View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UPCommons. Portal del coneixement obert de la UPC

FUNDAÇÃO EDSON QUEIROZ UNIVERSIDADE DE FORTALEZA – UNIFOR CENTRO DE CIÊNCIAS TECNOLOGICAS-CCT CURSO ENGENHARIA EM TELECOMUNICAÇÕES

UM ESTUDO DA SIMULAÇÃO DE SERVIÇOS DA AMAZON EC2 UTILIZANDO O SIMULADOR CLOUDSIM

Alberto Crespo Guzmán

Fortaleza - Ceará
2011

Alberto Crespo Guzmán

UM ESTUDO DA SIMULAÇÃO DE SERVIÇOS DA AMAZON EC2 UTILIZANDO O SIMULADOR CLOUDSIM

Monografia apresentada para obtenção dos créditos da disciplina Trabalho de Conclusão do Curso do Centro de Ciências Tecnológicas da Universidade de Fortaleza, como parte das exigências para graduação no Curso de Engenharia em Telecomunicações.

Orientador: Prof. Dr. Américo Falcone Sampaio.

Fortaleza - Ceará
2011

UM ESTUDO DA SIMULAÇÃO DE SERVIÇOS DA AMAZON EC2 UTILIZANDO O SIMULADOR CLOUDSIM

Alberto Crespo Guzmán

PARECER: _____

DATA: ___/___/____

BANCA EXAMINADORA:

_____
Américo Tadeu Falcone Sampaio (Ph.D.)

_____
Nabor das Chagas Mendonça (Ph.D.)

LISTA DE ABREVIATURAS

TI - Tecnologia da Informação

PDA – *Personal Digital Assistant*

REST - *Representational State Transfer*

GAE - *Google App Engine*

OS – *Operative System*

IDE - *Integrated Development Environment*

EC2 - *Elastic Compute Cloud*

S3 - *Simple Storage Solution*

SaaS - *Software as a Service*

PaaS - *Platform as a Service*

IaaS - *Infrastructure as a Service*

JVM - *Java Virtual Machine*

AMI - *Amazon Machine Image*

API – *Application Programming Interface*

AWS – *Amazon Web Service*

HCP – *High Performance Computing*

CPU – *Central Processing Unit*

HTTP – *Hyper Text Transfer Protocol*

HTTPS – *Hyper Text Transfer Protocol over SSL*

IP – *Internet Protocol*

QoS – *Quality of Service.*

VM – *Virtual Machine*

SLA - *Service Level Agreements*

FCFS – *First Come First Service*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

O conceito de *CloudCompunting*, ou como é conhecido no Brasil, computação em nuvem, está ganhando uma ênfase cada vez maior na pesquisa mundial e também vêm sendo vista como estratégica pelas grandes empresas de Tecnologia de informação (TI). Dados da empresa Gartner (SAMPAIO et al. 2009) mostram que espera-se que o mercado de *CloudComputing* atinja um volume de negócios de 42 bilhões em 2012.

Apesar da dependência com a TI crescer cada vez mais existe uma preocupação constante com a redução de custos. As empresas têm que gastar muito capital com licenças de aplicações e também para instalar, configurar e manter software. Além disto, muitas empresas optam por comprar servidores suficientemente potentes para poder suportar a demanda máxima prevista em períodos de pico. Isto muitas vezes resulta em parques computacionais ineficientes, pois na maioria do tempo esta infraestrutura é subutilizada já que o tráfego de pico acontece apenas em momentos específicos e sazonais.

A partir do ano 2005 surgiu a idéia de computação em nuvem, que abriu novas possibilidades para esse setor. Este paradigma foca em oferecer recursos, aplicações e plataformas como serviço pela internet com modelos de preço baseados no consumo efetivamente utilizado (UNIV. of CALIF. 2009). Isto começou graças a provedores de internet de grande escala como *Google* ou *Amazon* (GOOGLE, 2010, AMAZON, 2009a), entre outros, que construíram sua própria infraestrutura. Eles tiveram a idéia de alugar os recursos ociosos de sua própria infraestrutura para usuários que preferem não adquirir seus próprios recursos, mas sim, alugar os mesmos pagando pelo uso efetuado.

Uma definição interessante de *CloudComputing* é a que *George Gilder* (INF. FACTORIES 2006): um sistema de recursos horizontalmente distribuídos, apresentado como serviços virtuais cujos recursos TI são altamente escaláveis, configurados e gerenciados como um conjunto contínuo.

Para se beneficiar deste modelo computacional tão complexo precisa-se de mecanismos que permitam fazer simulações para tentar descobrir e analisar distintos cenários de aplicações que utilizam nuvens. Por exemplo, pode-se simular uma determinada carga a ser imposta em uma aplicação de comércio eletrônico e tentar prever quanto custaria para executar tal aplicação em um ou mais provedores de nuvem distintos. As simulações, portanto, são essenciais para tentar prever o comportamento e os custos de uma aplicação na nuvem antes de colocar a aplicação em operação.

Neste projeto utiliza-se o *CloudSim* (CLOUDS 2010) como simulador de cenários *CloudComputing*. *CloudSim* basea-se em um conjunto de ferramentas de simulação extensível que permite a modelagem e simulação de sistemas de computação em nuvem e ambientes de aplicação de aprovisionamento. *CloudSim* suporta tanto o sistema e modelagem do comportamento de todos os componentes do sistema Cloud.

O objetivo deste trabalho é a implementação das características oferecidas pelo conhecido provedor de serviços de computação em nuvem, *Amazon*, no simulador *CloudSim*. Faz-se uma extensão do código em Java do *CloudSim* para poder inserir os diferentes parâmetros que *Amazon EC2* nos oferece, tais como os tipos de serviços oferecidos e os diferentes preços e recursos computacionais como memória e processamento.

O Capitulo 1 tem como objetivo fazer uma introdução sobre o conceito de *Cloud Computing* e seus pontos chave.

No Capítulo 2 vai ter uma explanação sobre o serviço disponibilizado pela *Amazon* denominado de *Elastic Compute Cloud* (EC2) e *Amazon S3.*

No Capítulo 3 faz-se uma explanação sobre o simulador de cenários *CloudSim* assim como toda sua estrutura de forma detalhada.

No Capítulo 4 serão mostradas e explicadas as mudanças no código de *CloudSim* para fazer ele compatível com as características de *Amazon EC2*.

1. COMPUTAÇÃO EM NUVEM

O objetivo principal desse capítulo é apresentar os conceitos que envolvem a idéia de computação em nuvem. Este iniciará descrevendo algumas tecnologias de Cloud relacionadas e alguns conceitos importantes.

1.1. CONCEITOS RELACIONADOS

1.1.1. VIRTUALIZAÇÃO

Em computação, (SUNMICROSYSTEMS 2009) virtualização refere-se à captação de recursos de computador, chamado de *hypervisor* ou VMM (Virtual Machine Monitor), quem trata, gerencia e intermedeia os quatro principais recursos de um computador (CPU, memória, rede, armazenamento) podendo assim alocar dinamicamente os recursos entre todas as máquinas virtuais definidas pelo computador físico (host). Assim podemos ter várias máquinas virtuais rodando no mesmo computador físico.

Dentre alguns dos tipos de virtualização (VIRTUAL_LINUX 2010a), estão (ver Figura 1):

- Para-virtualização: a para-virtualização consiste em executar sistemas operacionais *guests* sobre outro sistema operacional que atua como (*hypervisor*) *host*. Os *guests* têm que se comunicar com o *hypervisor* para realizar a virtualização. Isso proporciona um bom rendimento e possibilidade de executar diferentes sistemas operativos como *guests*.
- Virtualização completa: é similar a para-virtualização, mas não precisa de sistemas operacionais *guest* que colaborem com o *hypervisor*. Esse método tem todas as avantagens da para-virtualização, mas sem modificar os *guests*. A desvantagem é que tem que poder suportar a arquitetura de hardware utilizada e também existe uma perda de performance.

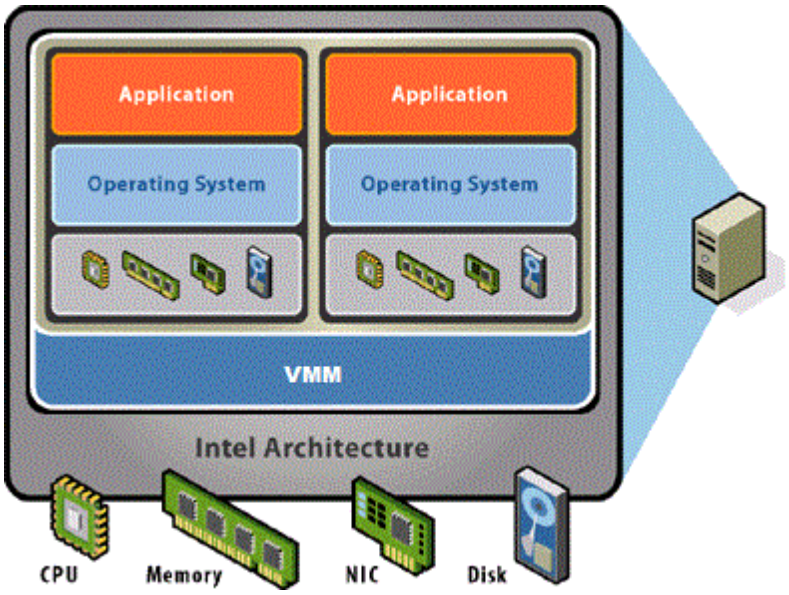


Figura 1 - Camada de Virtualização. Fonte: (http://blog.corujadeti.com.br/virtualizacao-alguns-pontos-quanto-a-tecnologia/)

1.1.2. SERVIÇOS WEB

Uma tecnologia que permite (WORDPRESS 2010a) o desenvolvimento da computação em nuvem é o *Web Service*. Serviço web é um conjunto de protocolos e padrões utilizados para a troca de dados entre aplicações. Essa tecnologia é acessível remotamente através da web. Isso permite a utilização da infraestrutura da web existente, sem ter que instalar nenhum software ocultando a implementação real, já que o serviço pode ser implementado por diferentes tecnologias no cliente e no servidor desde que um determinado padrão seja respeitado como SOAP ou REST.

1.2. VISÃO GERAL DA COMPUTAÇÃO EM NUVEM

A computação em Nuvem é uma área que está passando por um rápido avanço tanto no meio acadêmico como no meio da indústria. Esta tecnologia, que visa oferecer virtualização e recursos elásticos como utilidades para os usuários finais, tem o potencial para suportar a realização plena da "computação como uma utilidade" no futuro próximo.

Anteriormente, o desenvolvimento de tais aplicações precisava da aquisição de servidores com uma capacidade fixa capaz de lidar com a demanda máxima prevista a aplicação, a instalação da infraestrutura de software de toda a plataforma que suporte o aplicativo e configuração do aplicativo. Mas os servidores estavam subutilizados na maioria das vezes por causa dos picos de tráfego, já que isso ocorre somente em determinados períodos de tempo curtos. Com o avanço da nuvem, a hospedagem tornou-se mais barata e mais fácil com o uso de serviços flexíveis de infraestrutura elástica prestados por fornecedores de *CloudComputing* como é a *Amazon* que oferece os muitos recursos computacionais como computação, rede e armazenamento para os usuários.

CloudComputing é um modelo computacional em que a infraestrutura e aplicativos são fornecidos como serviços para os usuários finais sob um modelo de pagamento pelo uso. Existem alguns desafios na computação em nuvem como avaliar o desempenho de aplicações, políticas de aprovisionamento e escalonamento automático de aplicações.

Abaixo mostram-se os vários pontos que definem em detalhes o significado da tecnologia *CloudComputing*.

1.2.1. CARATERÍSTICAS PRINCIPAIS

A computação em nuvem está composta por quatro características principais que fazem dele um modelo de computação diferenciado, são:

- Aplicação de auto-serviço: (CONSEGUI 2009a) O consumidor de um serviço de nuvem pode, unilateralmente, providenciar recursos computacionais de forma automática (normalmente usando-se de interfaces de programação ou APIs), um recurso computacional; como um servidor virtual ou de espaço em um servidor de armazenamento em rede, de acordo com sua necessidade, sem necessidade de interação humana com o fornecedor do serviço.

- Elasticidade: O usuário do serviço pode solicitar quando precise de mais ou menos recursos para sua aplicação para se adaptar à demanda dos seus usuários. As capacidades podem ser configuradas de forma rápida e elástica, em alguns casos automaticamente, aumentando ou diminuindo o consumo de recursos de modo que parecem ilimitados. Um exemplo pode ser quando em períodos de pico o usuário solicita à nuvem mais recursos computacionais,

podendo depois liberar tais recursos quando os mesmos não forem mais necessários.

