics mentioned in this section are the ones that seem to cover better the needs for understanding the next sections. Specially the last two mentioned characteristics are the most relevant ones. Although different benchmarks may evaluate different systems in multiple ways regarding several aspects, they should include at least those characteristics so that one has the possibility to satisfactorily compare/test given systems in an affordable way.

3.2 Cloud Benchmarking

The term *Cloud Benchmarking* is quite complicated to be defined in a way that it leaves no remaining questions. However, the purpose of benchmarks were elucidated in 3.1 as well as the way they work. Based on that, the definition adopted in this work is the one given in [22], in which one can see cloud benchmarking simply as a benchmark "in which the system under test contains a Cloud service as component of interest". Unfortunately, the idea of cloud benchmarking is not as simple as the definition itself. The first point to be considered is that the service models described in 2.2 delimit the types of benchmarking that can be done in a cloud environment. There is no sense (and it may be not possible) in benchmarking CPU or Memory in a PaaS or SaaS environment, while benchmarking CPU or memory is more related to IaaS, PaaS and SaaS are more associated to benchmarking focused on applications. However, IaaS consumers might also want to specifically benchmark their applications and therefore there is a variety of combinations of components of interest that can be benchmarked in cloud environments with respect to the service model's restrictions.

Another crucial point, and perhaps the most challenging one, is that traditional benchmarks usually rely on known hardware and software configuration [24].In other words, they have a knowledge of the system configuration which allows the key characteristics presented in the previous section to be more easily reached. Back to the figure 2.3, one can note that in cloud environments, due to the varying level of control that the user has from model to model, the knowledge of the underlying infrastructure is compromised. Independent of the service model there are precious information being hidden, which makes the task of benchmarking really complicated. Additionally, as the services are provided on-demand, resources can be allocated and unallocated at anytime and therefore there might be a constant system configuration changing. The work presented in [22] approaches this lack of knowledge of the underlying infrastructure when arguing about system under tests in cloud environments. In such work, it is said that a common benchmark requirement is to "lay open all properties of the involved system under test components". As there is limited control over cloud environments, one is dependent on which properties are exposed by the cloud providers.

As one may notice, benchmarking the cloud is not as simple as benchmarking normal systems and there exists many points of view on how it should be done. Based on what was presented so far regarding the varying level of control, lack of infrastructure's knowledge, the wide range of components of interest and also based on the ideas presented in [20], it is possible to summarize two ways/motivations of benchmarking a given cloud environment to the point of view of this work, as follow:

- First case: benchmarking a component that is in fact under control of the cloud provider. In this case, the goal is to verify the environment provided to the consumer in order to assess weather it really offers what the consumer is paying for. Within this possibility one can benchmark for instance the performance of virtual machines, memory and CPU power. Benchmarking the environment in which the consumer's solutions is deployed on could spot possible cheating on resources or poor resource administration. Note that generally in this case the consumer is using IaaS. In fact, this way of benchmarking a cloud environment is the most relevant because strongly deals with the trust relation between consumers and providers through SLAs. Therefore, this is the way of cloud benchmarking that this work will consider from now on when stating about cloud benchmarking.

- Second case: benchmarking a component that is under control of the cloud consumer. In this case, the goal is to verify the behavior of the consumer's deployed solution in regard to several aspects such as application's communication and configuration. In this way of benchmarking the components are directly affected by the consumer and therefore the provider plays a secondary role (if any at all, since it is assumed no infrastructure interference). As a consequence, there is no such strong relation between consumers and providers as there is in the first case. If consumers assume that any bad behavior of their solution is due to poor infrastructure provision, then they fall on the first case.

As one may notice, in benchmarking either in one way or in another, there is still the need to deal with the specific features of cloud computing that require an approach other than the one used for traditional benchmarking. Taking in account what was elucidated about benchmarking in 3.1, it is possible to identify and summarize two major cloud benchmarking challenges that are relevant for this work. More precisely, some of the desirable characteristics of traditional benchmark programs are strongly challenged by some of cloud computing features, as follow:

- Challenge of being Economical - In cloud environments the consumer pays for computing resources and part of such resources should be allocated specifically for benchmarking. Therefore, complex and long-time run benchmarks would consume more resources for longer time and this implies in more costs for the cloud consumers. In [25], this challenge is refereed as Experiment compression: "Long setup times, (...) , and/or long periods of continuous evaluation add the disadvantage in clouds of greatly and visibly increasing the cost of benchmarking". On the one hand, it is obvious that a new approach to avoid high resources consuming and long time running benchmarks should come up. On the other hand, assessing how current benchmarking approaches with its complexities affect the system to conclude about the economical factor is important subject of work. Therefore, this work is focused on this point, it aims to partially conclude on the impact of benchmarking a cloud environment. In more details, it aims to conclude on the performance of a given benchmark that specifically has the characteristic of being secure. As already mentioned, the benchmarking approach considered in this work is the one that aims to verify the provided environment. In this sense, being secure is important to guarantee that the results really reflect the system and because if a benchmark can have its results tampered than the evaluation of its performance might be biased and might not reflect what a secure benchmark would look like in terms of performance.

- Challenge of being Secure – As cloud providers may have the motivation to tamper results to either improve profit or create a fake image of quality, security on benchmarking raises as an important issue since the providers might have not just the motivation but also the tools to cheat. Again, they control the infrastructure and may hide crucial information (not exposing all properties for benchmarking), making the task of tampering results easier and the task of creating secure benchmarks harder. This problem is the focus of the work of [20]. In such work, a benchmark for verifying CPU performance in cloud environments is developed with the partial property of being secure. As yet to be approached, the results of [20] constitute part of this work.

It is clear that cloud benchmarking is an open issue. The cloud computing paradigm claims for new approaches other than traditional benchmarking.

In this section it was possible to show some of the reasons for these claims; Additionally, two specific challenges that relates to this work were identified. The first one is the focus of this thesis; The second one was the focus of the work presented by [20], in which this thesis aims to contribute. The next section clarifies uncheatable benchmarks, a notion that comes from the traditional benchmarking theory and can be used in solving the problem of computing benchmarking cheating.

3.3 Uncheatable benchmarks

There is no formal definition for the term uncheatable benchmarks. Nevertheless, for this work uncheatable benchmarks are considered as benchmark programs that can evaluate systems without being cheated, therefore producing a reliable result in the end, i.e, a result that really represents the system. The background regarding uncheatable benchmarks points to the work of [26], where this kind of benchmarks is described as "resistant to tampering and hence more trustworthy". In such work, it is stated something that was observed during the research part of this work: much effort has been done in order to create highly representative benchmarks, with relevant metrics and workloads that can simulate a systems closely to what it is; However, there is less effort than the adequate in answering questions such as if the benchmark's results are tampered or not. Still according to [26], uncheatable is a term seen as a property that a benchmark can satisfy; In addition, another important presented property is that it must be drastically easier to prepare and check the results than running the benchmark. In this case, there is an advantage for the one benchmarking over the one being benchmarked. To create benchmarks that supposedly satisfy such properties, [26] uses modern cryptography and complexity theory. Its proposed benchmarks are The Power Benchmark, which is based on the "power function modulo a composite number" (prime numbers), and The One-way Benchmark, which is based on one-way functions chosen by the one benchmarking. Details of both proposals can be found in the work aforementioned.

