

Naylor Garcia Bachiega

Algoritmo de Escalonamento de Instância de Máquina Virtual na
Computação em Nuvem

São José do Rio Preto
2014

Naylor Garcia Bachiega

Algoritmo de Escalonamento de Instância de Máquina Virtual na
Computação em Nuvem

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração - Sistemas de Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Campus de São José do Rio Preto.

Orientador: Prof^a. Dr^a. Roberta Spolon

São José do Rio Preto
2014

Bachiega, Naylor Garcia.

Algoritmo de escalonamento de instância de máquina virtual na computação em nuvem / Naylor Garcia Bachiega. -- São José do Rio Preto, 2014

92 f. : il., gráfs., tabs.

Orientador: Roberta Spolon

Dissertação (mestrado) – Universidade Estadual Paulista “Júlio de Mesquita Filho”, Instituto de Biociências, Letras e Ciências Exatas

1. Computação. 2. Computação em nuvem. 3. Recursos de redes de computadores. 4. Algoritmos de computador. 5. Armazenamento de dados. I. Spolon, Roberta. II. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. III. Título.

CDU – 681.3.025

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Naylor Garcia Bachiega

Algoritmo de Escalonamento de Instância de Máquina Virtual na
Computação em Nuvem

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração - Sistemas de Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Campus de São José do Rio Preto.

Comissão Examinadora

Prof^a. Dr^a. Roberta Spolon
UNESP – Bauru
Orientador

Prof. Dr. Antonio Carlos Sementille
UNESP – Bauru

Prof. Dr. Luís Carlos Trevelin
UFSCAR – São Carlos

São José do Rio Preto
19 de maio de 2014

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela existência de todas as coisas, inclusive a minha. Em segundo lugar agradeço a minha esposa Kaliane, pela ajuda, paciência, compreensão e lealdade ao meu lado nessa fase da minha vida. À Família pela paciência e a distância que mantive nesses anos.

Em especial, agradeço aos professores Drs. Roberta Spolon e Marcos Antônio Cavenaghi por ter acreditado, me orientado e por sempre estarem presentes. Também agradeço aos professores Drs. Aparecido Nilceu Marana, Renata Spolon Lobato, Adriano Cansian, Aleardo Manacero Jr., Hilda Carvalho, José Remo Ferreira Brega e todos os demais professores da pós-graduação.

Ao amigo Henrique Martins, companheiro de mestrado que tanto ajudou nesses três anos. Ao Rafael Hamamura e Felipe Maciel pela ajuda e disponibilidade e todos àqueles da pós-graduação que de alguma forma participaram e contribuíram para esse trabalho.

E conhecereis a verdade, e a verdade vos libertará.

Yeshua Hamashiach

RESUMO

Na tentativa de reduzir custos aproveitando de maneira eficiente recursos computacionais, novas tecnologias e arquiteturas desenvolvidas estão conquistando grande aceitação do mercado. Uma dessas tecnologias é a Computação em Nuvem, que tenta resolver problemas como consumo energético e alocação de espaço físico em centros de dados ou grandes empresas. A nuvem é um ambiente compartilhado por diversos clientes e permite um crescimento elástico, onde novos recursos como *hardware* ou *software*, podem ser contratados ou vendidos a qualquer momento. Nesse modelo, os clientes pagam por recursos que utilizam e não por toda a arquitetura envolvida. Sendo assim, é importante determinar de forma eficiente como esses recursos são distribuídos na nuvem. Portanto, esse trabalho teve como objetivo desenvolver um algoritmo de escalonamento para nuvem que determinasse de maneira eficiente a distribuição de recursos dentro da arquitetura. Para alcançar esse objetivo, foram realizados experimentos com gestores de nuvem *open-source*, detectando a deficiência dos algoritmos atuais. O algoritmo desenvolvido foi comparado com o algoritmo atual do gestor *OpenStack Essex*, um gestor de nuvem *open-source*. Os resultados experimentais demonstraram que o novo algoritmo conseguiu determinar as máquinas menos sobrecarregadas da nuvem, conseguindo desse modo, distribuir a carga de processamento dentro do ambiente privado.

Palavras-chave: Computação em nuvem. Economia de recursos. Escalonamento de recursos. Escalonador.

ABSTRACT

In an attempt to reduce costs by taking advantage of efficient computing resources, new technologies and architectures developed are gaining wide acceptance in the market. One such technology is cloud computing, which tries to solve problems like energy consumption and allocation of space in data centers or large companies. The cloud is an environment shared by multiple clients and enables elastic growth, where new features such as hardware or software, can be hired or sold at any time. In this model, customers pay for the resources they use and not for all the architecture involved. Therefore, it is important to determine how efficiently those resources are distributed in the cloud. Therefore, this study aimed to develop a scheduling algorithm for cloud efficiently determine the distribution of resources within the architecture. To achieve this goal, experiments were conducted with managers of open-source cloud, detecting the deficiency of current algorithms. This algorithm was compared with the algorithm of the OpenStack Essex manager, a manager of open-source cloud. The experimental results show that the new algorithm could determine the machines less the cloud overloaded, achieving thereby distribute the processing load within the private environment.

Keywords: Cloud computing. Resource saving. High availability. Resource scheduling.