- Modelo de negócio**,** pagamento pelo Uso e Garantias de serviço (*Service Level Agreements – SLAs*). Os usuários do serviço pagam aos provedores só pelo consumo efetuado. Um fator importante é que a nuvem possui a propriedade de poder hospedar de forma eficiente aos usuários e pode compartilhar os recursos sem dificuldades.

- Acesso ubíquo através da rede. Os recursos computacionais são disponibilizados pela rede e podem ser acessados através de padrões (como *API*s *REST* baseadas em HTTP ou SOAP) por diversos tipos de clientes heterogêneos (*browsers*, telefones celulares, laptops e *PDA*s).

Depois da explanação das características e ter uma idéia geral do que é o modelo de computação em nuvem pode-se entender que esse modelo traz um conjunto de vantagens (BLOGFR 2009a), as mais importantes são explicadas a seguir:

- Integração de serviços Web. A tecnologia da computação em nuvem pode ser integrada com maior facilidade e velocidade com o resto das suas aplicações de negócio (o software tradicional e também o software *Cloud Computing* baseado em infraestrutura).

- Prestação de serviços em todo o mundo. As infraestruturas *Cloud Computing* proporcionam uma maior adaptação, recuperação de desastres de forma completa e fornece uma redução até o mínimo dos tempos de inatividade.

- A computação em nuvem não precisa instalar nenhum hardware. O que faz importante a computação em nuvem é sua simplicidade, isso exige um investimento menor em infraestrutura para colocar o negócio em operação. .

- Contribui para a utilização eficiente da energia. Neste caso, a energia necessária para funcionar a infraestrutura. Nos *Datacenters* tradicionais, os servidores consomem muito mais energia do que realmente precisam. Em contraste, nas nuvens, a energia consumida é apenas a necessária, reduzindo significativamente o desperdício.

Mas suas características também provocam desvantagens:

- Um erro humano ou uma catástrofe natural no provedor de nuvem pode ocasionar a interrupção de serviço de diversas empresas se o provedor estiver geograficamente replicado.

- As disponibilidades das aplicações estão ligadas à disponibilidade de acesso à Internet.

- Vulnerabilidade dos dados importantes das empresas, já que estes residem na nuvem. Tem possibilidade de uso indevido por parte de "*hackers"*.

- Escalabilidade em longo prazo. Quanto mais usuários comecem a compartilhar a infraestrutura da nuvem, a sobrecarrega vai aumentar, se a empresa não tem um padrão de crescimento ideal pode levar à degradação do serviço.

1.2.2. TIPOS DE NUVENS

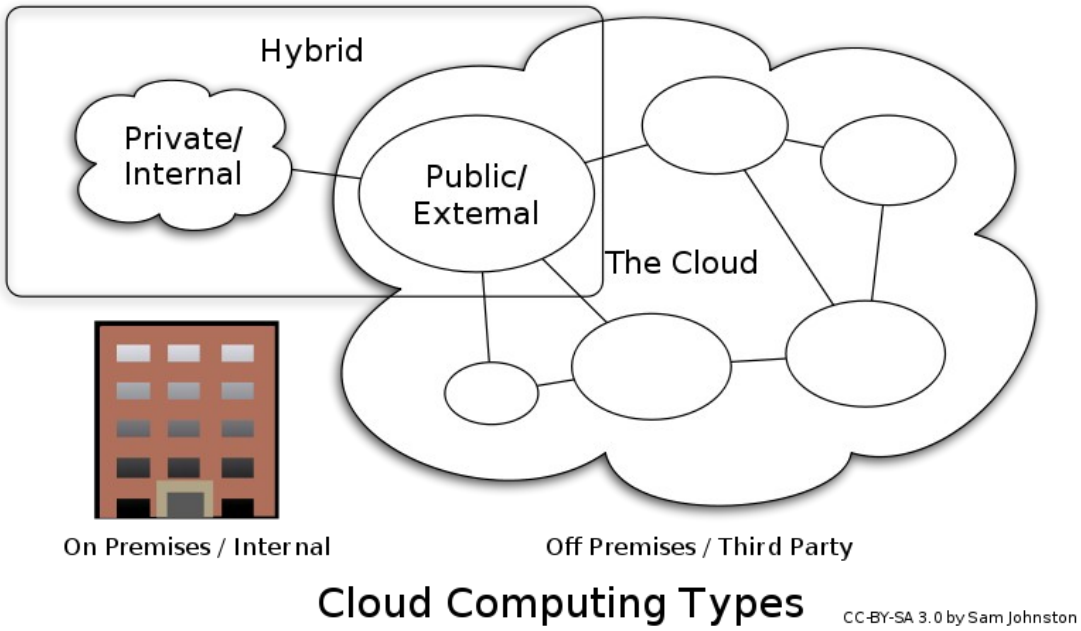
As nuvens podem ser caracterizadas nos seguintes tipos (Figura 2)

- Nuvem privada**:** A infraestrutura de nuvem pertence (VMWARE 2011) completamente a uma organização. Pode ser gerenciada pela própria organização ou por terceiros. Pode estar fisicamente na organização ou fora dela. As nuvens privadas são uma boa opção para empresas que necessitam de alta proteção de dados. Possuem os recursos físicos (servidores, armazenamento etc.) e podem decidir quais usuários estão autorizados a utilizar a infraestrutura.

- Nuvem Comunitária: é compartilhada por diversas organizações onde uma delas tem o poder de controlar as políticas de compartilhamento, segurança e cumprimento. Pode ser gerida por uma organização interna ou terceirizar.

- Nuvem Pública: Infraestrutura de nuvem disponível para o público em geral, pertence a uma grande organização que vende serviços da nuvem e é controlada por terceiros (geralmente o provedor da nuvem). (SUNMICROSYSTEMS 2009) Os trabalhos de muitos usuários podem ser misturados em diferentes servidores, sistemas de armazenamento e outras infraestruturas na nuvem. Os usuários finais não sabem o que outros clientes podem estar executando no mesmo servidor.

- Nuvem Híbrida: (UNIV. OF CALIFORNIA AT BERKELEY 2009) As nuvens híbridas misturam os modelos das nuvens públicas e privadas independentes. Essas nuvens são conectadas por tecnologia proprietária ou padronizada e permite interoperabilidade de dados e aplicações. As nuvens híbridas oferecem a promessa de escalonamento de aplicações provisionados externamente, sob demanda, mas adicionam a complexidade de determinar como distribuir os aplicativos através destes ambientes diferentes (SUNMICROSYSTEMS 2009). Uma boa definição pode ser o uso conjunto do hardware físico e instâncias de nuvem virtualizadas para conseguir único serviço comum.



http://en.wikipedia.org/wiki/Cloud_computing

1.2.3. TIPOS DE SERVIÇOS OFERECIDOS

Os tipos de serviços oferecidos pelas diferentes tipos de nuvens se dividem em três categorias segundo o tipo de serviço que elas dão aos usuários (ver figura 3).

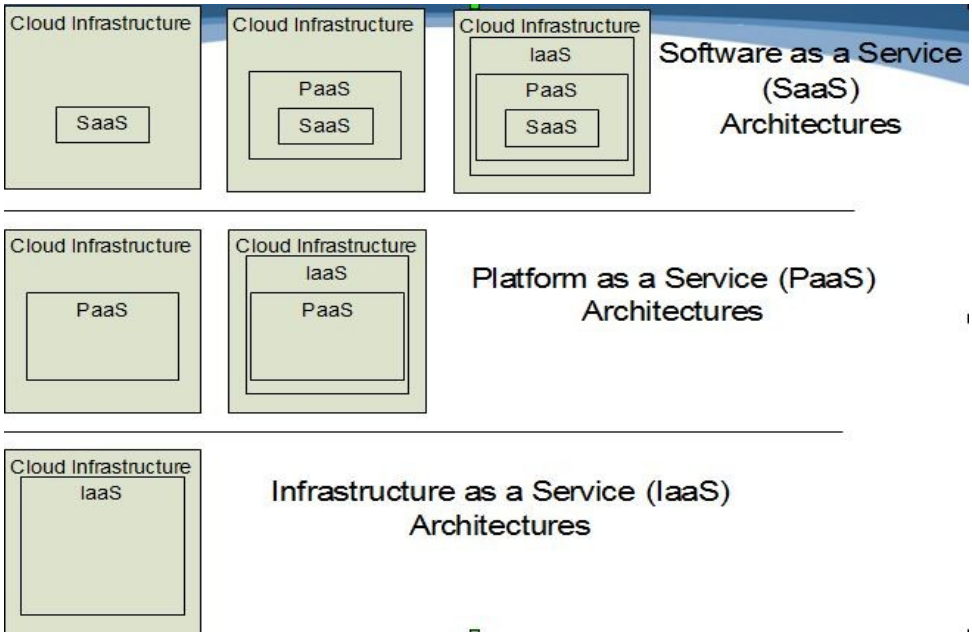
- Software como serviço, (em inglês *software as a service*, **SaaS**) (UNIV. OF CALIFORNIA AT BERKELEY 2009), é oferecido aos consumidores de uma nuvem do provedor, acessível por múltiplos dispositivos de clientes através de uma interface como um navegador web. O consumidor não controla a infraestrutura, apenas as configurações de usuário da aplicação. Uma das características mais importantes é que não precisa da instalação do software no servidor local. SaaS está no nível mais alto da camada da computação em nuvem e oferece uma aplicação completa prestada como um serviço, sob demanda, via multitendência - o que significa que apenas uma instância do software que se executa na infra-estrutura do provedor e serve para várias organizações de clientes-. O exemplo de SaaS mais conhecido é *Salesforce.com*, podendo incluir também o *Google Apps,* que prestam serviços básicos de negócio, tais como e-mail.

- Plataforma como serviço, (em inglês *platform as a service*, **PaaS**) (TECHNOLOGY NEWS 2009) está no nível intermediário da camada da computação em nuvem e oferece ao consumidor as maneiras de publicar as aplicações geradas ou adquiridas por ele, em infraestrutura do provedor de nuvem. Essas são criadas por linguagens de programação e ferramentas de apoio. O consumidor não gerencia a

infraestrutura, mas tem controle sobre os aplicativos e configurações do ambiente. As ofertas PaaS podem atender a todas as fases do ciclo de desenvolvimento e testes do software, ou podem ser especializadas em qualquer área específica, tais como o gerenciamento do conteúdo.
Os exemplos comerciais incluem o *Google App Engine*, que atende aplicações de infraestrutura do *Google*. Serviços PaaS como estes permitem uma grande flexibilidade, mas podem ser limitados pelas capacidades que estão disponíveis através do provedor.

- Infraestrutura como serviço, (em inglês infrastruture as a service, **IaaS**) (SUN 2009) está no nível inferior da camada de computação em nuvem e é um meio de entrega de serviços básicos como computação (máquinas virtuais) e armazenamento. São oferecidos aos consumidores formas de provisionar espaço em disco, redes e outros recursos essenciais, para que o consumidor possa configurar seus aplicativos. O consumidor não gerencia a infraestrutura da nuvem, mas tem o controle dos recursos provisionados, incluindo algumas configurações de componentes de rede (*firewalls*). O exemplo mais conhecido comercial é a *Amazon Web Services*, EC2, S3 serviços que oferecem serviços de computação e armazenamento essenciais (respectivamente). Esses dois depois vão ser explicados posteriormente.

A Figura 3 da uma visão geral das camadas da nuvem.



m. Fonte: Presentation on Effectively and Securely Using the Cloud Computing Paradigm v26, http://csrc.nist.gov/groups/SNS/cloud-computing/index.html

O estudo de pesquisa centra-se em na simulação da infrastrutura como serviço, exatamente da empresa Amazon, por isso é necessário dar uma explicada da empresa, com os detalhes pertinentes.

2. INFRASTRUTURA COMO SERVIÇO: AMAZON

Cada vez estão surgindo mais fornecedores de serviços de computação em nuvem. Alguns desses serviços seguem as definições apresentadas antes. Esses serviços caracterizam-se pelo uso pago de recursos, tais como processamento, armazenamento de dados e transferência de dados na net.

A seguir, será apresentado um dos mais conhecidos fornecedores de infra-estrutura como serviço. Especialmente se faz ênfase na explicação de alguns serviços da *Amazon* como EC2, S3 e EBS.

Apesar de ter seu foco em comércio eletrônico, a *Amazon* desde 2006, começou a oferecer serviços que possibilitavam utilizar os recursos computacionais de seus *Datacenters*. Isso aconteceu devido ao fato da *Amazon* ter adquirido uma infraestrutura muito poderosa para garantir que a demanda gerada em períodos de picos de vendas do seu site seria atendida. Nos momentos de períodos de baixa demanda, muitos recursos eram inativos; por isso a *Amazon* optou por vender a capacidade computacional excedente.