As already mentioned, not much work has been done which relates to uncheatable benchmarks. In cloud computing the situation would not be different. As to the knowledge of this work, ways of benchmarking the cloud through use of uncheatable benchmarks have received little to none effort, while new approaches for identifying relevant metrics and evaluation criteria that lead to a proper cloud benchmarking have received significantly much more effort. Nevertheless, there are works that focus specifically on cloud cheating - the possibility of cloud providers to cheat on resources. Such works deal with the problem more as a violation/cheating on the SLA and they aim at presenting approaches to detect if any cheating if being done. As one may realize, these works have something in common with the goal of benchmarking the cloud in a secure manner: they present approaches that aim to be secure. The difference is that the techniques/proposals to detect SLA cheating are not really classified as benchmarks because they do not fulfill specific benchmark's requirements, goals and results. The work of [17] proposes a SLA verification framework that includes a third party auditor. It presents a testing algorithm that is able to detect memory size violation on VMs in respect to a given SLA. In addition, it shows experiments that demonstrate cheating detection and also avoids that a provider could hide the SLA violation. In [27], the focus is on detecting CPU speed cheating. In order to achieve that, the work proposes a lightweight "stealthy test algorithm" for video batch processing applications that can detect CPU cheating with low computation overhead. Also, [28] proposes a user-based CPU verification scheme that detects cloud cheating on CPU resources. The scheme is based on task execution time comparison, where a given task with known execution time is given to a cloud service to execute and then the differences are compared.

However, the works aforementioned are not really in the context of benchmarks because they do not assess about the performance of any object . That said, to the knowledge of this work, perhaps the only approach that focuses on benchmarking cloud environments in the sense it tries to satisfy the property of being uncheatable is the work of Falk [20]. In his approach, cloud consumers cannot verify whether providers really deliver what was agreed or not and therefore there is the possibility of benchmarking the service. However, the control over the infrastructure gives the possibility to the provider to tamper benchmark's results. Therefore, [20] identifies four attack vectors for result tampering and implements proof-of-work functions in a prototype benchmark that can disable three of them. In a nutshell, "Proof-of-work functions are challenge response systems, where it is simple to generate a challenge and verify the result while solving the challenge is compute intensive". As one may notice, this approach satisfies the previously mentioned property presented in [26] in the sense that the one benchmarking has the advantage over the one being benchmarked. By disabling most of the possibilities of tampering results, this approach presents itself as partially uncheatable. Therefore, for the purposes of this thesis, it will be assumed that the benchmark suite implemented by [20] constitutes a uncheatable benchmark for cloud environments.

As mentioned in 3.2, assessing how benchmarks affect cloud systems is important subject of research. As in the cloud higher usage of computing resources means more expenses to consumers, benchmarks that consume a lot of resources lead to high costs. As to the knowledge of this work, no specific work was done to assess about the performance and consequent impacts of uncheatable benchmarks in the proposed scope of cloud computing. Also, the approaches aforementioned either do not present enough information to let one conclude about performance. The work of [27] claims to detect CPU cheating with low overhead when stating that "the testing tasks are part of the original computation tasks, which need to be run in the cloud anyway". However, neither such claim is clarified in details and neither the approach itself aims to benchmark. Therefore, the next chapters describe the contribution of this thesis regarding this lack of knowledge on the performance of (uncheatable) benchmarks for cloud environments.

3.4 Summary

Up to this point, it was given the necessary background to understand the concepts involving this work. The basis of cloud computing and benchmarking were explained and the relevant points for this thesis were elucidated. Furthermore, the way this work approaches the relation between these two areas was explained in details. In a nutshell, Infrastructure-as-a-Service, with help of virtualization, is the environment that might raise trust questions between providers and consumers regarding the fulfillment or not of the established SLAs. In respect to that, benchmarking the provided environment becomes important and even more important is to guarantee that the results represent the truth and are not tampered. As a way to cope with that, uncheatable benchmarks and their insertion on cloud systems were presented.

Unfortunately, benchmarking the cloud implies in extra use of computing resources and as a consequence it generates extra costs to consumers. Therefore, evaluating the performance of uncheatable benchmarks for cloud environments and their impacts on the system becomes as important as guaranteeing that results are not tampered. Therefore, through an experiment, this work aims to assess the performance and impact of an uncheatable benchmark for cloud systems. The experiment description is explained in the next chapter in details and comprises the goals, the requirements to accomplish them and the methodology applied.

Chapter 4

Performance of an uncheatable benchmark for cloud systems

As already stated, the general goal of this thesis is to assess on the performance of an uncheatable benchmark designed for cloud systems. In order to do that, an experiment was done. Such experiment is based on the impact evaluation of a given uncheatable benchmarks on a simulated cloud system. That said, the next sections are dedicated to give detailed information about the experiment that composes this work in order to clarify the aims as well as the procedure used to achieve the goal.

4.1 The scenarios

The starting point for the practical part of this work is to establish the basis for the experiment, that is, the scenarios that correspond to real cloud configurations in which it is desirable to perform the experiment. Since all requirements and choices are dependent on the scenarios, starting by defining and explaining each scenario clarifies the next sections and helps one to firstly understand the main idea on what the experiment is all about before introducing specific details. That said, two general scenarios were identified. Their difference is given exactly by the addition of the necessity of benchmarking the target system and not by the difference on solutions, infrastructure offerings or by any other factor (as stated in previous chapters, Infrastructure-as-a-Service is the delivery model relevant for this work). This allows to focus only on the interesting point which is the uncheatable benchmark. Furthermore, as it is desirable to collect correct data to provide meaningful results, both scenarios must ensure that no cheating is being done and therefore the established amount of resources are correctly delivered. This is achieved by the fact that everything related to the scenario is fully controlled and managed by the author of this work. For simplicity, the scenarios are named scenario *one* and scenario *two* and are described as follow:

- Scenario *one*: Corresponds to a typical cloud configuration where benchmarking has little to no importance. Here, the cloud consumer has a given solution deployed on a given cloud environment. Their relation is based on the SLA and the consumer trusts the provider. As a consequence of the trust relation, there is no need of benchmarking the infrastructure delivered by the provider. This scenario is crucial because the results produced here are compared to the ones from the scenario *two* in order to generate the final results, yet to be described. As one may notice, the basic activity on this scenario is limited to the execution of the consumer's solution. In addition, the performance evaluation of such solution on the cloud environment is the contribution of this scenario. Since it is assumed that the performance here is free of interferences (either by other applications or by benchmark programs), its evaluation corresponds to the real performance that the consumer can expect on the agreed infrastructure. Furthermore, it is desirable that the infrastructure does not provide any extra resources other than the required by the consumer's solution. That is, such solution must allocate (at least for the processing peaks) all or close to all available resources in its execution. This situation of full allocation is relevant for this work and will be better understood in the description of the scenario *two*. In summary, the scenario *one* represents the basic cloud configuration and it is responsible for providing the performance evaluation of a given solution on the cloud environment with respect to the restriction that either the solution must basically allocate all available resources or the evaluation should be done on processing peaks. As to be described later, the choice of the consumer's solution (application) is heavily based on this restriction.