LISTA DE ILUSTRAÇÕES

Figura 1 – Pesquisa pelo termo: Computação em Nuvem.	17
Figura 2 – Controle de máquinas virtuais pelo <i>hypervisor</i>	19
Figura 3 – Tipos de serviços da Computação em Nuvem.	20
Figura 4 – Modelos da Computação em Nuvem.	22
Figura 5 – Infraestrutura da Computação em Nuvem.	24
Figura 6 – Redes virtuais na Computação em Nuvem.	26
Figura 7 – Modelo de nuvem utilizando cache.	35
Figura 8 – Listagem de processos do <i>KVM</i> no Linux.	38
Figura 9 – Sistema de escalonamento.	39
Figura 10 – Taxonomia de Escalonamento.	40
Figura 11 – Seleção de nós – <i>nova-scheduler</i>	46
Figura 12 – Escalonamento no gestor de nuvem.	48
Figura 13 – Sistemas operacionais virtualizados com cargas distantes.	50
Figura 14 – Perfis de memória.	50
Figura 15 – Sistemas operacionais disponíveis para o gestor.	51
Figura 16 – Teste de escalonamento – comportamento 1: 08 instâncias.	51
Figura 17 – Sistemas operacionais virtualizados com cargas aproximadas. ...	52
Figura 18 – Teste de escalonamento – comportamento 2: 30 instâncias.	52
Figura 19 – Programa de simulação de consumo variável.	53
Figura 20 – Teste de escalonamento – comportamento 3: 08 instâncias.	54
Figura 21 – Taxonomia de escalonamento em <i>Grid</i>	55
Figura 22 – Tabela de armazenamento das informações dos nós.	58
Figura 23 – Agendamento de execução de programa (<i>crontab</i>).	58
Figura 24 – Teste do protótipo – comportamento 1: 08 instâncias.	60
Figura 25 – Teste do protótipo – comportamento 2: 30 instâncias.	60
Figura 26 – Teste do protótipo – comportamento 3: 08 instâncias.	61
Figura 27 – <i>Layout</i> de instalação do <i>OpenStack Essex</i>	73
Figura 28 – Interface <i>Web</i> de gerenciamento – <i>Horizon</i>	74
Figura 29 – Ambiente de Instalação – <i>Eucalyptus</i>	75

LISTA DE TABELAS

Tabela 1 – Comparação entre os gestores de nuvem.....	33
Tabela 2 – Especificações dos nós.	49
Tabela 3 – Recursos disponíveis antes do teste.	49
Tabela 4 – Especificação de <i>hardware</i> para a nuvem.....	73

LISTA DE ABREVIATURAS E SIGLAS

AR	<i>Advance Reservations</i>
API	<i>Application Programming Interface</i>
AWS	<i>Amazon Web Services</i>
BGP	<i>Border Gateway Protocol</i>
BIOS	<i>Basic Input/Output System</i>
CaaS	<i>Communications as a Service</i>
CPU	<i>Central Processing Unit</i>
DaaS	<i>Datacenter as a Service</i>
DDoS	<i>Distributed Denial-of-Service</i>
DoS	<i>Denial of Service</i>
EBS	<i>Amazon Elastic Block Store</i>
EC2	<i>Amazon Elastic Compute Cloud</i>
HaaS	<i>Hardware as a Service</i>
HTTP	<i>Hypertext Transfer Protocol</i>
HVM	<i>Hardware Virtual Machine</i>
IaaS	<i>Infrastructure as a Service</i>
iSCSI	<i>Internet Small Computer System Interface</i>
IPS	<i>Intrusion prevention systems</i>
KaaS	<i>Knowledge as a Service</i>
KVM	<i>Kernel-based Virtual Machine</i>
LAN	<i>Local Area Network</i>
LVM	<i>Logical Volume Manager</i>
MB	<i>Megabyte</i>
MMU	<i>Memory Management Unit</i>
NASA	<i>National Aeronautics and Space Administration</i>
NFS	<i>Network File System</i>
NTP	<i>Network Time Protocol</i>
NUMA	<i>Non-uniform Memory Access</i>
OVF	<i>Open Virtualization Format</i>
PaaS	<i>Platform as a Service</i>
PCI	<i>Peripheral Component Interconnect</i>
PHP	<i>HyperText PreProcessor</i>

RAM	<i>Random Access Memory</i>
RPC	<i>Remote Procedure Call</i>
SaaS	<i>Software as a Service</i>
SAN	<i>Storage Area Network</i>
SCSI	<i>Small Computer System Interface</i>
SLA	<i>Service Level Agreement</i>
SMP	<i>Processor Symmetric Multiprocessing</i>
SO	<i>Sistema Operacional</i>
SOAP	<i>Simple Object Access Protocol</i>
SQL	<i>Structured Query Language</i>
S3	<i>Amazon Simple Storage Service</i>
TLB	<i>Translation Address Table</i>
VCL	<i>Virtual Computing Lab.</i>
VIM	<i>Virtual Infrastructure Manager</i>
VLAN	<i>Virtual Local Area Network</i>
VM	<i>Virtual Machine</i>
VMM	<i>Virtual Machine Monitor</i>
VPN	<i>Virtual Private Network</i>
UHCI	<i>Universal Host Controller Interface</i>
USB	<i>Universal Serial Bus</i>
XML	<i>Extensible Markup Language</i>
XSS	<i>Cross-site scripting</i>