Essa empresa que foi uma das pioneiras na tecnologia de computação em nuvem, atualmente possui vários serviços relacionados com esta tecnologia, os chamados AWS (*Amazon Web Services*) (AMAZON, 2009a). Esses serviços que são oferecidos pela internet, como seu nome sugere, são oferecidos de maneira que o usuário paga apenas pelos recursos usados.

A seguir serão explicados os três principais serviços de *Amazon*, EC2, S3 e EBS.

2.1. AMAZON ELASTIC COMPUTE CLOUD (EC2)

Amazon Elastic Compute Cloud (*Amazon EC2*) é um serviço web que oferece capacidade computacional redimensionável na nuvem. *Amazon EC2* (AMAZON, 2009b) permite aos usuários alugar computadores virtuais nos quais podem executar suas próprias aplicações fornecendo um controle completo sobre seus recursos computacionais e permitindo a execução no ambiente computacional. As aplicações dos usuários são configuradas através de uma imagem *Amazon Machine Image* (imagem de máquina de *Amazon*) também conhecida como instância.

Cada usuário pode criar, executar e encerrar todas as instancias que precisa-se, pagando pelo uso. *Amazon* muda a economia da computação permitindo pagar pela capacidade utilizada, por isso o nome “Elastic”. EC2 provisiona usuários com controle sobre a localização geográfica das instâncias as quais permitem otimização de latência e diferentes níveis de redundância. Por exemplo, para garantir um tempo mínimo de download de certa aplicação, um usuário pode criar instâncias de servidor em múltiples zonas.

As instâncias são gerenciadas via *web services*, o que permite gerenciar todo o ciclo de vida delas. São divididas em 5 categorias de acordo com as capacidades de memória e processamento: *Standard Instances, Micro Instances, High-Memory Instances, High-CPU Instances, Cluster Compute Instances*.

As AMI permitem escolher seu próprio sistema operacional. Os OS que atualmente *Amazon EC2* oferece são: *Linux, OpenSolaris, Windows e Debian (GNU/Linux).*

Os passos para a correta utilização de EC2 podem-se centrar em:

(1) O usuário escolhe alguma AMI pré-configurada ou cria uma do zero.

(2) Seguidamente o usuário pode carregar a AMI no serviço de armazenamento da *Amazon*, o S3 (será explicado na seguinte seção). Essa AMI poderá ser para uso público ou apenas para o próprio usuário.

(3) Personalizar o tipo de segurança (ex: regras de firewall) e configurar o acesso a rede.

(4) Fazer uma escolha das instâncias e sistemas operacionais da Região e da Zona de disponibilidade em que a AMI irá executar.

(5) colocar a AMI em operação, ou seja, fazer o “start up” da instância.

O usuário pode configurar o tipo de endereço de rede da instância. A escolha é entre Elastic IP (um IP fixo vinculado a uma conta na *Amazon*) ou um IP aleatório adquirido através de um servidor DHCP da *Amazon*, podendo associar o IP a qualquer instância que lhe pertença a qualquer momento.

2.1.1. TIPOS DE INSTÂNCIAS

As instâncias Standard são uma família (*AMAZON*, 2009b) as quais são adequadas para a maioria das aplicações e são as mais comumente utilizadas pelos usuários de *Amazon*.

Tabela 1 – Características de Instâncias do EC2

Tipo Instância	Memória (GB)	EC2 Compute Unit[1]	Armazenamento (GB)	Plataforma
Small	1,7	1	160	32
Large	7,5	4	850	64
Extra Large	15	8	1690	64
Micro	613 (MB)	2	EBS	32 ou 64
High-Memory Extra Large	17,1	6,5	420	64
High-Memory Double Extra Large	34,2	13	850	64
High-Memory Quadruple Extra Large	68,4	26	1690	64
High-CPU medium	1,7	5	350	32
High-CPU Extra Large	7	20	1690	64
Cloud Computer Quadruple	23	33,5	1690	64

a Tabela 2 mostra outras instâncias da *Amazon EC2*. Estão divididas em 4 tipos: as micro instâncias, as instâncias de Alta-Memória, as instâncias de alta CPU e instâncias de *Compute Cluster*.

As micro instâncias são adequadas para aplicações com baixo desempenho e sites que usam periodicamente ciclos de computação significativos, já que querem fornecer uma pequena quantidade de recursos da CPU e explorar ao máximo a capacidade da CPU.

As instâncias de *High-Memory* oferecem uma grande memória para aplicações de alto desempenho, incluindo aplicações e banco de dados e caché.

Instâncias *High-CPU*, oferecem, proporcionalmente, mais recursos da CPU do que a memória (RAM) e são ideais para aplicações que exigem muitos recursos computacionais.

Instâncias *Compute Cluster*, oferecem um processador de alto desempenho com aplicações de alto desempenho da rede e são adequados para HPC (High Performance Compute) e outras aplicações de rede mais exigentes.

2.1.2. TIPOS DE CONTRATO

Uma das características de *Amazon EC2* é que permite escolher entre 3 tipos de pagamentos:

- On-Demand Instances: em português instâncias sob demanda. Esse tipo permite pagar um preço fixo pela capacidade de computação utilizada por hora.

- Reserved Instances: em português instâncias reservadas. Com essa opção o usuário pode reservar uma instância (fazendo um único pagamento reduzido para cada instância que se deseja reservar), podendo optar pela execução da instância pela taxa horária reduzida no seu prazo (desconto significativo durante uma taxa horária) ou pode optar por não pagar taxa de utilização (enquanto não estiver em uso a instância).

- Spot Instances: *Amazon EC2* oferece a possibilidade de poder alugar a capacidade que não está sendo utilizada no momento. O usuário tem a oportunidade de comprar instâncias quando o preço delas está nas suas expectativas já que o preço das instâncias muda periodicamente de acordo com oferta e demanda. Nesse momento, os clientes cujas taxas igualem ou superem o preço atual obtém o acesso a *Spot Instances* (instâncias disponíveis).

2.2. AMAZON S3

Amazon S3 (AMAZON 2009c) é a parte de *Amazon* que encarrega-se do armazenamento. Esse armazenamento é ilimitado é pode se realizar através da web, oferecendo como armazenar e recuperar a quantidade de dados que o usuário quiser. S3 é pode ser acessado desde qualquer interface de serviço web e tem a capacidade de escalabilidade. S3 é usado pela própria *Amazon* para ter em funcionamento sua própria rede internacional de web sites. Foi disponibilizado em março de 2006 e facilita aos desenvolvedores a computação em escala web.

Com esse serviço web permite aos desenvolvedores criar facilmente aplicações que tenham uso do armazenamento na Internet. Como *Amazon S3* é altamente escalável e só paga-se pelo uso, os desenvolvedores podem começar fazendo um sistema reduzido e aumentar a aplicação deles como eles quiseram.

O tamanho dos objetos individuais pode oscilar entre 1 byte e 5 gigabytes, e o número de objetos que pode-se armazenar é ilimitado. Pode-se armazenar todo tipo de dados em qualquer formato. O *S3* armazena os dados do usuário e só realiza um seguimento do uso para a fatoração, mas os dados serão protegidos e privatizados, com a exceção de quando a lei o exija. Incluem-se mecanismos de autenticação desenhados para garantir que os dados estão seguros em caso de acessos não autorizados. Cada objeto é armazenado em um depósito (bucket), podendo-se recuperar com uma chave

exclusiva designada pelo desenvolvedor. Um depósito pode ser armazenado em uma de varias Regiões. O usuário tem que escolher uma Região próxima para otimizar a latência e minimizar os custos.

2.3. AMAZON ELASTIC BLOCK STORE (EBS)

Amazon Elastic Block Store oferece armazenamento em bloco para instâncias de Amazon EC2(AMAZON 2009d). Ele oferece um armazenamento fora da instância, tendo independência da vida da instância. EBS contem volumes, partições de memória de armazenamento para hospedar as instâncias. EBS oferece também a possibilidade de criar copias de segurança dos volumes num momento determinado, que permanecem no Amazon S3. Essas cópias podem ser utilizadas como base para a criação de novos volumes do EBS e para proteger os dados.

Os volumes criam-se em uma Zona de Disponibilidade específica e podem ser utilizados por instâncias da mesma Zona de Disponibilidade, tendo um tamanho entre 1GB e um 1T. Quando o volume é montado em uma instância ele se assemelha a uma unidade de disco. Isso permite que a instância possa interatuar com o volume podendo formatar ele com um sistema de arquivos ou instalar aplicações nele.

Um volume só pode ser utilizado em uma única instância, mas uma instância pode ter vários volumes. Isso significa que a instância pode distribuir os dados entre os vários volumes para aumentar o rendimento e a velocidade. Pode ser útil para aplicações de que envolvem um processamento de dados em larga escala. Cada volume de armazenamento replica-se automaticamente dentro da mesma Zona de Disponibilidade. Isto evita uma possível perda de dados devido a uma falha de um componente de *hardware*.

Depois de dar uma idéia geral do que é a empresa Amazon e dos serviços de infrastrutura como serviço, vai se dar uma idéia geral do funcionamento do simulador CloudSim, que foi o simulador escolhido para este projeto.

3. SIMULAÇÃO DA NUVEM: CLOUDSIM

Alguns dos tradicionais e dos novos serviços de aplicativos baseados em nuvem incluem redes sociais, web hosting, e gerência de conteúdo. Cada um desses tipos de aplicações tem diferentes composições, configurações e requisitos de implantação.

É uma tarefa difícil medir o desempenho de aprovisionamento de uma aplicação em estruturas reais como a *Amazon devido a fatores como:*

- Não se pode ter controle sobre a infraestrutura da nuvem onde as aplicações executam.
- Fica difícil medir, testar e avaliar parâmetros de configuração sem ter controle das políticas de alocação de recursos da infraestrutura.

Isso faz com que a reprodução de resultados possa ser extremamente difíceis de conseguir. Além disso, configurar parâmetros de avaliação dos testes em uma infraestrutura de grande tamanho como é uma nuvem escalável pode ser demorado e custoso. Tais limitações são causadas porque os ambientes baseados em nuvem não estão no controle de desenvolvedores de serviços de aplicativos. Assim, fica difícil realizar certos tipos de experimentos em ambientes reais de computação em nuvem.

Uma das alternativas mais viáveis é a utilização de ferramentas de simulação porque permite testar de forma confiável e com um custo muito pequeno. Estas ferramentas permitem a possibilidade de avaliar o estudo comparativo da aplicação em um ambiente controlado, onde pode-se facilmente reproduzir os resultados, permitindo as empresas de IT:

- Testar em ambiente repetível e controlável dos seus serviços de teste em ambiente.
- Descobrir os gargalos do sistema e tentar ajustar eles antes de implantar as nuvens reais.
- Possibilidade de desenvolvimento e testes de técnicas de aplicações adaptáveis de aprovisionamento devido a que se tem uma experiência continua com a mistura de diferente carga de trabalho de e desempenho dos recursos em cenários simulados.

3.1. VISÃO GERAL DE CLOUDSIM

Atualmente existem poucos simuladores de ambientes de computação em nuvem. Um desses poucos é o *CloudSim*.

CloudSim oferece uma nova estrutura de simulação, a qual é generalizada e extensível e permite uma experimentação, modelagem e simulação de infraestruturas e serviços de aplicações de computação em nuvem. Uma peça chave de *CloudSim* é que permite aos desenvolvedores e pesquisadores testar o desempenho que pode ser controlado e configurado com facilidade. As principais características (Calheiros et al 2010) que oferece nesse ambiente são:

- Eficácia: exige-se muito menos tempo e esforço para implementação do teste.
- Flexibilidade e Aplicabilidade: os desenvolvedores de modelo podem testar com pouco esforço de programação.
- Suporta a modelagem e simulação em grande escala de ambientes de computação em nuvem, incluindo os *Datacenters*, na computação em um único nodo físico, tanto como a simulação de conexões de rede entre os elementos do sistema simulado.
- Possui plataformas independentes de modelagem de nuvens, corretores de serviços, fornecimento e atribuições de políticas.
- Facilidades para a simulação de federações de nuvem.
- Mecanismo de virtualização que focaliza-se na criação e gerenciamento de múltiplos, independentes serviços de *Datacenter*.
- A flexibilidade para mudar entre as políticas de atribuição de *space-shared* e *time-shared* (que serão explicados a seguir) de processamento de núcleos de serviços virtualizados.

3.1.1. MODELO DA NUVEM

Um componente de *CloudSim* pode ser uma classe (abstrata ou completa), ou um conjunto de classes que representam um modelo CloudSim (*Datacenter, Host, etc.*).