- Scenario *two*: Corresponds to a cloud configuration where benchmarking plays an important role. In contrast to the scenario *one*, here the consumer does not really trust the provider and wants to assess the delivered infrastructure. Therefore, there is the addition of the uncheatable benchmark tool that is used by the consumer to assess the cloud. Basically, this scenario has exactly the same characteristics as described in the first one plus the extra that here the benchmark tool impacts the performance of the system and therefore the consumer's application. As mentioned before, the restriction that all or close to all available resources are allocated by the consumer's application should be respected. This allows to evaluate how such benchmark tool can really impact the system. If, for instance, such restriction would not exist, one would be able to assess about the impact of such benchmark on a system that could be in idle state and not really being used by the application, providing incorrect results. That said, the contribution of this scenario is to provide the performance evaluation of the consumer's solution that is being affected by the run of the uncheatable benchmark. In the end, one is able to compare how the uncheatable benchmark affects the application by comparing the data collected here with the data collected in the scenario *one*. Furthermore, this scenario provides information on how the uncheatable benchmark behaves in different situations. With the basic description given, it is important to explain the different situations involving the consumer's application and the benchmark tool. Firstly, it was defined that the benchmark tool must run during the entire time that the application is running (or the entire time it is on its processing peak). This already gives the impact that such benchmark has on the system. In order to collect more data, observe different behaviors and see the benchmark tool acting as a background service, the benchmark must be able to split the job over the time, that is, after every x time units, the benchmark should be able to perform part of the job so that in the end it accomplishes the task (in this experiment, to collect enough number of measurements). Every different choice of the parameter x represents one configuration of the scenario *two*. As one may notice, all data from the different configurations can be compared to the data collected by the scenario *one* with the goal of creating a more complete view of the benchmark's impact.

In this section the scenarios were described as well as their relation. The procedures that are part of each scenario were detailed and characteristics were given. Nevertheless, some basic remarks and extra information should be given. First, each one of the two scenarios was simulated in two environments, one comprised by a one-core processor and the other one comprised by a two-cores processor. The remaining configuration is the same. Further details on the environment simulation is given in section 4.3. The choice of adding both one-core and two-core machine to the experiment is due to the fact that, as yet to be described, the consumer's application might have benefits from the extra core while the benchmark might not.

4.2 Experiment requirements

With the scenarios already described, it is important to define the requirements that must be satisfactorily fulfilled to provide a meaningful experiment and therefore generate relevant results. Taking the scenarios into account, the following requirements were defined:

- It is necessary to simulate a cloud environment providing Infrastructure-as-a-Service. This environment will be evaluated by the given benchmark. In addition, it should allow one to represent as close as possible workloads of typical cloud systems. This allows one to assume that the collected data is close to the reality both in relation to the performance and the impact.

- It is necessary to choose and use relevant tools and technologies to achieve the aforementioned points. More specifically it is important to define which tool is used as the consumer's solution as well as which uncheatable benchmark is used.

- It is necessary to define relevant metrics that allow one to conclude about the points that constitute this work as well as to define the procedure to collect the measurements.

- It is necessary to present the collected raw data in a relevant and easy to understand manner, so that one is able to see the results generated by the experiment.

With the above points as well as the description of the scenarios, one has the main idea regarding the purposes of the practical part. Also, it becomes important to explain the choices and the process adopted in this work to satisfy those requirements. Each one of the key points demands detailed information on how it can be achieved, this is important to make the process clear. Therefore, the next sections are dedicated on explaining in details the choices adopted in this work.

4.3 Environment simulation

A typical environment offered by cloud providers providing Infrastructure-as-a-Service is based on virtualization. As virtualization leads to the notion of virtual machines (VMs), it makes sense to assume that VMs can simulate cloud environments. In fact, this is the technology used by cloud providers.

Therefore, the environment simulation was done through the use of a hypervisor that creates and controls pre-configured VMs.

With that defined, the next step is to choose the hypervisor. In this regards, the Xen hypervisor[1] presents itself as a reasonable choice, since it runs on the top of the hardware allowing multiple virtual machines to run simultaneously. The choice of a hypervisor that runs on the hardware instead of running as a normal application in a given operating system fits better to this work, since creating dependency to the operating system might heavily impact the isolation between the virtual machines as well as other parts of the system. To support the choice of xen, it is important to mention that Amazon's EC2, perhaps the most famous and used cloud service, uses Xen virtualization to instantiate its VMs. The last but also important factor to take in account is what type of virtualization should be used. The Xen hypervisor allows roughly full and para virtualization, although it is possible to use a mix of them. Despite the fact that para virtualization may provide optimal performance, the use of full virtualization excepts the need of support from kernel and allows the use of all set of hardware as well as provides a better isolation. Therefore, it was the type of virtualization used to create the VMs in this work.

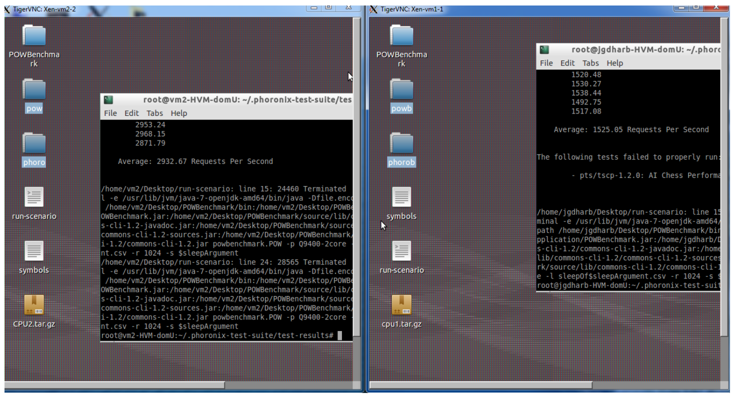
That said, the physical machine on which the Xen hypervisor was installed comprised the following attributes:

- Processor: Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GHz
- Memory: 8 GB RAM
- Kernel Version: Linux centos 3.10.43-11.el6.centos.alt.x86-64

Two virtual machines were created for the experiment. Each VM simulated both scenario *one* and scenario *two*. The configuration addressed to each VM comprised:

- Processor: Intel Core 2 Quad Q9400 @ 2.67GHz (1 and 2 Cores)
- Motherboard: Xen HVM domU
- Memory: 1 x 1024 MB RAM
- Operating System: Ubuntu 14.04
- Kernel: 3.13.0-24-generic (x86-64)
- System Layer: Xen HVM domU 4.2.4-33.el6