SUMÁRIO

1	INTRODUÇÃO.....	15
1.1	OBJETIVOS.....	16
1.2	ORGANIZAÇÃO DO TEXTO.....	16
2	COMPUTAÇÃO EM NUVEM.....	17
2.1	CONSIDERAÇÕES INICIAIS.....	17
2.2	DIFERENÇAS ENTRE VIRTUALIZAÇÃO E EMULAÇÃO.....	18
2.3	VIRTUALIZAÇÃO NA NUVEM.....	18
2.4	CLASSES DE SERVIÇOS DA COMPUTAÇÃO EM NUVEM.....	20
2.4.1	INFRAESTRUTURA COMO UM SERVIÇO.....	20
2.4.2	PLATAFORMA COMO UM SERVIÇO.....	21
2.4.3	SOFTWARE COMO UM SERVIÇO.....	21
2.5	MODELOS DA COMPUTAÇÃO EM NUVEM.....	21
2.6	CARACTERÍSTICAS DA COMPUTAÇÃO EM NUVEM.....	22
2.7	INFRAESTRUTURA DA COMPUTAÇÃO EM NUVEM.....	23
2.7.1	SUORTE PARA VIRTUALIZAÇÃO.....	24
2.7.2	PROVISIONAMENTO DE RECURSOS SOB DEMANDA.....	24
2.7.3	MÚLTIPLOS <i>BACKEND HYPERVISORS</i>	25
2.7.4	VIRTUALIZAÇÃO DE ARMAZENAMENTO.....	25
2.7.5	INTERFACE PARA NUVENS PÚBLICAS.....	25
2.7.6	REDE VIRTUAL.....	26
2.7.7	ALOCAÇÃO DINÂMICA DE RECURSOS.....	26
2.7.8	MECANISMO DE RESERVA E NEGOCIAÇÃO.....	27
2.7.9	ALTA DISPONIBILIDADE E RECUPERAÇÃO DE DADOS.....	27
2.8	PRINCIPAIS GESTORES DA COMPUTAÇÃO EM NUVEM.....	28
2.8.1	<i>APACHE VCL</i>	28
2.8.2	<i>CITRIX ESSENTIALS</i>	28
2.8.3	<i>EUCALYPTUS</i>	29
2.8.4	<i>NIMBUS3</i>	29
2.8.5	<i>OPENNEBULA</i>	30
2.8.6	<i>OPENPEX</i>	30
2.8.7	<i>OVRT</i>	31
2.8.8	<i>VMWARE VSPHERE</i>	31
2.8.9	<i>OPENSTACK ESSEX</i>	32
2.9	LIMITAÇÕES ENCONTRADAS COM OS GESTORES.....	34
2.10	CONSIDERAÇÕES FINAIS.....	36
3	ESCALONAMENTO DE RECURSOS.....	37
3.1	CONSIDERAÇÕES INICIAIS.....	37
3.2	ESCALONAMENTO.....	38

3.3	TAXONOMIA DE ESCALONAMENTO.....	39
3.4	BALANCEAMENTO DE CARGA.....	42
3.4.1	CLASSIFICAÇÃO DOS ALGORITMOS DE BALANCEAMENTO DE CARGA.....	42
3.4.2	ALGORITMOS DE BALANCEAMENTO DE CARGA	43
3.4.3	MÉTRICAS PARA BALANCEAMENTO DE CARGA.....	44
3.5	ESCALONAMENTO E BALANCEAMENTO NO OPENSTACK.....	45
3.6	CONSIDERAÇÕES FINAIS.....	47
4	METODOLOGIA E PROTÓTIPO.....	48
4.1	CONSIDERAÇÕES INICIAIS	48
4.2	METODOLOGIA.....	48
4.2.1	COMPORTAMENTO 1: MÁQUINAS VIRTUAIS DE CONSUMO CONSTANTE COM CARGAS DISTANTES.....	49
4.2.2	COMPORTAMENTO 2: MÁQUINAS VIRTUAIS DE CONSUMO CONSTANTE COM CARGAS APROXIMADAS	51
4.2.3	COMPORTAMENTO 3: MÁQUINAS VIRTUAIS DE CONSUMO VARIÁVEL.....	53
4.3	O PROTÓTIPO	54
4.3.1	RECURSOS DOS NÓS	56
4.3.2	ESCOLHA DOS NÓS.....	59
4.3.3	COMPORTAMENTO 1: MÁQUINAS VIRTUAIS DE CONSUMO CONSTANTE COM CARGAS DISTANTES.....	59
4.3.4	COMPORTAMENTO 2: MÁQUINAS VIRTUAIS DE CONSUMO CONSTANTE COM CARGAS APROXIMADAS	60
4.3.5	COMPORTAMENTO 3: MÁQUINAS VIRTUAIS DE CONSUMO VARIÁVEL.....	61
4.4	CONSIDERAÇÕES FINAIS.....	61
5	CONCLUSÃO	63
5.1	CONSIDERAÇÕES INICIAIS	63
5.2	CONTRIBUIÇÕES	64
5.3	TRABALHOS FUTUROS.....	65
	REFERÊNCIAS.....	66
	GLOSSÁRIO.....	70
	APÊNDICE A - INSTALAÇÃO DO EUCALYPTUS	73
	APÊNDICE B - INSTALAÇÃO DO OPENNEBULA	78
	APÊNDICE C - INSTALAÇÃO DO OPENSTACK ESSEX.....	81
	ANEXO A – TEST NODE FX.PL	85

ANEXO B – TEST NODE RAND.PL..... 87

ANEXO C – NODE RESOURCES.PY 89

ANEXO D – CHOICE NODE.PY 92

1 INTRODUÇÃO

Com o aumento constante do uso computacional, problemas como demanda energética e espaço em centros de dados estão ocorrendo em todo o mundo. Muitas soluções estão sendo projetadas para sanar esse tipo de situação, entre elas está a Computação em Nuvem (*Cloud Computing*) (HE; HE, 2011).

A Computação em Nuvem é vista como uma tendência no cenário atual em quase todas as organizações. As vantagens de usar a computação em nuvem são: redução de *hardware* e custo de manutenção, acessibilidade, flexibilidade e um processo altamente automatizado em que o cliente não precisa se preocupar com *upgrade* de *software* (BHADAURIA; CHAKI, 2011).