A entidade *DataCenter* (Calheiros et al. 2010) gerencia o número de entidades *Hosts*. Um *Datacenter* pode gerenciar vários *Hosts*, os quais gerenciam as VMs durante os ciclos de vida delas.

Os *Hosts* são designados para um ou mais VMs, baseadas na atribuição de políticas (operações tais como o aprovisionamento de um *Host* pra um VM, criação,

destruição e migração de VM) que devem ser definidas pelos provedores de serviços de Nuvem.

Um *Host* é um componente *CloudSim* que representa um servidor físico em uma nuvem. Ele é designado para pré-configurar a capacidade de processamento dos núcleos e as VMs.

A atribuição de uma VM é o processo de criação de instâncias VM nos *hosts*, que definem as essenciais características (armazenamento, memória), configurações (parâmetros do software) e requerimentos dos provedores de IaaS. *CloudSim* suporta o desenvolvimento de modelos de serviços de aplicações personalizadas que podem ser usadas com uma instância VM e os usuários realizam suas demandas as VMs em blocos de informação chamados *Cloudlets*.

A responsável de atribuir o *Host* mais apropriado a uma VM é a chamada *VmAllocationPolicy*. Os desenvolvedores e pesquisadores podem implementar novas políticas de alocação de acordo com sua necessidade.

A *VmAllocationPolicy* implementa uma política simples nas VMs para o *Host* no tipo *FCFS* (primeiro que entra, primeiro que é servido).

Outras políticas, incluindo as feitas pelos provedores da Nuvem podem ser facilmente simuladas e modeladas em CloudSim. Mas as políticas usadas pelos provedores de nuvem públicos, como é *Amazon EC2*, não são disponíveis publicamente.

Para cada componente *Host*, a atribuição de núcleos de processamento para as VMs se faz baseando em uma política de atribuição de *Host* que leva em conta várias características de hardware como o número de núcleos, o compartilhamento de CPU e o conjunto de memória que vão ser tomadas pela instância VM. CloudSim suporta vários cenários que designam determinados núcleos para determinadas VMs (uma política *space-shared*) ou a política que distribui dinamicamente a capacidade de um núcleo para as VMs (política *time-shared*); e designa núcleos para as VMs quando é necessário.

Na política *space-shared* cada tarefa é atendida por um núcleo uma por uma até a finalização da mesma, enquanto que a política *time-shared* permite o atendimento em paralelo.

O *Host* configura essas políticas mediante o componente *VMSchedulerComponent*, Esse componente é onde pode-se implementar as políticas de *space-shared* ou *time-shared* para a atribuição de núcleos para as VMs, onde os pesquisadores e desenvolvedores de sistema poderão personalizar as políticas com o fim de otimizar.

3.1.2. MODELO A ATRIBUIÇÃO DE VMs.

Como já se comentou anteriormente, as nuvens se baseiam na tecnologia de virtualização.

O hardware de uma VM está limitado pelo total de poder de processamento e largura de banda disponível no *Host*. Esse fator é essencial e deve ser considerado durante o processo de aprovisionamento da VM, para evitar a criação de uma VM configurada com mais poder de processamento que permite-se em um *Host*.

Para permitir a simulação de diferentes políticas de aprovisionamento *CloudSim* provisiona a VM em dois níveis: o primeiro, o nível de *Host* e o segundo, o nível de VM. No nível de *Host*, é possível especificar o máximo de poder de processamento que cada VM vai possuir. No nível de VM, é possível especificar o poder de processamento máximo de cada aplicação de serviço individual (unidades de tarefa) que podem ser hospedadas, já que as VMs podem processar várias tarefas ao mesmo tempo dependendo do tipo de política de atribuição.

Nesse último nível, *CloudSim* implementa as políticas de provisionamento *time-shared* e *space-shared*. A Figura 4 mostra um exemplo onde um *Host* que possui dois núcleos deve alocar duas VMs que possuem 4 tarefas por VM.

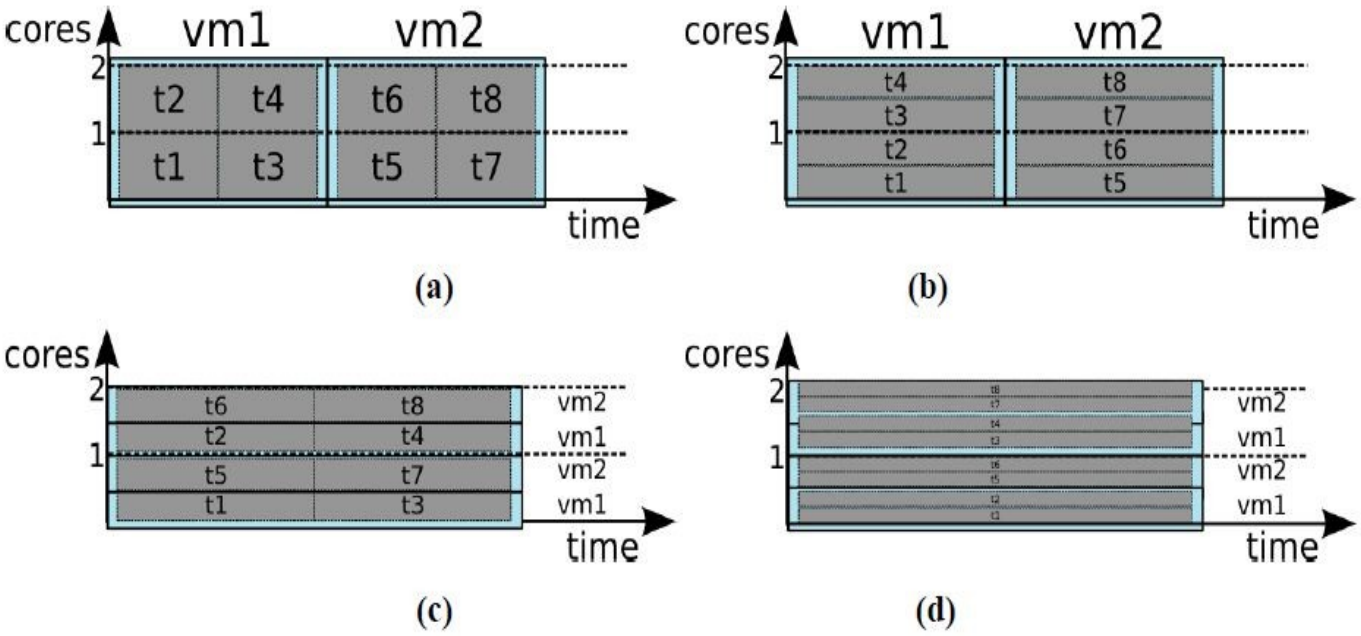


Figura 4. Exemplo de tipo de atribuição de VM. Fonte: (CALHEIROS et al 2010).

A Figura 4A apresenta o cenário de aprovisionamento, (CALHEIROS et al 2010) onde aplica-se a política de *space-shared* para as duas VMs e também nas tarefas. Como cada VM requisita dois núcleos, no modo *space-shared* só uma VM pode se- executar no mesmo momento. Portanto a VM2 só pode ser designada para um núcleo quando a VM1

termine a execução de suas tarefas. O mesmo acontece para as tarefas de VM1: desde que cada tarefa solicite um núcleo, só dois núcleos podem funcionar no mesmo momento. Durante esse período, as tarefas 3 e 4 esperam na fila de execução.

Na figura 4B, aplica-se uma política *space-shared* para a atribuição de VMs aos *Hosts* e política *time-shared* para as tarefas que vão ser processadas pelos núcleos. Enquanto o tempo de vida de uma VM, todas as tarefas atribuídas nessa são dinamicamente mudadas de contexto durante o ciclo de vida delas. Como pode se observar, os núcleos seguem a política *time-shared*, o que permite ter duas tarefas se processando ao mesmo tempo no mesmo núcleo. Como as VM continuam utilizando uma política tipo *space-shared*, até que a VM1 não finaliza seu processamento, a VM2 não pode começar.

Na figura 4C, usa-se uma política *time-shared* para as VMs, enquanto que as tarefas são atribuídas mediante uma política *space-shared*. Nesse caso, cada VM recebe um pedaço do tempo em cada núcleo de processamento, o qual depois distribui os pedaços entre as tarefas se baseando em *space-shared*. Os núcleos são compartilhados, com a totalidade do poder de processamento, permitindo variar de VM. Isso é determinado calculando as VMs que são ativas no *Host*. Como as tarefas são designadas se baseando na política *space-shared*, isto significa que em um instante de tempo só uma tarefa pode ser ativa usando o núcleo de processamento.

Finalmente, na figura 4D a atribuição de *time-shared* é aplicada nas duas VMs e tarefas. O poder de processamento é concorrentemente compartilhado pelas VMs e esses compartilhamentos de cada VM são simultaneamente divididos entre as tarefas. Nesse caso, não tem fila de espera associada com as tarefas.

Cada tipo de combinação dos tipos de política permite diferentes rendimentos de tempo e poder de computação. Dependendo das necessidades de um usuário e dos recursos da nuvem poderão se utilizar um tipo de combinação ou outro.

3.1.3. MODELO DA FEDERAÇÃO DE NUVENS

Para fazer possível a existência de federação de nuvens é básico uma entidade tipo *CloudCoordinator*, que será a responsável de comunicar os diferentes *DataCenters* de cada nuvem e usuários finais no entorno da simulação, e monitorizar e gerenciar o estado interno da entidade do *DataCenter*. A informação recebida como parte do processamento de monitorizarão é usada para tomar decisões relacionadas com o aprovisionamento do conjunto de nuvens. Os provedores de nuvem como *Amazon EC2*, não oferecem a entidade *CloudCoordinator*, o desenvolvedor tem que criar essa classe no caso de precisar simular um conjunto de nu*vens*.

Dois aspectos são fundamentais no contexto da federação de nuvens (CALHEIROS et al 2010): a comunicação e o monitoramento. O primeiro aspecto é solicitado pelo *Datacenter* no processo de mensagem padrão. O segundo aspecto é executado pelo *CloudCoordinator*. Para permitir o monitoramento on-line do *DataCenter*. Precisa-se de uma entidade chamada *Sensor*, a qual está vinculada com o *CloudCoordinator* e avisa ao mesmo de todos os eventos que acontecem em cada *Datacenter*. Em cada passo de monitoramento, o *CloudCoordinator* consulta o *Sensor*.

`

3.1.4. PROJETO E IMPLEMENTAÇÃO DE CLOUDSIM.

A seguir são descritas as principais classes da implementação do simulador *CloudSim* (Ver Figura 5).

.

- Cloudlet: Essa classe modela os serviços de aplicações baseados em nuvem (como conteúdo entregado, redes sociais, e fluxo de trabalho de negócios). *CloudSim* dirige a complexidade de um aplicativo em termos dos requisitos computacionais.

- CloudletScheduler: essa classe abstrata é estendida pela implementação de diferentes políticas que determinam o compartilhamento do poder de processamento entre as *Cloudlets* e uma VM. Como explicou-se anteriormente, tem dois tipos de políticas de aprovisionamento: *space-shared* (*CloudletSchedulerSpaceShared*) e *time-shared* (*CloudletSchedulerTimeShared*).

- DataCenter: essa classe modela os serviços de nível da infraestrutura do núcleo (hardware) que são oferecidos pelos provedores de Nuvem. Ele encapsula um conjunto de *Hosts* os quais podem ser homogêneos ou heterogêneos com respeito a suas configurações de hardware (memória, núcleo, capacidade e armazenamento). Além disso, cada componente do *DataCenter* instancia um componente de aprovisionamento de aplicativo generalizado que implementa um conjunto de políticas para a atribuição de largura de banda, memória e armazenamento dos *Hosts* e as VMs.

- DataCenterBroker ou Cloud Broker: Essa classe modela um *Broker*, quem é o responsável para mediar às negociações entre SaaS (o usuário do serviço de nuvem) e os provedores de Nuvem, os quais seguem uma política de requerimentos de QoS. O *Broker* atua em nome do usuário. Ele é quem procura os provedores de serviço de Nuvem mais adequados, consultando o CIS (*Cloud Information Service*) e toma negociações on-line para a atribuição de recursos ou serviços que podem precisar os aplicativos de QoS. Os pesquisadores e programadores devem estender essa classe para avaliar e testar políticas personalizadas de *brokering*.

- DatacenterCharacteristics: essa classe contem informação da configuração dos recursos do *DataCenter*.

- Host: Essa classe modela o recurso físico como um servidor físico. Ele também encapsula informação importante como o conjunto de memória e armazenamento, uma lista e tipos de núcleos de processamento (pode representar também uma máquina tipo mono-núcleo ou multi-núcleo), a atribuição de política para o compartilhamento do poder de processamento entre as VMs, e políticas de aprovisionamento de memória e banda nas VMs.