[1]http://www.xenproject.org/developers/teams/hypervisor.html



```
        2953.24
        2968.15
        2871.79

    Average: 2932.67 Requests Per Second


/home/vm2/Desktop/run-scenario: line 15: 24460 Terminated
l -e /usr/lib/jvm/java-7-openjdk-amd64/bin/java -Dfile.enco
 /home/vm2/Desktop/POWBenchmark/bin:/home/vm2/Desktop/POWBe
OWBenchmark.jar:/home/vm2/Desktop/POWBenchmark/source/lib/c
s-cli-1.2-javadoc.jar:/home/vm2/Desktop/POWBenchmark/source
commons-cli-1.2-sources.jar:/home/vm2/Desktop/POWBenchmark/
i-1.2/commons-cli-1.2.jar powbenchmark.POW -p Q9400-2core -
nt.csv -r 1024 -s $sleepArgument
/home/vm2/Desktop/run-scenario: line 24: 28565 Terminated
l -e /usr/lib/jvm/java-7-openjdk-amd64/bin/java -Dfile.enco
 /home/vm2/Desktop/POWBenchmark/bin:/home/vm2/Desktop/POWBe
OWBenchmark.jar:/home/vm2/Desktop/POWBenchmark/source/lib/c
s-cli-1.2-javadoc.jar:/home/vm2/Desktop/POWBenchmark/source
commons-cli-1.2-sources.jar:/home/vm2/Desktop/POWBenchmark/
i-1.2/commons-cli-1.2.jar powbenchmark.POW -p Q9400-2core -
nt.csv -r 1024 -s $sleepArgument
root@vm2-HVM-domU:~/.phoronix-test-suite/test-results#
```

```
        1520.48
        1530.27
        1538.44
        1492.75
        1517.08

    Average: 1525.05 Requests Per Second


The following tests failed to properly run:

        - pts/tscp-1.2.0: AI Chess Performa



/home/jgdharb/Desktop/run-scenario: line 15
minal -e /usr/lib/jvm/java-7-openjdk-amd64/
path /home/jgdharb/Desktop/POWBenchmark/bin
pplication/POWBenchmark.jar:/home/jgdharb/D
s-cli-1.2/commons-cli-1.2-javadoc.jar:/home
lib/commons-cli-1.2/commons-cli-1.2-sources
rk/source/lib/commons-cli-1.2/commons-cli-1
e -l sleepOf$sleepArgument.csv -r 1024 -s $
root@jgdharb-HVM-domU:~/.phoronix-test-suit
```

Figure 4.1: Virtual Machines - remote control

As one may notice, the only difference in the VMs is the number of core (1 and 2). The remaining configuration is exactly the same, as described above. The choice for the operating system is due to the easy of manipulation it provides, since the time remaining for the completion of this work was short. No need to state that both VMs were fully controled and therefore cheating played no role on the experiment, as desired. As the underlying machine was not physically accesible, the procedures on the VMs were done remotely, through the use of a SSH tunnel and the TigerVNC[2] viewer. The figure 4.1 illustrates the VMs running.

4.4 Choice of the tools

The next step consists of the choice of the tools that will run on the simulated environments (in this case, the VMs). As the goal comprises evaluating an uncheatable benchmark, the first tool is the benchmark itself. Afterwards, it is necessary to overload the VMs with specific workloads. This step is important because it is assumed in this work that cloud systems are not in standby mode waiting for workloads but they are constantly processing incoming loads from the consumer's application. Furthermore, it is desirable that the benchmark be able to split its job and run as a background service. To conclude, as the simulated environments do not include any application able to generate proper workloads, the need of a tool that acts as a main application is crucial.

[2]http://tigervnc.org/

4.4.1 The uncheatable benchmark

There are two points from the last chapter that must be remembered before going into further details. The first point is that, as stated in 3.2, this work aims in part to contribute to the work presented in [20], which implements a benchmark for cloud systems based on proof-of-work (POW) functions. The second point is that, as stated in 3.3, such POW benchmark disables most of the possibles ways of tampering results and therefore it is considered as an uncheatable benchmark. In addition, there is little to none work relating uncheatable benchmark and cloud computing. That said, it becomes obvious that the POW benchmark is the one used in this experiment. In a nutshell, the benchmark program is a processor-focused command-line Java tool that runs the three implemented Proof-Of-Work functions: the *HashCash*, the *HashCashLin* and the *Extracting Square Roots*. The benchmark suite is able to measure the execution time of all functions and the execution rate of the functions. The execution time measurement consists of checking how long the function needs to find a solution, the execution rate measurement consists of checking how many solutions can be found by the function in a pre-determined time. These two metrics therefore are responsible for representing the performance of a given machine. The detailed information regarding the POW functions can be found in [20]. Although the POW benchmark is composed by three POW functions, it is interesting to take the best out of the benchmark offerings to use in the experiment. Firstly, the analysis of such POW suite has shown that the most suited function to be used as a benchmark is the HashCashLin implementation, even though all functions share similarities. This choice was based on the tests presented in the related work in question. Therefore, the HashCashLin is the function that will be taken in account in this experiment.

With the proper choices made, the next step was to adapt the POW benchmark to the needs of this work. In this regard, the suite was modified to satisfy the purposed scenarios. Firstly, the suite was changed to run just the HashCashLin implementation. In addition, the warm-up phase was taken out of the execution flow because that would not make difference in the experiment, specially if taking in account the long time it keeps running on the virtual machines. Furthermore, the Java class *TimeUnit*[3] was used through the call of the function *sleep* and allowed to run the benchmark as a background service splitting the job for every x (*sleep*) time units . In order to automatize that, an additional parameter was added to the call of the suite, which is the sleep time in Milliseconds. The function itself was called after each collected measurement. This allowed the POW benchmark

[3]http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html

to collect one measurement and wait a determined amount of time before collecting another one. Without this change the suite would keep collecting measurements until the end is reached. Another important point is that the POW benchmark was modified to run in a loop, that is, to run the benchmark for as many times as desired so that the application is impacted during its entire runtime.

4.4.2 The main application

The tool used as the main application is the one that represents the consumer's application that is deployed on virtualized environments and that is affected by the POW benchmark. To make a decision in this regards, it is important to firstly take into account the proper scope in which the POW benchmark works. As it is a benchmark to evaluate performance of processors, it is convenient to have an application that heavily makes use of the processing power. As the experiment wants to verify the performance and impact of a processor-focused benchmark, it would not make sense for instance to choose as main application any network-based tool. As a consequence, it is desirable a processor-focused application that is able to generate relevant workloads that require a high parcel of the processing power so that the difference can be noticed and correctly related to the POW benchmark. In addition, such application should be able to perform in the simulated environment (therefore in a cloud environment). As one may imagine, benchmarks are known by evaluating performance through relevant workloads in the domain in question. The use of a processor-focused benchmark as a main application would not just apply the desirable workload but would also perform an evaluation of the system in different situations (for example, in scenarios with and without the POW benchmark). That said, the experiment had a benchmark acting as a main application so that it fulfills the needs aforementioned and also facilitates the process of impact evaluation.