Sabahi (2011) define Computação em Nuvem como sendo um ambiente de rede baseado no compartilhamento de recursos computacionais. Na verdade, nuvens são baseadas na Internet e tentam disfarçar a complexidade para os clientes. A Computação em Nuvem refere-se ao *hardware* e *software* entregues como serviços por meio da Internet pelos centros de dados. Empresas que fornecem nuvens utilizam-se de tecnologias de virtualização, combinadas com suas habilidades para fornecer recursos de computação por meio de sua infraestrutura de rede.

Ainda segundo Sabahi (2011), em ambientes de nuvem, vários tipos de máquinas virtuais estão hospedados no mesmo servidor físico como infraestrutura. Na nuvem, clientes só devem pagar por aquilo que usam e não devem pagar por recursos locais que precisam como armazenamento ou infraestrutura.

Ao ligar um aparelho elétrico em uma tomada, nós não nos importamos como a energia elétrica é gerada e nem como ela chega a esse estabelecimento. Isto é possível porque eletricidade é virtualizada, isto é, está prontamente disponível a partir de uma tomada de parede que esconde estações de geração de energia e um enorme grid. Quando essa distribuição é estendida a tecnologias de informação, este conceito significa entregar funções úteis escondendo como funcionam internamente. A Computação em Nuvem, para ser considerada totalmente virtualizada, deve permitir que os

computadores sejam construídos a partir de componentes distribuídos tais como os recursos de armazenamento, processamento de dados e software (BUYA; BROBERG; GOSCISNSKI, 2011).

Nuvens utilizam algoritmos de escalonamento para distribuir recursos dentro da arquitetura, como por exemplo, instâncias de máquinas virtuais que são oferecidas para clientes como serviço. Nesses moldes, determinar o escalonamento de modo eficiente pode assegurar a qualidade dos serviços oferecidos.

1.1 OBJETIVOS

Tendo em vista que os atuais algoritmos de escalonamento das nuvens *open-source* determinam de modo estático os recursos disponíveis, esse trabalho teve como objetivo criar um algoritmo dinâmico de escalonamento para determinar de modo eficiente quais os nós computacionais dentro de uma nuvem, possuem recursos para abrigar novas instâncias de máquinas virtuais. Esse algoritmo ainda permite determinar o consumo de recursos físicos de cada instância em um nó computacional, permitindo que trabalhos futuros possam realocar essas instâncias, aproveitando a capacidade total do nó.

1.2 ORGANIZAÇÃO DO TEXTO

Este trabalho foi dividido em cinco capítulos:

- No capítulo 2 foi feito um estudo sobre Computação em Nuvem, levantando as principais características, arquitetura, modelos de serviços, aspectos de segurança e as principais ferramentas.
- No capítulo 3 foi apresentado um estudo sobre os algoritmos de escalonamento de recursos e balanceamento de carga.
- No capítulo 4 é apresentada metodologia e o protótipo desenvolvido. Também são apresentados os resultados e discussões sobre o trabalho.
- No capítulo 5 foi apresentada a conclusão desse trabalho e sugestões para estudos futuros.

2 COMPUTAÇÃO EM NUVEM

Esse capítulo apresenta detalhes da arquitetura de nuvem, definindo seus modelos, características e tipos, analisando os principais gestores e suas diferenças.

2.1 CONSIDERAÇÕES INICIAIS

O conceito de Computação em Nuvem vem sendo discutido e essa nova arquitetura tem ganhado repercussão em larga escala. Conforme mostra a Figura 1, há um crescente aumento em pesquisas pelo Google.

Figura 1 – Pesquisa pelo termo: Computação em Nuvem.



Fonte: TRENDS, 2012.

A ideia básica da arquitetura é fornecer poder computacional sob demanda, onde conforme a necessidade, mais máquinas virtuais podem ser ativadas na nuvem, oferecendo mais recursos se necessário, tendo assim um crescimento elástico, onde máquinas são adicionadas para aumentar o poder de processamento e/ou armazenamento.

A nuvem utiliza-se de conceitos existentes, como virtualização e emulação, e a partir disso monta-se uma nova arquitetura reunindo-os para atingir seu objetivo, disponibilizando recursos conforme demanda. Segundo Buyya, Broberg e Goscinski (2011) a virtualização de *hardware* permite criar

múltiplas máquinas virtuais sobre uma máquina física.

2.2 DIFERENÇAS ENTRE VIRTUALIZAÇÃO E EMULAÇÃO

Em um emulador todas as instruções realizadas pela máquina real são abstraídas em um ambiente de software, possibilitando executar um aplicativo de uma plataforma em outra, por exemplo, um aplicativo do Windows sendo executado no Linux ou um aplicativo i386 sendo executado em uma plataforma Sparc. (LAUREANO, 2006)

Portanto, um emulador tem a capacidade de simular um computador físico, este não tem nenhuma relação com o *hardware* subjacente, assim, é necessário fornecer todos os dispositivos através da transcrição de instruções. Porém essa emulação causa uma perda de eficiência ao realizar a tradução de cada instrução da máquina emulada para a máquina física. (LAUREANO, 2006)

No caso da virtualização, um ambiente é criado por um monitor de máquina virtual (*Virtual Machine Monitor – VMM*), sendo este ambiente controlado pelo *hypervisor*¹. Esse monitor fornece uma interface de *hardware* idêntica a subjacente através da multiplexação, permitindo que instruções que não comprometam o isolamento das máquinas virtuais sejam executadas diretamente sobre esse *hardware*. (LAUREANO, 2006)