- VM: essa classe modela uma máquina virtual, a qual é gerida e hospedada por um *Host*. Cada componente VM tem que acessar um componente o qual armazene as seguintes características relacionadas com a VM: memória acessível, processador, tamanho de armazenamento e a política de aprovisionamento interna que é estendida desde um componente abstrato chamado-se *CloudletScheduler*.

- VmmAllocationPolicy: Essa classe abstrata representa uma política de aprovisionamento que um Monitor de VM usa para atribuir VMs e Hosts. A principal funcionalidade essa classe é escolher o melhor *Host* dentro de um *DataCenter* que tenha a memória, armazenamento, e requerimentos que façam possível a execução.

- VmScheduler: essa classe abstrata implementada pelo *Host* modela as políticas (*space-shared*, *time-shared*) requisitadas para atribuir os núcleos de processamento as VMs. As funcionalidades dessa classe podem trocar para se acomodar as políticas de compartilhamento de processamento de aplicativos específicos.

- NetworkTopology: Essa classe contem a informação para induzir comportamento de rede (latências) na simulação. Ele armazena a informação topológica da rede, a qual é gerada usando o gerador de topologia BRITE.

- BwProvisioner: Essa classe abstrata modela a política para o aprovisionamento da largura de banda nas VMs. A função desse componente é controlar a atribuição de larguras de banda para o conjunto de VMs que são implementadas dentro de um *DataCenter*. Os desenvolvedores do sistema de Nuvem e pesquisadores podem estender essa classe com as políticas próprias deles (prioridade, QoS) para refletir as necessidades dos seus aplicativos O *BwProvisioningSimple* é uma política que permite uma VM reserve toda a largura de banda que ela precisar dentro do limite máximo que tiver nesse *Host*.

- RamProvisioner: Essa é a classe abstrata que representa a política de aprovisionamento para a atribuição de memória *RAM* nas VMs. A execução e a implementação da VM em um *Host* é possível só se o componente *RamProvisioner* aprova que o *Host* tem suficiente memória livre. A classe *RamProvisionerSimple* não

coloca restrição para a memória requisitada em uma VM. Caso o requerimento seja além da máxima capacidade de memória então é simplesmente rejeitado.

- SanStorage: essa classe modela a área de armazenamento de rede freqüentada nos *Datacenters* para armazenar grandes pedaços de dados (como no *Amazon* S3). SanStorage implementa uma interface simples que pode ser usada para simular armazenamento e recuperação de qualquer tipo de dado, sujeito a os limites de largura de banda.

- CloudCoordinator: Essa classe abstrata estende o *DataCenter* pra a federação. Ela é a responsável de monitorizar periodicamente o estado interno dos recursos dos *DataCenters* e tomar dinamicamente decisões de pedaços de carga de solicitações (load-shredding). Uma implementação concreta desse componente inclui uns sensores específicos e uma política que deveria ser seguida durante a realização de requisições. O monitoramento dos recursos do DataCenter é controlado pelo método *updateDatacenter( )* o qual envia consulta aos sensores.

- Sensor: Essa interface deve ser implementada pa*ra gerar u*m componente tipo sensor, que pode ser usado pela *CloudCoordinator* para monitorar parâmetros de exe*cução espec*íficos como são a energia de consumo e recursos de utilização. Além disso, *CloudCoordinator* usa a informação especificas de monitoramento para controlar o balanceamento de cargas.

A Figura 5 apresenta as diferentes classes descritas anteriormente e a relação entre elas. O *Host* recebe informação das classes *BwProvisioner, RamProvisioner* e *VmScheduler* para poder configurar a VM desejada. A VM recebe a configuração por um lado do *Host* e por outro pela classe *CloudletScheduler*, a qual permite escolher o tipo de política de atribuição da Vm. Por outro lado, tem o *Datacenter* o qual recebe a configuração do *VmAllocationPolicy*, *DatacenterCharacteristics* e (no caso de ter uma Federação de nuvens), do FederatedDatacenter para ser totalmente configurado. As classes *NetworkTopology*, *SANStorage* e *Cloudlet* são configuradas independentemente e não têm vinculo com as outras classes.

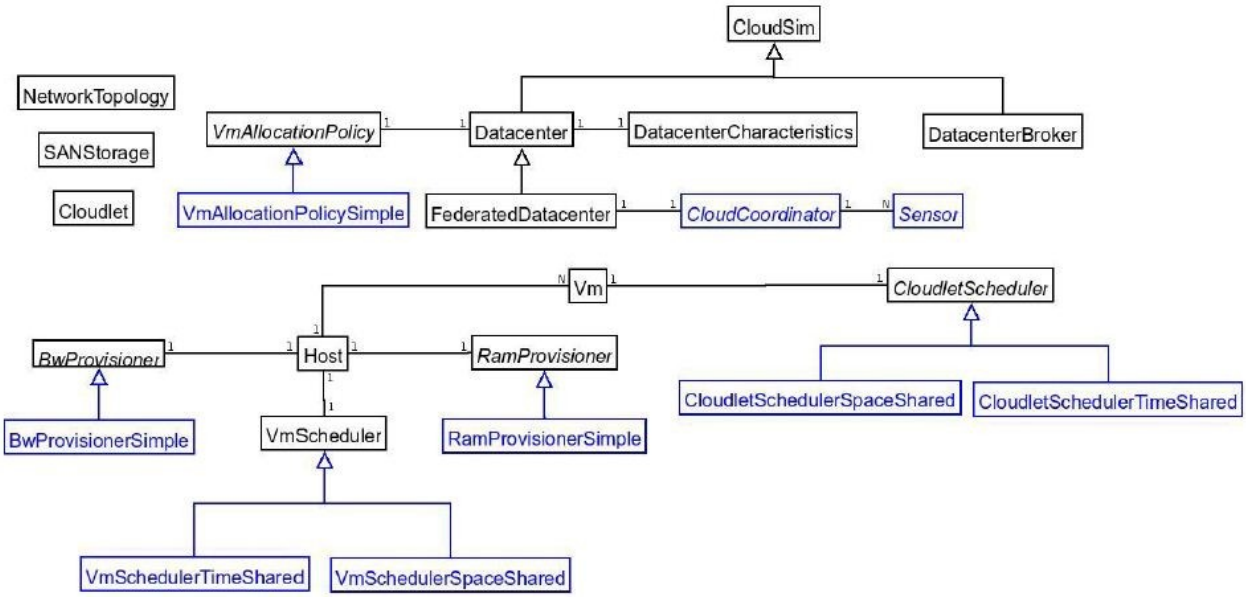
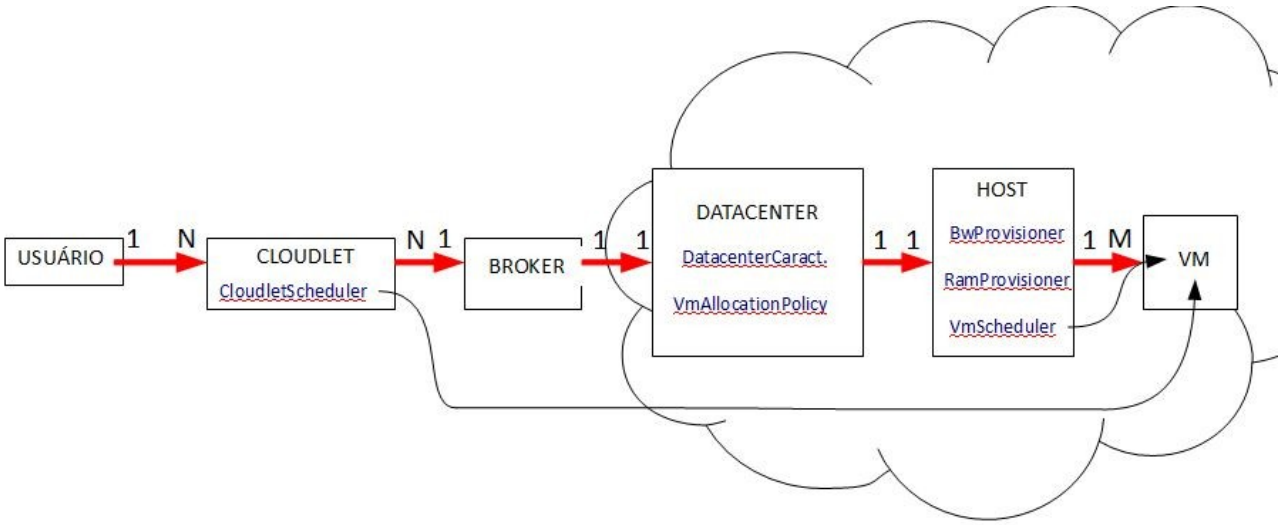


Figura 5. Diagrama de classes CloudSim. Fonte: (**CALHEIROS et al 2010)**

3.2. EXEMPLO DE REQUISIÇÃO DE PROCESSAMENTO

Um exemplo simples pode ser um ambiente no qual tem um usuário com uma aplicação (*Cloudlets*) que precisa ser processada por uma nuvem simples.



(CALHEIROS et al 2010)

O primeiro passo é encapsular a informação que será processada automaticamente em uma *Cloudlet* (ou em N *Cloudlets* se for necessário). Seguindo as configurações de política de atribuição previamente designadas pela simulação (como podem ser as políticas *Space-Shared*, *Time-Shared* ou outras políticas personalizadas pelos desenvolvedores) as quais vão chegar até a VM, mediante a classe *CloudletScheduler*. Seguidamente, o *Broker* (cada usuário tem 1 *Broker*) será o encarregado de procurar as melhores condições de processamento dentro da nuvem. Ele negocia o QoS e as configurações, requisitos e serviços nos quais o *Cloudlet* precisa ser processado, buscando o *Datacenter* que mais adapte-se a essas necessidades da nuvem (achando essas características na classe *DatacenterCaracteristics* que pertence aos *Datacenters*).

O *Datacenter* aceita a entrada do *CloudLet* e procura um *Host* livre com as capacidades precisadas pelo *CloudLet* se for possível seguindo o método *VmmAllocationPolicy,* o qual tem como função procurar o melhor *Host* para a *Cloudlet.* Uma vez escolhido o *Host,* ele vai configurar o número de VMs necessárias para processar o *CloudLet*, segundo as características especificadas pelo usuário sempre que seja possível, tais como a configuração de Largura de Banda (seguindo a classe *BwProvisioner*), a memória RAM (com a classe *RamProvisioner*), e política que vão seguir os núcleos da própria VM (com a classe *VmScheduler*) e sabendo o tipo de política que

deve seguir a *CloudLet* que chegou até ela (com a classe *CloudLetScheduler*). Depois disso, a VM pode processar a informação e retornar os resultados ao usuário quando o processamento terminar.

3.3. EXEMPLO DE EXECUÇÃO DE CLOUDSIM.

Será apresentado um exemplo simples que mostra o funcionamento de *CloudSim*.

Nesse exemplo parte da simulação de um *Cloudlet* processado em um *Datacenter*. Para isso será necessário criar um *Broker*, um *Datacenter*, um *Host* e uma VM para atender a solicitação de processamento. Nesse exemplo só precisa-se de um *Datacenter*, um *Host* e uma VM porque suponha-se que o volume de dados do *Cloudlet* é suficiente com uma única VM.

Initialising...

Starting CloudSim version 2.0

Datacenter_0 is starting...

Broker is starting...

Entities started.

0.0: Broker: Cloud Resource List received with 1 resource(s)

0.0: Broker: Trying to Create VM #0 in Datacenter_0

0.0: Broker: VM #0 has been created in Datacenter #2, Host #0

0.0: Broker: Sending cloudlet 0 to VM #0

9000.0: Broker: Cloudlet 0 received

9000.0: Broker: All Cloudlets executed. Finishing...

9000.0: Broker: Destroying VM #0

Broker is shutting down...

Simulation: No more future events

CloudInformationService: Notify all CloudSim entities for shutting down.

Datacenter_0 is shutting down...

Broker is shutting down...

Simulation completed.

Simulation completed.

========== OUTPUT ==========

Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish

```
 Time
   0      SUCCESS      2          0      9000       0      9000
*****PowerDatacenter: Datacenter_0*****
User id        Debt
3              15,12
********************************
CloudSimExample1 finished!
```

Como pode-se observar na tela primeiramente cria-se um *Datacenter* onde será escolhido o *Host* e seguidamente cria-se um *Broker* o qual mediará à comunicação com a Cloud.

No inicio se inicializam o *Datacenter* e o *Broker* segundo as configurações previas feitas no código Java. Quando elas estão prontas, começa a requisição do usuário.