The chosen benchmark (from now on called just application) is the *Phoronix Test Suite*[4]. "The Phoronix Test Suite is the most comprehensive testing and benchmarking platform available ... designed to effectively carry out both qualitative and quantitative benchmarks in a clean, reproducible, and easy-to-use manner". These are important attributes, since it is not convenient to work with tools that might be difficult to deal with and also may not apply qualitative workloads to the processor. Furthermore, the fact that the suite is composed by several benchmarks allows different workloads to be applied to a given system and therefore have an impact evaluation view of not just

[4]http://www.phoronix-test-suite.com/

one single component as parameter. As an open-source offering compatible in many operating systems (specially Linux), the suite enables automated and repeatable runs, allowing the creation of different scenarios without the need of user's interaction all the time. A complete repeatable run can be created as simple as by running one command in a Linux terminal. Another important characteristic that helps in the choice of the Phoronix suite is its adaptability to run on different platforms such as cloud computing infrastructures. This characteristic allows one to use the tool into virtual machines without loss of any of its attributes or efficiency. Before starting with the specific details on how the suite was used, it is important to highlight and clarify here two features that came in handy:

- The Phoronix suite incorporates statistical accuracy, that is, for any test it detects whether the calculated standard deviation between runs exceeds a predefined threshold defined by the user. If so, the suite automatically adds calls to the test so that it is executed as many additional times as needed to ensure that the reported result is accurate. The number of additional times is up to a limit of time defined by the user (to avoid an infinity number of calls). This feature not only automatically generates important information as well as it allows the user to define which rate of accuracy the results should respect in a trade-off with the execution time since it may require a considerable higher amount of time to reach a given accuracy rate.

- The Phoronix suite offers environment variables that can be adjusted in the calls to adapt the runs to the user's needs in a fast and easy way. There are variables to force tests to run a certain number of times, without having to edit the test's configuration file. There is also possible to adjust the time of the run as well as many other parameters that can be found in the respective documentation.

At this point, one has the necessary view regarding the application that is used in the experiment. The exact procedure used to run the Phoronix suite will be described later. To have a more detailed view, the benchmarks that are part of the processor suite and that together compose the whole application are listed below:

- NAS Parallel Benchmarks - Test / Class: MG.B

- OpenSSL - RSA 4096-bit Performance

- Apache Benchmark - Static Web Page Serving

- TSCP - AI Chess Performance
- John The Ripper - Traditional DES
- John The Ripper - Blowfish
- TTSIOD 3D Renderer - Phong Rendering With Soft-Shadow Mapping
- x264 - H.264 Video Encoding
- GraphicsMagick - HWB Color Space
- GraphicsMagick - Local Adaptive Thresholding
- GraphicsMagick - Sharpen
- GraphicsMagick - Resizing
- Himeno Benchmark - Poisson Pressure Solver
- 7-Zip Compression - Compress Speed Test
- C-Ray - Total Time
- Parallel BZIP2 Compression - 256MB File Compression
- Smallpt - Global Illumination Renderer; 100 Samples
- Crafty - Elapsed Time
- FLAC Audio Encoding - WAV To FLAC
- LAME MP3 Encoding - WAV To MP3
- FFmpeg - H.264 HD To NTSC DV
- Tachyon - Total Time
- Timed MAFFT Alignment - Multiple Sequence Alignment

The 23 benchmarks comprise different kind of tests such as encoding, compression, rendering, parallel execution and floating point calculation among others. In addition, some of the tests make use of multi-core machines in contrast with the uncheatable benchmark, that is designed for one-core machines and has no benefits from any extra core. This fact justifies the choice of running the scenarios in two different VMs and compare the impact on both cases. Further information regarding each one of the tests can be found in the Openbenchmarking.org[5] website. This platform comprises the whole set of tests of the Phoronix Suite as well as a description of them.

4.5 Analysis Methodology

This section aims to explain how the collected data is further compared and analyzed as well as in which form it is presented to provide a clear view. This covers the two last requirements as defined in 4.2. The first point that should be taken into account is that the application and the POW benchmark have distinct metrics and procedures to obtain their results. That said, it is convenient to explain each one of them separately.

The POW benchmark stores the collected data in date-stamped log files in .csv format. Each full run (which may include several runs in a loop) generates a unique log file that represents the results correspondent to the choice of the parameter *sleepTime*. Again, this parameter determines the interval in which the benchmark waits before going further in completing another part of the task (one part of the task means to collect one more measurement). The task is finished when the application(Phoronix processor suite) has finished its job. Therefore, the POW benchmark runs exclusively during the period that the application is running. This impacts the system during the entire time and also allows the storage of data collected just during such period. Moreover, as already stated before, The POW benchmark metric is the execution time and the execution rate. This allows a series of statistical values to be derived such as maximum and minimum values, the sample arithmetic mean, the standard deviation and the coefficient of variation among others. The metric that is taken in account for the POW benchmark is the execution time, since it is not convenient to analyze both metrics again as in [20]. From now on when this work refers to precision of the POW benchmark it is taking into account just the precision regarding the execution time. In addition, the statistical value that better fits the needs of this work is the coefficient of variation and therefore it is the one to be used. Following [29], "The coefficient of variation (CV) is a normalized measure of dispersion. It is defined as the ratio of the standard deviation to the mean and known as the variation coefficient". The CV tells one how some values (in this case, execution time) can be dispersed from the mean value (which is the desirable value for all measurements). The higher the coefficient the less accurate the results are. Using the CV as statistical value allows one to also compare different sets of data (in this experiment, the sets generated by two different VMs).

[5]http://openbenchmarking.org/

The application stores the data in a set of files (including XML files) that allows a graphic visualization through any web browser. However, the suite also allows one to convert the results to .csv, .pdf and .txt files and this comes in handy. The use of .csv files eases the processing of data. The metrics used by the application varies among the tests. Metrics include interactions per second, time spent for execution, frames per second and number of operations. For some tests lower values are better and for other higher are better. For each run of the application, the whole set of tests evaluate the system in their particular procedures and the results of all tests compose the final stored file. Each single test was set to run for 10 times and in case the result is not inside the threshold determined for the standard deviation, the test would run more times up to a limit of 30 minutes running. In addition, multiple result files can be merged into a single file, this eases even more the processing of data generated by multiple runs, which is the case of this experiment. Note that, as mentioned before, during the entire time of each run of the application the POW benchmark is running as well. If we consider the *sleepTime* as zero then the POW benchmark is running without breaks and this generates the maximum impact on the system. As we increase the parameter, the POW benchmark waits longer to collect measurements and therefore the system might be less impacted. We consider the total time the one needed by a full run of the application.

Note that the parameter *sleepTime* and its possible different impacts on the system is exactly the interesting point. This works aims to check the changes generated by the choice of such parameter (if any) and therefore assess on the impact. The evaluation of the impact is done through a simple comparison. It is considered that the performance of the application running alone represents the 100 % value. In the cases the POW benchmarking is running as well (scenario *two*) the performance evaluation is done by determining the difference between the two values and calculating its correspondent percentage, which is lower or equal to 100 % (It does not make sense if the application has a better performance while running in parallel with the benchmark rather than running alone). Furthermore, each single choice of the *sleepTime* produces a different log file of the POW benchmark with a different number of measurements that might produce higher or lower precision if compared to the others. In fact, a relation of the impact and the precision of the POW benchmark for a given *sleepTime* can be created. To create and analyze such relation is the main goal of this work. The set of the application's results can also generate statistical values such as mean, maximum and minimum value. To obtain the variation on the precision this work merges the coefficient of variation generated by each log file of the POW benchmark into one single file. This ends the description on what the analysis is based on and on how it is done.