2.3 VIRTUALIZAÇÃO NA NUVEM

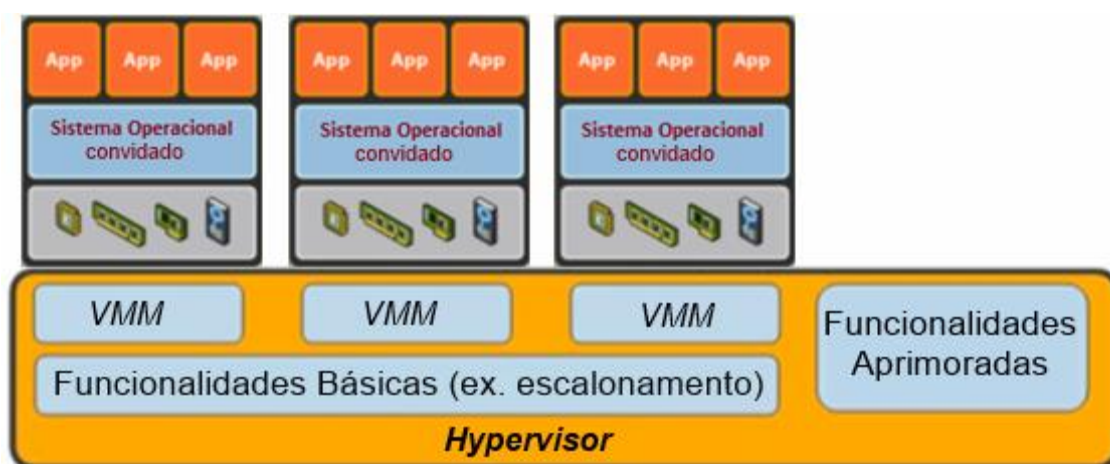
A Computação em Nuvem utiliza-se da virtualização para criar um ambiente (nuvem), onde aloca instâncias (sistemas operacionais virtualizados) de acordo com os recursos disponíveis (máquinas físicas). Essas instâncias de máquinas virtuais são alocadas de acordo com as máquinas físicas que

¹ É uma camada de *software* entre o *hardware* e o sistema operacional, que controla o acesso dos sistemas operacionais convidados aos dispositivos de *hardware* (VMWARE, 2012).

compõem o parque de máquinas da nuvem.

Conforme Figura 2, a virtualização consiste basicamente na camada que faz a mediação entre o *hardware* e a máquina virtual (VM). Largamente utilizadas, as VMs contribuíram no melhor aproveitamento de recursos, considerando que ocasionalmente um computador atinge a totalidade de sua capacidade de processamento.

Figura 2 – Controle de máquinas virtuais pelo *hypervisor*.



Fonte: VMWARE, 2012.

A camada de virtualização é formada por um monitor de máquina virtual (*Virtual Machine Monitor – VMM*), sendo este controlado pelo *hypervisor*. Esse monitor fornece uma interface de *hardware* idêntica a subjacente através da multiplexação, permitindo que instruções que não comprometam o isolamento das máquinas virtuais sejam executadas diretamente sobre esse *hardware* (VMWARE, 2012).

Existem no mercado diversas soluções para máquinas virtuais, sendo a maior parte soluções *open-source*, suas diferenças se destacam na gama de serviços oferecidos. Os principais *hypervisors* atualmente são *Xen*, *VMware*, *KVM* e *QEMU*, este último, diferentemente de um virtualizador, é um emulador, em que o hardware é totalmente simulado através de software (BUYA; BROBERG; GOSCISNSKI, 2011).

2.4 CLASSES DE SERVIÇOS DA COMPUTAÇÃO EM NUVEM

Segundo Buyya, Broberg e Goscisnski (2011), a Computação em Nuvem é dividida em três classes de serviços de acordo com o modelo de serviços oferecidos pelos provedores: Infraestrutura como um Serviço (*IaaS*), *Software* como um Serviço (*SaaS*) e Plataforma como um Serviço (*Paas*) conforme pode ser visualizado na Figura 3.

Figura 3 – Tipos de serviços da Computação em Nuvem.

Classes Serviço	Acesso Principal e Ferramentas de Gerenciamento	Conteúdo do Serviço
 SaaS	Web Browser	Aplicações em Nuvem Redes Sociais, Pacote Office, ERP, CRM, Processamento de Vídeo
 PaaS	Ambiente de Desenvolvimento em Nuvem	Plataforma em Nuvem Linguagem de Programação, dados estruturados.
 IaaS	Gestor de Infraestrutura Virtual	Infraestrutura de Nuvem Servidores, Armazenamento de Dados, Firewall, Balanceamento de Carga

Fonte: BUYYA; BROBERG; GOSCISNSKI, 2011.

2.4.1 INFRAESTRUTURA COMO UM SERVIÇO

Nesse tipo de serviço, o cliente da nuvem tem a sua disposição processamento, redes, armazenamento e outros recursos computacionais, onde poderá instalar sistemas operacionais ou qualquer outro sistema próprio. O cliente não administra ou controla a infraestrutura da nuvem subjacente e paga somente pela estrutura que utilizar. Pode-se citar como exemplos de *IaaS*, serviços como *Amazon Elastic Compute Cloud* (Amazon EC2), *Amazon Simple Storage Service* (Amazon S3), *Eucalyptus*, *OpenNebula* e *OpenStack*

Essex (SASIKALA, 2012).

2.4.2 PLATAFORMA COMO UM SERVIÇO

Esse tipo de serviço oferece um ambiente para desenvolvedores criarem, testarem e implantarem suas aplicações, não se importando com a infraestrutura, quantidade de memória ou requerimentos de *hardware*. Pode-se citar o *Google Apps* como exemplo desse serviço, onde é oferecido um ambiente escalável para desenvolvimento e hospedagem de aplicações *Web* ou *Microsoft Azure* (SASIKALA, 2012).