No milissegundo 0.00, o *Broker* percebe um recurso (*resource*), que é a *Cloudlet*. Nesse momento busca a *Datacenter_0*, a qual procura um *Host* o qual tenha o recursos necessários para processar a Cloudlet. Tenta-se criar uma VM com as condições estabelecidas pelo usuário. Finalmente, consegue-se criar a VM desejada, que nesse caso tem o nome de *VM #0 e* será hospedada no *Host #0*. Envia-se a *Cloudlet* até a *VM #0*.

A *Cloudlet* começa ser processada pela VM e no milissegundo 900.0 consegue-se receber e finalizar o processo de *Cloudlet*. Quando tudo acaba, o *Datacenter* e o *Broker* destroem-se e a simulação finaliza com sucesso.

Mostram-se seguidamente os resultados da simulação incluindo os registros das *Cloudlets* processadas, os *Datacenters* e VM utilizadas e o tempo de execução. Finalmente, se mostra a dívida que o usuário tem. Nesse caso, tem um custo de 15,12 dólares.

No próximo capítulo explicam-se as mudanças que se fizeram para conseguir simular o serviço de *Amazon, Amazon EC2*.

4. SIMULANDO UMA APLICAÇÃO NA AMAZON EC2

Nesse capítulo se da uma explicação da implementação feita para adaptar o simulador *CloudSim* às instâncias principais oferecidas pela empresa *Amazon EC2* (*Small Instace, Large Instance* e *Extra Large Instance*), assim como uma exposição das mudanças efetuadas no código Java do simulador para refletir com exatidão os diferentes custos que dão diversas combinações de instâncias que processam *Cloudlets*. Também se faz um estudo sobre os tipos de instâncias que são mais rentáveis segundo as necessidades do usuário em relação com o custo final e tempo de processamento.

4.1. MÉTODOS IMPLEMENTADOS

Nessa seção se comentam os distintos métodos implementados total ou parcialmente para adaptar o simulador *CloudSim* as instâncias (máquinas virtuais) da Amazon. Os métodos principais que permitem o desenvolvimento são: *createVM, createCloudlet, createDatacenter* e *costperSecond* Também se mostra o funcionamento da classe principal que contém o método *main()*.

4.1.1. MÉTODO *CREATEVM*

Esse método modela a criação das instâncias de Amazon EC2. Na figura 7 apresenta-se o código Java:

```
private static List<Vm> createVM(int userId, int vms,int size,int ram, int mips,
                                 int bw,int pesNumber,String vmm)
{
    if(vms<=20)
        {
//Cria uma lista onde colocam-se as VMs criadas.Essa lista se passará ao Broker mais tarde
LinkedList<Vm> list = new LinkedList<Vm>();



//create VMs
Vm[] vm = new Vm[vms];

    for(int i=0;i<vms;i++)
    {
        vm[i] =new Vm(i, userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());
        //deixo escrito a maneira para a política SpaceShared.
        //vm[i] = Vm(i, userId, mips, pesNumber, ram, bw, size, priority, vmm, new CloudletSchedulerSpace

        list.add(vm[i]);
    }
        }
else{
        Log.printLine("ERROR de configuração,máximo 20 VMS");
     }
    return list;
   }
```

Figura 7. Código fonte da classe createVM

Amazon EC2 autoriza criar no máximo 20 instâncias para um usuário (AMAZON 2011). Passam-se como parâmetros todas as características necessárias para a criação das instâncias, como a identificação do usuário, o número de instâncias, o tamanho de armazenamento e de RAM, as MIPS, a largura de banda e o número de processadores que cada VM tem.

Primeiramente cria-se uma lista onde se colocam as instâncias criadas e também se reserva o espaço de memória para todas as instâncias. Se o número de instâncias não supera 20, se geram as instâncias mediante o comando de Java *for*, chamando a função *Vm*, método já implementado pelo *CloudSim*. No caso contrario onde se supera as 20 VMs permitidas, envia-se uma mensagem de erro. Por último se devolve a lista com as instâncias criadas.

4.1.2. MÉTODO CREATECLOUDLET

Esse método foi desenvolvido para a criação das *Cloudlets* onde tem como parâmetros de entrada as características próprias da *Cloudlet* que são descritas posteriormente junto com o número de *Cloudlets* que tem que se gerar e a identificação do usuário.

Seguidamente se cria uma lista onde se colocam as *Cloudlets* que depois serão geradas. Se cria um *array* de elementos tipo *Cloudlet* (classe implementada pelo *CloudSim* que produz as *Cloudlets*) e mediante o comando *for* se criam todas as *Cloudlets*. Finalmente se devolve a lista com as *Cloudlets*. Na Figura 8 mostra-se o código Java do simulador.

```
private static List<Cloudlet> createCloudlet(int userId, int cloudlets,long length,
                                    long fileSize, long outputSize, int pesNumber)
{
            // Se cria uma lista de Cloudlets
            LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

                        UtilizationModel utilizationModel = new UtilizationModelFull();

            Cloudlet[] cloudlet = new Cloudlet[cloudlets];
            for(int i=0;i<cloudlets;i++){
                    cloudlet[i] = new Cloudlet(i, length, pesNumber, fileSize, outputSize,
                                    utilizationModel, utilizationModel, utilizationModel);
                    // se da o proprietario da Cloudlet
                    cloudlet[i].setUserId(userId);
                    list.add(cloudlet[i]);
            }

            return list;
    }
```

Figura 8. Código fonte da classe createCloudlet

4.1.3. MÉTODO CREATEDATACENTER

O método modela a geração do *Datacenter* e que precisa de uns parâmetros de entrada como são: o nome que o identificará, o número de instâncias que são solicitadas pelo usuário, assim como as características próprias das instâncias e das *Cloudlets* (que se comentam mais abaixo).

.

Para implementá-lo se requer um conjunto de passos que são explicados a seguir. Podemos ver um pedaço do código Java do método *createDatancenter* na Figura 9.

```
private static Datacenter createDatacenter(String name,int vm,int Ram, int Storage,int Bw,int Mips, long length,long fileSize,
                                    int size){
        // permite-se 20 VM no maximo (Amazon).
        // Aquí são descritos os passos necesários para a criação do Datacenter
        // 1.Cria-se uma ou mais listas onde armazenar as máquinas
        List<Host> hostList = new ArrayList<Host>();
        // 2. A maquina contem um ou mais unidades de processamento.
        //Cria-se uma lista onde armazenar esses processadores antes de criar a VM.

        List<Pe> peList1 = new ArrayList<Pe>();

        int mips = Mips*vm;
```

Figura 9. Código fonte da classe createDatacenter

(1) Guardar espaço de memória para as listas de *Hosts* e de processadores que são utilizados pelo *Datacenter*.

(2) Se criam a lista de processadores que se utilizam.

(3) Seguidamente se adicionam os núcleos dos processadores na lista. Cada tipo de instância tem diferente número de núcleos.

A *SmallInstance* utiliza um único núcleo de processamento e sua implementação no código é mostrada na figura 10:

```
// 3.Criam-se processadores e armazenam-se na lista deles

peList1.add(new Pe(0, new PeProvisionerSimple(mips))); //
```

Figura 10. Código fonte da geração de processadores

Pode-se observar que para a criação da lista de núcleos por processador, também é necessário colocar o identificador do núcleo e a política de provisionamento que seguirá o núcleo e a capacidade em MIPS que ele possui.

Enquanto a *LargeInstance* usa 2 processadores de 2 núcleos, unicamente se adiciona 1 núcleo mais na lista de núcleos por processador. O mesmo acontece com a *ExtraLargeInstance*, a qual possui 4 unidades de processamento com 2 núcleos cada um.

(4) Geração do Host que conterá as máquinas virtuais. Os parâmetros necessários para a criação são: o identificador de Host (já que pode podem se criar mais de um) a memória RAM, armazenamento e largura de banda máxima que o Host oferece. Se adiciona o Host na lista de Hosts chamando a função implícita no *CloudSim*, chamada *Host()*. Nela introduze-se os parâmetros mencionados e as políticas que seguiram a memória e a largura de banda. Também se colocam a lista de núcleos e a política de provisionamento das máquinas virtuais (*TimeShared* ou *SpaceShared*). Na figura 11 mostra-se que os parâmetros do *Host* tem uma política tipo *Simple*.

```
//4. Criaçao de Hosts, colocando seus próprios recursos.
int hostId=0;

int ram = Ram*vm; //Memoria RAM do host (MB)
long storage = Storage*vm; // Armazenamento do host (MB)
int bw = Bw*vm; // Largura de banda do host.

hostList.add(
        new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList1,
                new VmSchedulerTimeShared(peList1)
        )
);
```

Figura 11. Código fonte da geração de Hosts

Tem que se esclarecer que a memória, o armazenamento e a largura de banda são multiplicados pelo número de instâncias que são geradas para assegurar que o Host tem capacidade suficiente para abrigar todas as instâncias.

(5) O quinto passo é a geração da classe *DatacenterCharacteristics*, onde se armazenam as propriedades do *Datacenter*. Essa classe que precisa da arquitetura do sistema, o sistema operacional, a zona horária onde está situado o *DataCenter* e os custos pelo uso (ver figura 12).

```
// 5. Criação do DatacenterCharacteristics e armazenamento das propriedades de um Datacenter

        String arch = "x86";      // Arquitetura do sistema
        String os = "Linux";          // Sistema operacional
        String vmm = "Xen";
        double time_zone = 10.0;                    //Zona horária onde está situado o Datacenter
        double cost =0.085;                         // Custo por hora de uma instância Small
      //double cost =0.34;                          // Custo por hora de uma instância Large
      //double cost =0.68;                          // Custo por hora de uma instância ExtraLarge
        double costPerMem = 0.00;                   //Custo por RAM ( incluida no preço por hora)


        double costPerBw = 0.0;                 // the cost of using bw in this resource
        LinkedList<Storage> storageList = new LinkedList<Storage>();    //we are not adding SAN devices by now

        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
        arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);
```

Figura 12. Código fonte da geração DatacenterCharacteristics

(6) Finalmente se gera o *Datacenter* introduzindo os parâmetros desta classe na função já implementada *DatacenterCharacteristics().*

4.1.4. MÉTODO COSTPERSECOND

Esse método foi implementado para solucionar um problema visto no *CloudSim* com as somas dos custos por segundo, já que (depois de fazer vários testes) se percebeu que depois de mudar o valor da variável *cost* (presente no DatacenterCharacteristics e que guarda o custo por segundo processado na instância), o custo total continua invariável (ver figura 13) devido a um erro.

```
    private static double costperSecond(List<Cloudlet>list, int vms)
{
double costTotal=0;
int size= list.size();
Cloudlet cloudlet;
double costperSecond=0.085;
int vmId[];
vmId=new int[vms];
for(int j=0;j<vms;j++){
vmId[j]=0;}
for (int i = 0; i < size; i++)
        {
                        cloudlet = list.get(i);

        if(vmId[cloudlet.getVmId()]==0){
        vmId[cloudlet.getVmId()]=1;
        double tempoinicial=cloudlet.getExecStartTime();
        double tempofinal=cloudlet.getFinishTime();
        double tempo= tempofinal - tempoinicial;
            if((tempo/3600000) <1){
        costTotal=costTotal + costperSecond;}
            else if((1<(tempo/3600000)) && ((tempo/3600000) <2)){
        costTotal=costTotal + 2*costperSecond;}
            else if((2<tempo/3600000) && ((tempo/3600000)<3)){
        costTotal=costTotal + 3*costperSecond;}
                        }
        }

return costTotal;
```

Figura 13. Código fonte da classe costperSecond

Para solucionar esse problema se fez um método cujos parâmetros de entrada são a lista de *Cloudlet*s e o número total de instâncias que se usam na simulação.

Se declara e inicializa com valor zero a variável *costTotal*, a qual guarda o valor final do custo total por segundo de todas as instâncias utilizadas no final da simulação.

Na variável *size* se coloca o número total de *Cloudlets* que são usadas e a variável *fileSize* indica o tamanho em MB que ocupa a *Cloudlet* antes de ser processada pela instância.

A variável *costperSecond* guarda o valor do custo por segundo do tipo de instância que se quiser simular.

Gera-se um *array* de 1 dimensão tipo *int*, inicializando-o em zero, com um tamanho igual ao número de instâncias. Este *array* tem a função de *flag*[1]; o qual vai registrar se uma instância foi examinada por este método (um 0 na posição da instância implica que a instância ainda não foi examinada, enquanto que um 1 é o contrario). Depois da

[1] Em informática, uma flag (bandeira, em português) é um mecanismo lógico que funciona como semáforo. A utilização de flags como interruptor (i.e., valores 0/1, ligado/desligado, activo/inactivo) permite optimizar as estruturas de dados.

inicialização, se cria um novo comando *for* o qual vai examinar o custo que provocou cada instância. Como as instâncias *Amazon* processam as *Cloudlets* de forma simultânea, só é preciso somar no preço final o tempo que se gastou em processar uma das *X Cloudlets* que cada instância executou (uma instância *Amazon* acaba de executar todas as *Cloudlets* que ela processa no mesmo tempo).