With the whole set of data collected and processed, the results can be presented through graphics and tables as well as images when applicable. This is the way this work presents the results. These options, in opposite to raw text, make the understanding process easier and softer. Since the analysis design was described. The next step is to go further into the run of the experiment and present the results.

Chapter 5

Analysis

This chapter presents the results obtained within the experiment and assesses on the performance of the POW benchmark based on such results. To automatize the whole process of running the scenarios a script was created. Roughly, the script was responsible for running the application for m times, each time with a different *sleepTime* argument that was increased by t units after every run. Both the application and the POW benchmark were run at the same time. They were started at the same time and as soon as the application finished the POW benchmark was stopped too. This ensured that the POW benchmark was collecting measurements and impacting the system during the application's runtime and not longer or shorter than that.

To start, the application was run alone on both VMs. This successfully concluded the scenario *one*. To get a better accuracy it ran about four times on each VM. All the results were stored in .csv files and served as basis for comparison since they represent the 100 % performance of the application as stated in 4.5. The application took about 4 hours to run on the two-core virtual machine and about 7 hours to run on the one-core virtual machine. The first remark is that, as desirable, the processor was being fully allocated for the execution of the application. This can be seen in the figure 5.1, which shows the CPU usage for each VM controlled by the Xen hypervisor. This ensured that there were no extra resources that could be allocated exclusively for the POW benchmark and therefore could sabotage the results. In the figure it is possible to see the one-core VM (VM1-1) and the two-core VM (VM2-2). The Domain-0 is the physical machine controlled by the Xen hypervisor. Furthermore, the different runtime in both VMs already shows that the application has benefits from the extra core, that is, it runs faster on the two-cores machine. This finding is important because helps to explain much of the results further described in this chapter.

That said, it is a good way to start by showing the results referent to

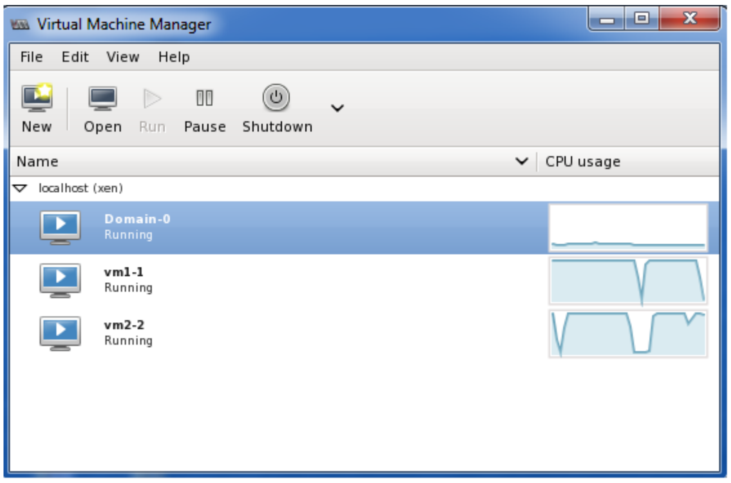


Figure 5.1: CPU usage by the application running alone

the application. The first conclusion is that in fact the POW benchmark impacts the system in a varying manner which depends on the *sleepTime* value. The performance achieved by the application alone is not reached when the POW benchmark is acting as a background service. The chart on figure 5.2 shows the average impact as well as the maximum value observed for each choice of the *sleepTime* on the one-core virtual machine. Remember that such parameter determines the interval that the POW benchmark should wait until it collects one more measurement. During such waiting interval the CPU is required just by the application. In general lines, as higher the interval is, higher is the time that the CPU is not being used by the POW benchmark and *vice versa*.

As it is possible to see in the figure 5.2, the time is given in Milliseconds and the values in percentage, since they are mathematically obtained by subtracting them from the value correspondent to the 100 % performance(scenario *one*). The first point to observe is that when the parameter is set to 0 Milliseconds, which means no interval between measurements, the average impact is in its highest value which corresponds to approximately 49 %. This is easily explained by the fact that in this case the application and the POW benchmark are sharing the processor during the entire runtime. In opposite to that, any waiting interval greater than zero (even the smallest one) already decreases the average impact. Still, the difference in the first interval (0-250 Milliseconds) is higher than in the other ones. Furthermore, the behavior of the line shows that the difference is getting lower as we compare intervals referent to higher *sleepTime* values up to the point where it is possible to notice the tendency to zero, both on difference and on the im-

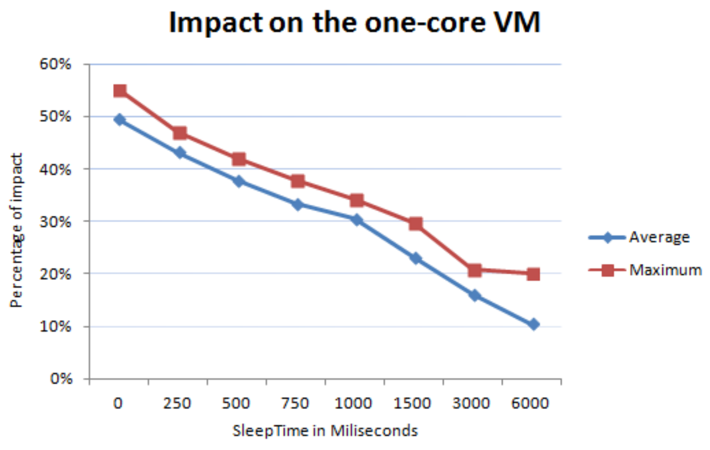


Figure 5.2: Maximum and Average Impact on the one-core VM

pact. At this point it can be observed that increasing such parameter even more would decrease the impact up to a point where no significant difference would be noticed. As to be described later, the number of measurements collected and also the precision of the POW benchmark have a relation with the behavior observed in this chart. Another point to be highlighted is that, as in the average approach, the difference regarding the maximum impact is also one of the highest in the first interval. This shows that such interval is the one in which decisions for the *sleepTime* makes more difference and where a wrong decision taken by the consumer could cause a significantly higher impact. This finding is important specially regarding the line representing the maximum impact because the values that compose such line in every interval are originated from one single benchmark and not from the average. As the 23 benchmarks that compose the Phoronix suite have different methods to evaluate the system, say a given consumer's application shares behaviors (that is, it is similar in the way it runs) with a single benchmark representing a certain interval in the chart, then that would mean a quite similar impact (which is maximum) to be expected by the consumer.

The next remark to be done is that, according to what was thought, it turned out that the results regarding the two-core VM are quite similar to the ones presented so far. In fact, the behaviors described in the last paragraph are the same observed for both VMs. The only point that deserves to be highlighted is the comparison of the average impact for both machines, which can be seen in the chart on the figure 5.3.