2.4.3 SOFTWARE COMO UM SERVIÇO

Nesse tipo, as aplicações residem no topo do modelo, oferecendo “*software* sob demanda”. As aplicações são acessíveis a partir de vários dispositivos como um navegador *Web*, (por exemplo, *webmail*). O cliente não administra ou controla a infraestrutura da nuvem, como a rede, servidores, sistemas operacionais, armazenamento ou mesmo a aplicação. A cobrança do serviço nesse caso pode ser feita com base no número de usuários (SASIKALA, 2012).

Além dos três tipos de serviços citados acima, outros autores também consideram: *CaaS* (Comunicações como um Serviço), *DaaS* (*Datacenter* como um Serviço), *KaaS* (Conhecimento como um Serviço) e *HaaS* (*Hardware* como um Serviço) (HE; HE, 2011).

2.5 MODELOS DA COMPUTAÇÃO EM NUVEM

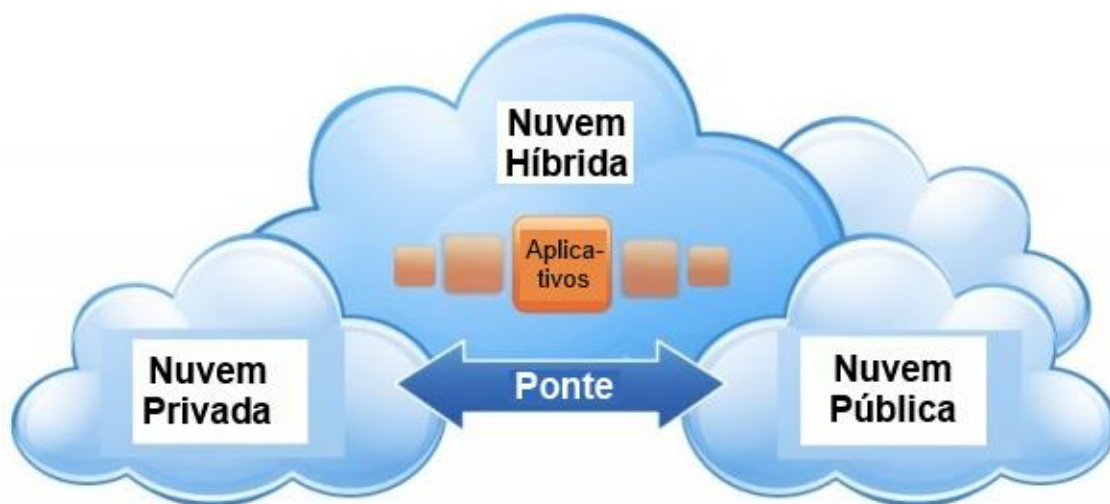
Segundo Sasikala (2012), os modelos de nuvem são definidos de acordo com o grupo que irá utilizar os serviços da arquitetura, estes podem ser de quatro tipos: nuvem pública, privada, comunidade e híbrida.

- **Nuvem Pública:** a infraestrutura é fornecida para muitos clientes e é gerenciada por terceiros. Várias empresas podem trabalhar simultaneamente na mesma infraestrutura. O usuário paga por aquilo que utilizar.
- **Nuvem Privada:** infraestrutura disponibilizada apenas para um

cliente específico e gerenciada pela própria organização ou por terceiros. Este utiliza o conceito de virtualização de máquinas, usando rede proprietária.

- **Nuvem Comunidade:** infraestrutura compartilhada por várias organizações por uma causa comum, podendo ser gerenciada por elas mesmas ou por um prestador de serviços.
- **Nuvem Híbrida:** compõe um ou mais modelos de implantação de nuvem, em que a comunicação entre elas é realizada de modo transparente (como uma *bridge*), conforme Figura 4.

Figura 4 – Modelos da Computação em Nuvem.



Fonte: THANGARAJ, 2012.

2.6 CARACTERÍSTICAS DA COMPUTAÇÃO EM NUVEM

Segundo Buyya, Broberg e Goscisnski (2011), de acordo com os modelos de nuvem, esta precisa oferecer alguns serviços para que se torne atraente para os clientes, entre eles, *self-service*, modo de faturamento, elástico e que permita customização.

- **Self-service:** nesse caso, a nuvem deve estar preparada para receber clientes por demanda e com acesso imediato aos recursos necessários, onde se possa pagar, utilizar e customizar os serviços sem intervenção de operadores.
- **Faturamento:** a nuvem deve oferecer serviço por bilhetagem ou

por hora (recursos consumidos), não obrigando o cliente a pagar por recursos não utilizados.

- **Elástico:** os clientes esperam que a nuvem lhes ofereça recursos sob demanda, tanto para cima quanto para baixo, de acordo com o uso.
- **Customização:** em uma nuvem há uma grande disparidade entre as necessidades do usuário, assim, os recursos alugados a partir da nuvem devem ser altamente personalizáveis. No caso de *IaaS*, personalização significa permitir que os usuários possam implantar dispositivos virtuais para ter acesso (*root*) privilegiado a servidores virtuais. Outras classes de serviço (*PaaS* e *SaaS*) oferecem menor flexibilidade e não são adequadas para computação de propósito geral, mas ainda podem fornecer um certo nível de personalização.