Mediante as classes já implementadas que possuem as Cloudlets (*getExecStartTime())* pode se obter e o tempo inicial de execução (tempo inicial) de cada uma e o tempo de finalização (*tempofinal*) mediante a classe *getFinishTime()*, implícitas no simulador. O resultado da diferencia dessas duas variáveis se armazena na variável tempo, que guardará o tempo exato de execução de cada *Cloudlet*.

Depois disso, se multiplica a variável *costperSecond* com o variável tempo e se soma com o *costTotal* obtido pelo momento. Esclarecer que *Amazon EC2* cobra por horas, por isso se faz diferentes comandos *if* os quais permitem saber entre que faixa de horas processou a instância.

Finalmente se devolve o resultado final.

4.1.5. Simulação

A seguir se detalham os passos que se seguem para gerar a simulação

(1) Criar e configurar o Datacenter. Para isso se tem que definir as variáveis que configuram a instâncias e as *Cloudlet*s segundo as características da *Amazon* (ver Tabela 1,Capítulo 2). Seguidamente se invoca a classe *createDatacenter()* colocando os parâmetros antes mencionados para que seja gerada. Nas figuras 16, 17 e 18 se mostram os parâmetros das instâncias e *Cloudlet*s.

As variáveis definidas no caso da instância tipo *SmallInstance* são mostradas na Figura 14:

```
//Parâmetros da instância
                int size = 160000; //Armazenamento da instância(MB)
                int ram = 1700; // Memória RAM da instância (MB)
                int mips = 2000;// MIPS de cada cpu de Amazon (EC2 Compute Unit)
                int bw = 1000;// Largura de banda precisada
                int pesNumber = 1; //Número de cpus da instância
                String vmm = "Xen"; //VMM name
                int vms=20; // Número de instâncias que se quiser simular, por exemplo 20
```

Figura 14. Código fonte de Small Instance

Deve-se comentar que o tipo de CPU utilizada pela *AmazonEC2* é a *Compute Unit* (ECU), que possui um processador tipo 2007 *Opteron* ou 2007 *Xeon* de 1.0-1.2 GHz, o que permite saber que tem uma potência de 2000 MIPS (MIPS, 2011).

Máquina virtual tipo *Large instance* precisa dos seguintes parâmetros:

```
//Parâmetros da instância

   int size = 850000;      // Armazenamento da instância (MB)
       int ram = 7500;     // Memória RAM da instância (MB)
       int mips = 2000;    // MIPS de cada CPU de Amazon (EC2 Compute Unit)
       int bw = 1000;      // Largura de banda precisada
       int pesNumber = 2; // Número de CPUs da instância
       String vmm = "Xen";// VMM name
       int vms=2;          // Número de instâncias que se quiser simular
```

Figura 15. Código fonte de Large Instance

Configuração da Extra Large instance:

```
//Parâmetros da instância

    int size = 1690000;     // Armazenamento da instância (MB)
        int ram = 15000;    // Memória RAM da instância (MB)
        int mips = 2000;    // MIPS de cada CPU de Amazon (EC2 Compute Unit)
        int bw = 1000;      // Largura de banda precisada
        int pesNumber = 2; // Número de CPUs da instância
        String vmm = "Xen";// VMM name
        int vms=2;          // Número de instâncias que se quiser simular
```

Figura 167. Código fonte de Extra Large Instance

Os parâmetros colocados para a *Cloudlet* podem ser por exemplos esses:

```
Parámetros da Cloudlet
          long length = 900000000;// em MI
          long fileSize = 300;// em MB
          long outputSize = 300; // em MB
          int pesNumberNeeded=1; // número de cpu precisadas para processar a Cloudlet
```

Figura 17. Código fonte dos parâmetros da Cloudlet

Os parâmetros são explicados abaixo:

- *length* e o tamanho da *Cloudlet* em unidades de magnitude tipo MIPS (milhões de instruções por segundo) que serão processadas pela instância.

- *fileSize* indica o tamanho em MB que ocupa a *Cloudlet* antes de ser processada pela instância.

- *outputSize* guarda o tamanho em MB da *Cloudlet* depois de ser processada pela instância.

- *pesNumberNeeded* indica o número de processadores que precisam-se para processar a *Cloudlet*.

Todos os parâmetros podem-se modificar já que uma *Cloudlet* pode ter todo tipo de tamanhos e requisitos.

(2) Geração do *broker* mediante a função já implementada pelo *CloudSim* chamada *createBroker()*.

(3) Criação das instâncias e as *Cloudlet*s mediante a invocação das funções antes comentadas: *createVM()* e *createCloudlet()*. Guardam-se as instâncias e as *Cloudlet*s em listas e enviam-se para o *Broker*.

(4) Inicialização da simulação.

(5) Recibem-se as *Cloudlet*s processadas e se finaliza a simulação. Também se imprimem os resultados de execução das *Cloudlet*s, os custos pelo uso do armazenamento e o custo pela utilização das instâncias.

4.2. EXPERIMENTAÇÃO

Nessa parte do capítulo se faz um experimento com os tipos de instâncias considerando seu custo e tempo de execução para 1, 5 e 10 *Cloudlets*. A *Cloudlet* selecionada tem as seguintes características:

Tamanho total em MIPS = 900000000;
Tamanho total em MB antes de ser processado = 300;
Tamanho total em MB depois de ser processado = 300;

Escolhe-se um tamanho total elevado para poder observar com major exatidão as propriedades e tempos das instâncias. Escolhe-se um tamanho de MB qualquer porque o custo por armazenamento será igual para todas as instâncias independentemente do número de instâncias utilizadas.

Para fazer um estudo detalhado dos custos, tempos, número de instâncias e tipos de instâncias são apresentados gráficos que ajudam a escolher as melhores instâncias segundo as necessidades do usuário. Os custos e tempos são sempre relacionados para a *Cloudlet* com as propriedades antes comentadas, que permite uma generalização com qualquer tipo de *Cloudlet*.

4.2.1 ESTUDO DE 1 CLOUDLET

Na figura 18 se mostram os custos e tempos de processamento de cada uma dos tipos de instâncias.

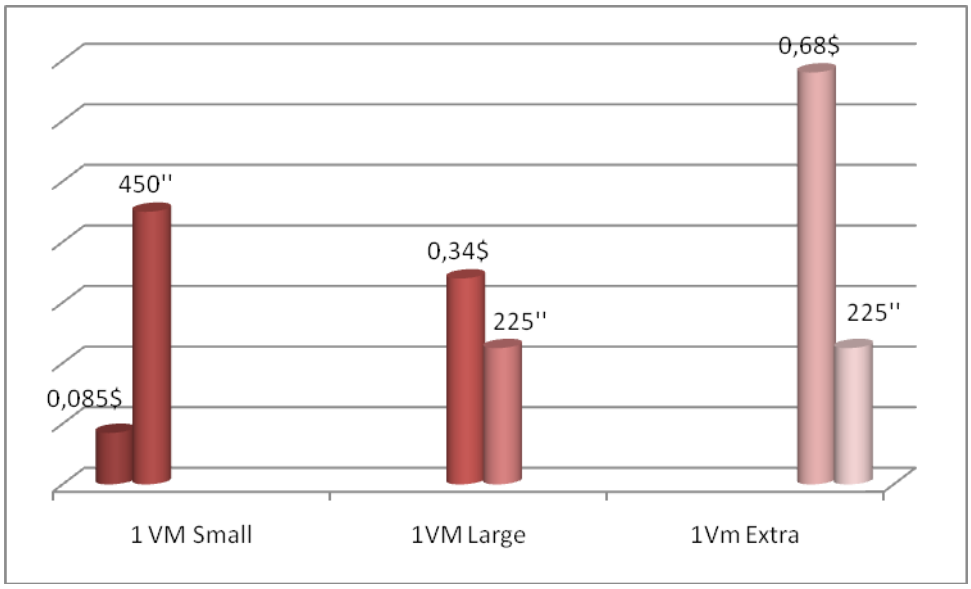


Figura 18. Gráfico custo/tempo 1 Cloudlet

Pode-se observar os 3 tipos de instâncias com o custo na parte esquerda e o tempo de execução na direita. A instância *Small* demora 450 segundos para processar uma *Cloudlet* com um custo de 0,085$. Isto é ideal para um usuário cujo objetivo se focaliza em obter um mínimo preço.

Por outro lado, se o usuário busca a instância ótima em relação custo e tempo, pode se observar que a instância *Large Instance* será a escolhida com um custo de 0,34 $ e um tempo de execução de 225 segundos.

A explicação para que as instâncias *Large* e *Extra Large* tenham o mesmo tempo de processamento é porque uma *Cloudlet* só é processada por um núcleo ao mesmo tempo. A instância *Small* tem um processador de um núcleo enquanto que as instâncias *Large* e *Extra Large* tem 2 processadores de duplo núcleo e 4 processadores de duplo núcleo respectivamente. Por tanto, as instâncias *Large* e *Extra Large* podem processar na metade de tempo que uma instância tipo *Small*. Como pode se observar, para um usuário, não é rentável escolher uma instância *Extra Large* para processar uma única *Cloudlet* com estas características.

4.2.2. ESTUDO DE 5 CLOUDLETS

Se processam 5 *Cloudlets* com os 3 tipos de instâncias, variando o número de instâncias de 1 até 5 (ver figura 19).

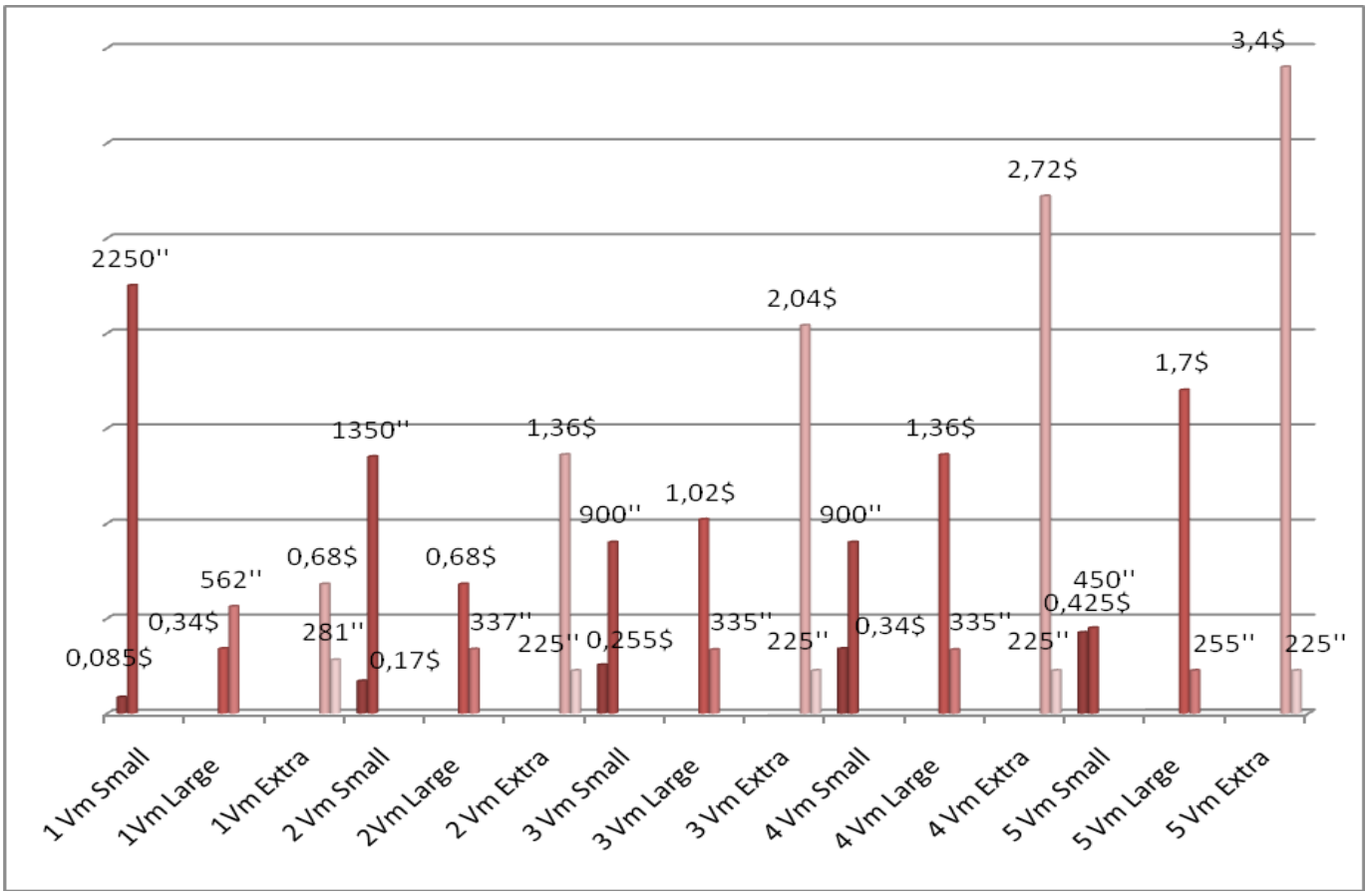


Figura 19. Gráfico custo/tempo 5 Cloudlets

A instância *Small* é a que possui o preço mais econômico porém a mais demorada. Para um usuário cujo objetivo é economizar o custo, a escolha vai ser uma instância *Small* por um preço de 0.085$ e um tempo de 2250 segundos. A melhor oferta em tempo mínimo que oferece uma instância *Small* é a contratação de 5 instâncias por um custo de 0.425$ e um tempo de 450 segundos.