With the behaviors and tendencies being the same, what is clear is that the overall impact on the two-core VM is lower. This can be explained by the

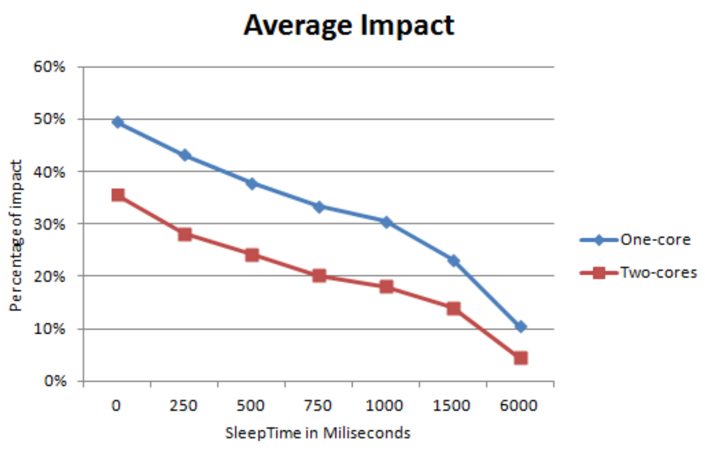


Figure 5.3: Average Impact on both VMs

fact that, as briefly cited here and explained in [20], the POW benchmark has no support for multi core processors. Furthermore, tests in the respective work have shown that the benchmark ”runs on one core ignoring the shift towards multi core processors”. As the POW benchmark has no benefits from the extra core, the difference on performance exists and it was stated previously the runtime difference regarding the application, it seems that in fact the application includes tests in which certain procedures take advantage of the extra core. Therefore the performance is higher (as it was in this experiment) and one consequence is such lower impact.

Regarding the results from the POW benchmark, it is important to observe the number of measurements collected as well as the difference on the precision in each choice of the *sleepTime*, as explained in 4.5. To start, the table on figure 5.1 shows, for each VM and for each case, the number of measurements collected in Millions (remember that the POW benchmark collected measurements during the entire application's runtime).

Table 5.1: Number of measurements collected (in Millions)

	0	250	500	750	1000	1500	6000
One-core	2,2	1,6	1,55	1,2	1,1	0,68	0,06
Two-cores	2,3	1,2	0,83	0,62	0,45	0,3	0,025

As expected, obviously the number of measurements collected differs in each case (being greater at 0 and lower as the parameter increases), since the POW benchmark had to wait longer in every new case, it also had a shorter runtime collecting measurements. Also, the difference between the

VMs is due to the fact that the application runs faster on the two-cores VM and therefore the POW benchmark is stopped earlier, having collected less measurements in the end.

The main conclusion that can be drawn with all the data collected and shown so far is that as greater the number of measurement is, also higher is the impact on the system. Choosing a low value as parameter (say 0) allows the POW benchmark to run longer and therefore collect more measurements, however it was shown that at 0 the impact on the performance is at its highest value. Another point to be highlighted in this experiment is the huge set of data generated by the POW benchmark. About 2GB was stored as measurements for each VM, which totals 4GB. This differs in much with the tests and results shown in [20]. There, it was taken a "higher" number of measurements (512) in order to mitigate the effect of randomness, which is characteristic to the POW benchmark. However, that produced a much smaller file which eases the processing. Still according to [20], that number of measurements was necessary because the less the POW benchmark collects to form the mean value, the more inaccurate the results are since the randomness might generate a high standard deviation. With that said and with the greater number of measurements collected by this experiment, it is plausible to suppose that the precision here is equal or greater than the one seen in [20] and therefore such precision is acceptable. To compare such difference was not the main goal of this work. However, to verify how the precision behaves in the different cases may lead to a better understanding on how to take a wise decision regarding the adjustments of the POW benchmark (adjustment of the *sleepTime* parameter).

As mentioned before, the coefficient of variation was chosen to represent the variation in the precision of the results. In this regard, as lower the coefficient is, higher is the precision. The first remark is that the results turned out to be completely different from what was thought initially. It was supposed that the greater number of measurements in some cases would represent in the end a more precise result. That did not happen. As it can be seen in the chart on figure 5.4, there is no pattern on the behavior of the results.

Sometimes the coefficient of variation is getting lower but that suddenly changes and it gets higher again. This randomness can be observed for both VMs. One explanation for that is the randomness nature of the POW benchmark itself, even the higher number of measurements collected in this experiment was not able (in this specific case) to minimize such characteristic. However, one should put aside such fact and observe that, within the range of variation, the difference for all choices of the *sleepTime* is small and that such difference does not follow any clear tendency. Therefore, if choosing

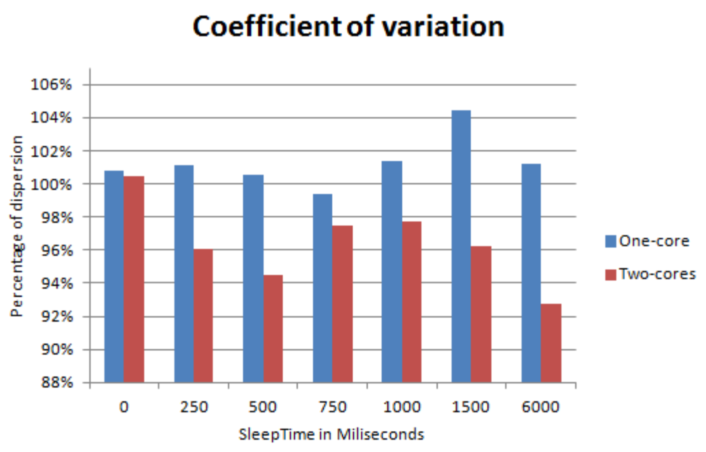


Figure 5.4: Coefficient of Variation for the collected measurements

either 0 or 6000 as parameter generates similar results in terms of precision, in terms of impact on the system the choices are completely different.

With everything that was drawn so far, it is just matter of putting everything together and see that for the scenarios presented in this experiment the better situation for a given consumer would be choosing the highest possible value for the parameter *sleepTime*. This puts the POW benchmark to wait longer before collecting one more measurement. It was seen that longer periods mean lower impacts on the system, therefore the longest period is the desirable choice. However it should be analyzed if such choice is still able to produce precise results. In this regards, all cases presented very small differences on the precision and no clear tendency. This means that the consumer can easily keep the choice for the longest period (highest *sleepTime* value) and have the lowest impact on its system without really compromising the precision. One important remark is that this situation is valid for the scenarios and situations presented in this specific experiment. As the application ran for several hours, the POW benchmark was still able to collect enough number of measurements in each one of the cases (much more than 512 as stipulated in [20]), so that the precision had not really changed between the cases. This implies that one could choose the longest period and minimize the impact.

That said, for a general scenario and a given environment, consumers could first analyze the total runtime in which their solution must allocate all the available resources (processing peaks). This is the situation where cheating might have its worst effect and where benchmarking comes in handy. Afterwards, consumers could draft how their chosen (uncheatable) benchmarks must be configured. To find out such configuration it should be taken into account the minimum requirements necessary to provide enough precision. In addition, the corresponding impact generated by this given configuration should be analyzed. With this analysis done, consumers could then decide whether it is worthing or not to give up on a given % of the performance in order to benchmark the system and ensure that no cheating is being done.

This chapter has shown the findings of the experiment as well as the relation they have with each other. Explanations for each one of the findings were given as well as a suggestion on how to proceed in general scenarios.