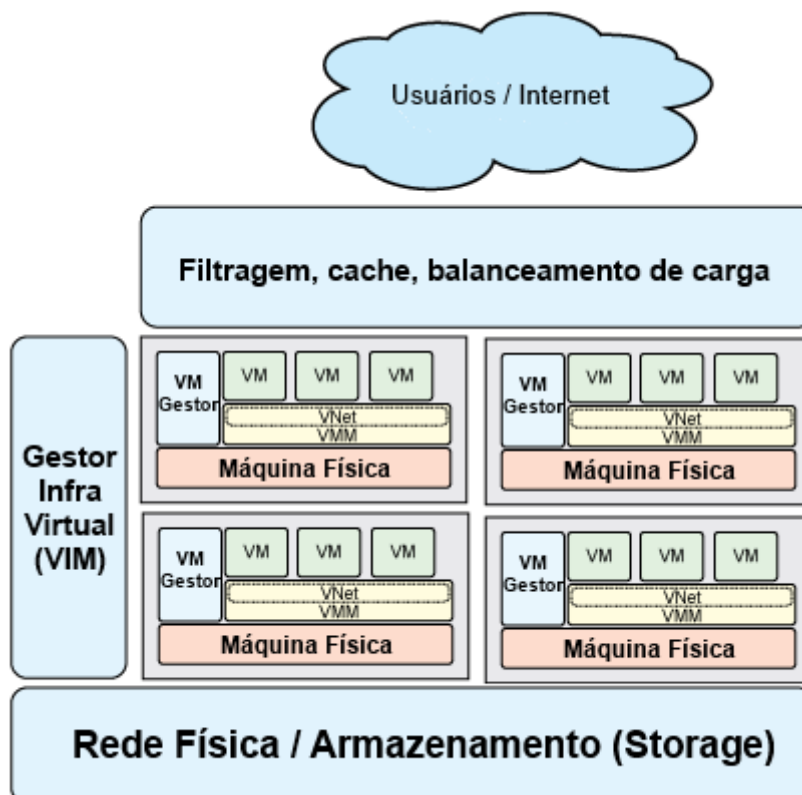
2.7 INFRAESTRUTURA DA COMPUTAÇÃO EM NUVEM

Conforme Figura 5, a infraestrutura *IaaS* de uma nuvem conta com o *VIM* (*Virtual Infrastructure Manager*), esse tipo de *software* se assemelha a um sistema operacional, porém este agrega recursos de vários computadores apresentando uma visão uniforme e aplicações para o usuário. Também pode ser conhecido como “*infrastructure sharing software*” e “*virtual infrastructure engine*” (BUYYA; BROBERG; GOSCISNSKI, 2011).

Segundo Buyya apud Sotomayor et al. (2011), propõe uma diferenciação entre duas categorias de ferramentas utilizadas para gerenciar as nuvens. A primeira categoria é chamada de “*cloud toolkits*” inclui *softwares* que expõem uma interface remota e segura para criar, controlar, visualizar e monitorar recursos, mas não se especializam em gestão de infraestrutura virtual. A Segunda categoria chamada “*Virtual Infrastructure Managers*” é capaz de gerenciar recursos avançados, tais como balanceamento de carga automático e consolidação de servidores. No entanto, os autores ressaltam que há uma superposição entre as categorias, “*cloud toolkits*” também podem gerenciar uma infraestrutura virtual, embora eles normalmente ofereçam recursos menos sofisticados do que gestores especializados nessa infraestrutura.

Sendo assim, um *software* de nuvem precisa ter algumas características básicas e avançadas presentes em um gestor, como poderá ser observado nas seções a seguir.

Figura 5 – Infraestrutura da Computação em Nuvem.



Fonte: JONES, 2012A.

2.7.1 SUPORTE PARA VIRTUALIZAÇÃO

Uma nuvem deve suportar múltiplos clientes com diferentes necessidades que serão dimensionadas para um servidor da infraestrutura. Recursos virtualizados podem ser dimensionados e redimensionados com certa flexibilidade. Estas características tornam a virtualização de *hardware*, a tecnologia ideal para criar uma infraestrutura virtual que abrigará vários clientes (BUYA; BROBERG; GOSCISNSKI, 2011). Deve ser capaz de criar e gerenciar vários grupos de *VM* em *clusters* virtuais. Esse recurso é importante para alocação de recursos sob demanda.

2.7.2 PROVISIONAMENTO DE RECURSOS SOB DEMANDA

Uma das características altamente desejável de um gestor é a

disponibilização de recursos onde o usuário possa adquiri-los sem a intervenção de um operador humano, como a criação de servidores, configurações de segurança e testes de *software* (BUYYA; BROBERG; GOSCISNSKI, 2011).

2.7.3 MÚLTIPLOS BACKEND HYPERVISORS

Alguns gestores devem fornecer uma camada uniforme de gestão, independentemente da tecnologia de virtualização utilizada. Esta característica é visível nos gestores *open-source*, que geralmente fornecem *drivers* para interagir com múltiplos *hypervisors*. Nesse sentido, o objetivo da *libvirt*² é fornecer uma *API* uniforme para gestores, que podem gerenciar domínios (uma *VM* ou uma instância de um sistema operacional) em nós virtualizados usando operações padrão para chamadas de *hypervisors* (BUYYA; BROBERG; GOSCISNSKI, 2011).

2.7.4 VIRTUALIZAÇÃO DE ARMAZENAMENTO

A virtualização de armazenamento (*storage*) consiste em abstrair armazenamento lógico a partir de um armazenamento físico. Ao consolidar todos os dispositivos de armazenamento disponíveis em um centro de dados, torna-se possível a criação de discos virtuais independentes do dispositivo e localização. *Storages* normalmente são organizados em *Storage Area Network* (*SAN*) e conectados através de protocolos como o *Fibre Channel*, *iSCSI* e *NFS* (BUYYA; BROBERG; GOSCISNSKI, 2011).