Se o usuário precisa de um tempo mínimo de execução, a eleição seria 2 instâncias *Extra Large* com um custo de 1,36 $ e um tempo de 225 segundos.

A escolha ótima em relação custo/tempo é 3 instâncias *Large* com um custo de 1,02 $ e um tempo de 335 segundos.

Também se observa que as instâncias *Large* e *Extra Large* quando utilizam a partir de 3 e 2 instâncias na frente respectivamente, o custo continua aumentando, mas o tempo continua fixo (tempo mínimo que o processador possui para processar a *Cloudlet*). Isso acontece porque essas instâncias têm um poder de processamento o qual lhes permite processar mais de uma *Cloudlet* no mesmo tempo, e ainda que se aumente o número de instâncias, o tempo mínimo fica fixado, já que uma *Cloudlet* não pode ser processada por duas instâncias no mesmo tempo (fato que faria diminuir o tempo mínimo se for possível).

4.2.3. ESTUDO DE 10 CLOUDLETS

Na figura 20 pode se ver o gráfico referente aos custos e tempos dos três tipos de instâncias.

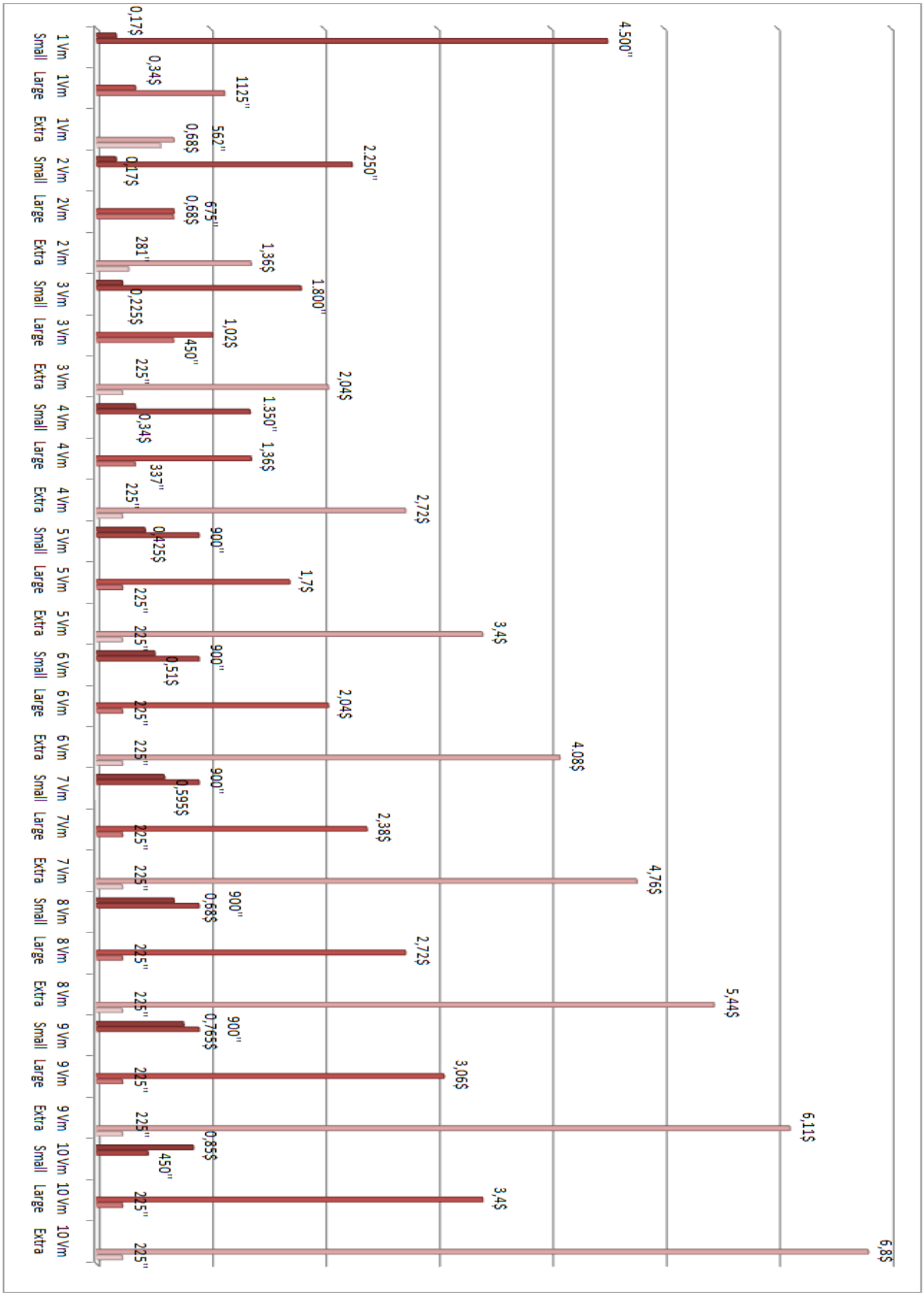


Figura 20. Gráfico custo/tempo 10 Cloudlets

A instância *Small* neste caso, pode se observar que a contratação de uma única instância, o tempo de demora supera uma hora, portanto o usuário deveria pagar por duas horas (0,17$ com um tempo de processamento de 4500 segundos). A contratação ótima de instâncias *Small* é 10 instâncias Small por um custo de 0,85$ e um tempo de demora de 450 segundos. Se o objetivo do usuário é um custo mínimo a contratação de instâncias *Small* será a escolha mais adequada.

A melhor opção para uma escolha de instância tipo *Large* é 5 instâncias por um custo de 1,7$ e um tempo de 225 segundos.

Das instâncias tipo *Extra Large* a contratação ótima é 3 instâncias por um tempo de 225 segundos e um preço de 2,04$.

Se um usuário pretende um custo mínimo deve escolher 2 instâncias *Small Instance* por um custo de 0,17 $ e um tempo de 2250 segundos.

No caso que o usuário precisar de o mínimo tempo deveria contratar 3 instâncias *Extra Large* por um preço de 2,4$ e um tempo de 225 segundos ou 5 instâncias *Large* por 1,7$ pelo mesmo tempo de execução. Então a escolha ótima em relação com o tempo e o custo é a contratação de 5 instâncias *Large*.

5. CONCLUSÕES

No início desse estudo foi abordada a intenção de simular mediante um simulador chamado *CloudSim*, os preços e tempos de processamento de serviços da nuvem *Amazon EC2*. O capítulo 4 mostrou as alterações que foram feitas no simulador bem como exemplos de simulação utilizando instâncias diferentes da *Amazon* (*Small, Large* e *Extra Large*) para processar aplicações *Batch* distintas. Depois de fazer um estudo sobre os custos e tempos de processamento para diferentes aplicações, pode-se tirar algumas conclusões.

Por exemplo, caso se precise executar poucas *Cloudlets* (de uma até dez *Cloudlets*, sendo estes números dados indicativos) a instância *Small* é a mais econômica e também a que mais tempo de processamento precisa. Será ótima para usuários com pouco poder aquisitivo e que não precisem de um tempo limite de processamento, como poderia ser a hospedagem de uma página web de baixa demanda. A instância *Large* é a que apresenta uma melhor relação entre o tempo de execução e o custo, enquanto que a instância *Extra Large* é a instância mais rápida e também a mais cara. A instância *Extra Large* pode ser adequada para usuários que precisem de velocidade de processamento como, por exemplo, para a utilização de uma base de dados.

Cabe destacar também que no momento que se quiser processar *Cloudlets* de grande tamanho ou um número elevado de *Cloudlets*, pode acontecer que um número de X instâncias tenha o mesmo custo que a contratação de X+1 instâncias, devido a que o preço das instâncias é cobrado por horas. Por exemplo, 1 instância *Small* demora mais de uma hora para o processamento de 10 *Cloudlets* o que faz com que o preço dessa instância seja o mesmo que a contratação de 2 instâncias *Small* que podem processar em menos tempo.

Podemos tirar como conclusão final que a computação em nuvem, em particular, as empresas IaaS como *Amazon EC2* estão permitindo a criação de empresas que, com pouco capital, podam ter no alcance deles uma tecnologia tão complexa por um custo muito menor ao que se tinha antes de existir este tipo de empresas. A simulação do custo e tempo de processamento pode ser de grande utilidade para empresas que desejam tomar decisões e investigar qual melhor custo benefício para executar suas aplicações em relação a tempo e dinheiro.

BIBLIOGRAFIA

AMAZON. Amazon Web Services. 2009. Disponível em: http:// aws.amazon.org

AMAZON. Amazon Elastic Compute Cloud. 2009. Disponível em: http://aws.amazon.com/es/ec2/

AMAZON. Amazon Simple Storage Service. 2009. Disponível em: http://aws.amazon.com/s3/

AMAZON. Amazon Elastic Block Storage. 2009. Disponível em: http://aws.amazon.com/es/ebs/

AMAZON. Perguntas freqüentes. 2011. Disponível em:

http://aws.amazon.com/es/ec2/faqs/#How_many_instances_can_I_run_in_Amazon_EC2

BLOGFR. Avantajem da Computação em Nuvem. 2009. Disponível em: http://www.feliperm.info/2009/03/04/cloud-computing-ventajas-y-desventajas/

CALHEIROS, Rodrigo N., et al. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software: Practice and Experience (SPE), Volume 41, Number 1, Pages: 23-50, ISSN: 0038-0644, Wiley Press, New York, USA, January, 2011.

CONSEGUI. Computação em Nuvem. 2010. Disponível em : http://www.consegi.gov.br/consegi-1-espanhol/computacion-en-nube?set_language=es

CLOUDS. Cloud computing and Distributed Systems. 2010. Disponível em: http://www.cloudbus.org/cloudsim/

GOOGLE. Informação geral da empresa Google. 2010. Disponível em: http://es.wikipedia.org/wiki/Google

INF. FACTORIES. Revista Wired, artigo The Information Factories. 2006.

MAESTROSDELWEB. Assegurança da Nuvem. 2010. Disponível em: http://www.maestrosdelweb.com/editorial/amenazas-seguridad-en-la-nube-cloud-computing/

MIPS. Definição de MIPS e tipos de processadores. 2011.Disponível em:

http://pt.wikipedia.org/wiki/MIPS_(medida)

PARADIGM. Assegurança da Nuvem. 2009. Disponível em: http://csrc.nist.gov/groups/SNS/cloud-computing/index.html

SAMPAIO, A., et al. Uni4Cloud Uma Abordagem para Implantação de Aplicações sobre Múltiplas Nuvens de Infra-Estrutura. In: WCGA - VIII Workshop em Clouds, Grids e Aplicações (SBRC-2010), 2010, Gramado. Anais do VIII Workshop em Clouds, Grids e Aplicações (WCGA), 2010.

SUNMICROSYSTEMS. Cloud Computing Architecture, Withe Paper. 2009

TECHNOLOGY NEWS. Is Cloud Computing Really Ready for Prime Time? .2009

UNIV. OF CALIFORNIA AT BERKELEY. Above the Clouds: A Berkeley View of Cloud Computing 2009.Electrical Engineering and Computer Sciences University of California at Berkeley.

VIRTUAL_LINUX. Virtualização. 2010. Disponível em: http://blog.smaldone.com.ar/2008/09/20/virtualizacion-de-hardware/

VMWARE. Nuvem privada. 2011. Disponível em: http://www.vmware.com/lasp/solutions/cloud-computing/private-cloud/

WORDPRESS. Serviço Web. 2010. Disponível em: http://eamodeorubio.wordpress.com/2010/07/19/servicios-web-1-soap-no-gracias/