Chapter 6

Conclusion

This chapter sums up the work done as well as the results obtained. The goal was to assess on the performance of an uncheatable benchmark for cloud systems, what is believed to have been reached through the proposed experiment and its findings. Obviously that the results are originated from one single experiment, however, the proposed scenarios do not differ in much from the real ones in which providers and consumers interact.

This work has lead the reader through the basics of cloud computing to the definition and goals of computing benchmarking. In addition, the need for monitoring as well as evaluating cloud systems in order to check whether they meet the SLA was presented. Afterwards, two motivations for cloud benchmarking were identified and discussed. Benchmarking a cloud component that is under control of the cloud provider could spot possible cheating on the SLA. However, benchmarks are designed for traditional systems and therefore may not deal well with the new cloud computing paradigm. Also providers can design ways of tampering benchmark's results to promote a false good image. As a solution, uncheatable benchmarks come in handy as they are resistant to tampering and may make the effort of cheating not worthing. However, one drawback is that benchmarking may demand resources that are allocated to the consumer's solution, therefore impacting the system. From this situation the motivation of this work was originated.

An experiment was designed to verify the impact of an uncheatable benchmark on cloud systems as well as its capability to provide precise results. Such experiment simulated a cloud environment and also workloads that could represent real consumer's applications. The uncheatable benchmark used was conveniently the one designed by [20] in order to also contribute to such work. The idea was to test the benchmark as a background service, splitting the job over the time to possibly minimize impacts. In this specific case, splitting the job means to collect a new measurement after a given pre-determined interval of time. In addition, such benchmark ran just during the time the workloads were being processed in order to prevent biased results. The findings showed that as longer the interval between jobs is, lower is also the impact on the system and *vice versa*. However, splitting the job on such way resulted on less measurements collected and that could produce variation in the precision of the results. This was surprisingly not the case for this experiment due to the fact that the total runtime was still able to provide enough measurements and therefore the precision had not changed much. In summary, this work showed results that can represent the configuration of general scenarios in cloud environments.

Bibliography

[1] P. Mell and T. Grance, "The nist definition of cloud computing," US National Institute of Standards and Technology, Tech. Rep., September 2011.

[2] B. Furht and A. Escalante, *Handbook of Cloud Computing*, ser. Computer science. Springer, 2010. [Online]. Available: http://books.google.de/books?id=jLNGCPs6rr4C

[3] S. Microsystems, "Introduction to cloud computing architecture," Sun Microsystems, Tech. Rep. [Online]. Available: https://java.net/jira/secure/attachment/29265/CloudComputing.pdf

[4] A. E. Inc., "What is cloud: Basic concepts and terminology," May, 2010. [Online]. Available: http://whatiscloud.com/basic_concepts_and_terminology/index

[5] S. C. Lee Badger, Robert Bohn and other, "Us government cloud computing technology roadmap," US National Institute of Standards and Technology, Tech. Rep., November 2011.

[6] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud Computing Principles and Paradigms.* Wiley Publishing, 2011.

[7] A. E. Inc., "What is cloud: Cloud delivery models." [Online]. Available: http://whatiscloud.com/cloud_delivery_models/index

[8] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE '08*, Nov 2008, pp. 1–10.

[9] M. Williams, *A Quick Start Guide to Cloud Computing: Moving Your Business Into the Cloud*, ser. New Tools for Business. Kogan Page, Limited, 2010. [Online]. Available: http://books.google.de/books?id=IRQSRAAACAAJ

[10] T. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, April 2010, pp. 27–33.

[11] Wikipedia, "Virtualization." [Online]. Available: http://en.wikipedia.org/wiki/Virtualization

[12] I. Corporation, "Planning guide: Virtualization and cloud computing," Intel Corporation, Tech. Rep., August 2013. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/guides/cloud-computing-virtualization-building-private-iaas-guide.pdf

[13] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, April 2010, pp. 222–226.

[14] O. Corporation, "Oracle® vm user's guide." [Online]. Available: http://docs.oracle.com/cd/E20065_01/doc.30/e18549/intro.htm

[15] D. Marinescu, *Cloud Computing: Theory and Practice.* Elsevier Science, 2013. [Online]. Available: http://books.google.de/books?id=mpcBw1OnyIgC

[16] M. Kavis, *Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)*, ser. Wiley CIO. Wiley, 2014. [Online]. Available: http://books.google.de/books?id=NcrDAgAAQBAJ

[17] L. Ye, H. Zhang, J. Shi, and X. Du, "Verifying cloud service level agreement," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, Dec 2012, pp. 777–782.

[18] B. EECS Department, University of California, "Above the clouds: A berkeley view of cloud computing," February 2009.

[19] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the weather tomorrow?: Towards a benchmark for the cloud," in *Proceedings of the Second International Workshop on Testing Database Systems*, ser. DBTest '09. New York, NY, USA: ACM, 2009, pp. 9:1–9:6. [Online]. Available: http://doi.acm.org/10.1145/1594156.1594168

[20] F. Köppe, "Computer performance verification in cloud environments," diploma thesis, Technische Universität Berlin, 2010.

[21] Wikipedia, “Benchmark (computing).” [Online]. Available: http://en.wikipedia.org/wiki/Benchmark_(computing)

[22] E. Folkerts, A. Alexandrov, K. Sachs, A. Iosup, V. Markl, and C. Tosun, “Benchmarking in the cloud: What it should, can, and cannot be,” in *Selected Topics in Performance Evaluation and Benchmarking*, ser. Lecture Notes in Computer Science, R. Nambiar and M. Poess, Eds. Springer Berlin Heidelberg, 2013, vol. 7755, pp. 173–188. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36727-4_12

[23] K. Huppler, “The art of building a good benchmark,” in *Performance Evaluation and Benchmarking*, ser. Lecture Notes in Computer Science, R. Nambiar and M. Poess, Eds. Springer Berlin Heidelberg, 2009, vol. 5895, pp. 18–30. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10424-4_3

[24] ——, “Benchmarking with your head in the cloud,” in *Proceedings of the Third TPC Technology Conference on Topics in Performance Evaluation, Measurement and Characterization*, 2012, pp. 97–110. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32627-1_7

[25] R. Iosup, R. Prodan, and D. Epema, “Iaas cloud benchmarking: Approaches, challenges, and experience (invited paper).”

[26] J.-Y. Cai, R. Lipton, R. Sedgewick, and A. C.-C. Yao, “Towards uncheatable benchmarks,” in *Structure in Complexity Theory Conference, 1993., Proceedings of the Eighth Annual*, May 1993, pp. 2–11.

[27] Q. Huang, L. Ye, X. Liu, and X. Du, “Auditing cpu performance in public cloud,” in *Services (SERVICES), 2013 IEEE Ninth World Congress on*, June 2013, pp. 286–289.

[28] H. Zheng, K. Li, A. Blaisse, C. Tan, and J. Wu, “Cloud cpu verification scheme for individual end users,” *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, vol. 2, no. 6, pp. 377–390, 2013. [Online]. Available: http://iaesjournal.com/online/index.php/IJ-CLOSER/article/view/5651

[29] Wikipedia, “Coefficient of variation.” [Online]. Available: http://en.wikipedia.org/wiki/Coefficient_of_variation