2.7.5 INTERFACE PARA NUVENS PÚBLICAS

Pesquisadores têm percebido que empréstimo de recursos a partir de nuvens públicas é vantajoso. Desta forma, as instituições podem fazer bom uso de seus recursos disponíveis e, em caso de picos de demanda, a carga extra pode ser transferida para recursos alugados, sendo assim, um gestor deve ser capaz de oferecer um controlador para o ciclo de vida dos recursos virtualizados obtidos a partir de fornecedores externos de nuvem. Para as

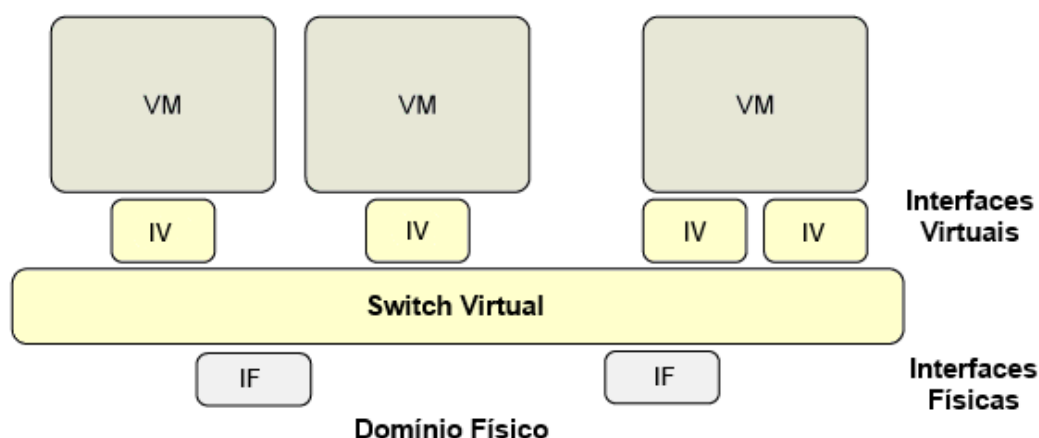
² *Libvirt* é uma *API open-source* para gerenciamento de ferramentas de virtualização, podendo ser usado para gerenciar *Linux KVM*, *Xen*, *VMware ESX*, entre outros *hypervisors*.

aplicações, a utilização dos recursos alugados deve ser transparente (BUYYA; BROBERG; GOSCISNSKI, 2011).

2.7.6 REDE VIRTUAL

As redes virtuais permitem a criação de uma rede isolada no topo de uma infraestrutura física. Uma *LAN Virtual (VLAN)* permite isolar o tráfego que compartilha uma rede comutada, permitindo que *VMs* sejam agrupadas no mesmo domínio de *broadcast*, como pode ser observado na Figura 6.

Figura 6 – Redes virtuais na Computação em Nuvem.



Fonte: JONES, 2012A.

Além disso, uma *VLAN* pode ser configurada para bloquear o tráfego originado de redes de outras *VMs*. Da mesma forma, o conceito de *VPN* (rede privada virtual) é usado para descrever uma rede de sobreposição segura e privada em cima de uma rede pública (mais comumente a Internet). Sendo assim, deve existir suporte para esse tipo de rede virtual em um gestor (BUYYA; BROBERG; GOSCISNSKI, 2011).

2.7.7 ALOCAÇÃO DINÂMICA DE RECURSOS

Em infraestruturas de nuvem, onde as aplicações possuem necessidades variáveis e dinâmicas, existe a necessidade de alocação dinâmica de recursos visando oferta e a demanda, redução do consumo

elétrico e uma melhor gestão dos *SLAs*³. Máquinas que não estão executando nenhuma instância de *VM* podem ser desligadas ou colocadas em estado de baixo consumo energético. Sendo assim, um gestor de *VI* deve ser capaz de alocação dinâmica de recursos, monitorando continuamente a utilização do seu parque de máquinas (BUYA; BROBERG; GOSCISNSKI, 2011).

2.7.8 MECANISMO DE RESERVA E NEGOCIAÇÃO

Quando os usuários solicitarem recursos computacionais para um momento específico, os pedidos são chamados de reservas antecipadas (*AR*), em contraste com o pedido por melhor esforço, em que recursos são disponibilizados a partir do momento em que estão disponíveis. Nesse caso um gestor deve permitir alocar recursos por períodos de tempo. Esta característica é ilustrada pelo caso em que uma *AR* é solicitada por um cliente, porém o provedor não pode oferecer exatamente essa *AR*, mas pode oferecer uma *AR* satisfatória para o cliente. Este problema foi abordado no *OpenPEX* (gestor de nuvem), que incorpora um protocolo de negociação bilateral que permite usuários e provedores uma negociação sobre o serviço (BUYA; BROBERG; GOSCISNSKI, 2011).

2.7.9 ALTA DISPONIBILIDADE E RECUPERAÇÃO DE DADOS

A alta disponibilidade de recursos de um gestor visa minimizar o tempo de inatividade do aplicativo e evitar a interrupção dos negócios. Alguns gestores realizam isso através de um mecanismo de *failover*⁴, que detecta a falha de servidores físicos e virtuais e reinicia as *VMs* em outros servidores físicos. Porém esse modelo de *failover* não funcionará para *VMs* de missão crítica, em que essas precisam ser redundantes e sincronizadas. O *backup* de dados em nuvens deve levar em conta o alto volume de dados, essas cópias devem ser feitas com uma interferência mínima no desempenho dos sistemas. Neste sentido, alguns gestores devem oferecer mecanismos de proteção de

3 *Service Level Management* – é um contrato entre duas partes, geralmente entre empresas e prestadores de serviços, que contém cláusulas de níveis de serviços que devem ser seguidas e obedecidas com tempos predefinidos, garantindo a qualidade e não interrupção do item contratado.

⁴ Segundo Gomes (2012), “O processo no qual uma máquina assume os serviços de outra, quando esta última apresenta falha, é chamado *failover*. O *failover* pode ser automático ou manual, sendo o automático o que normalmente se espera de uma solução de Alta Disponibilidade.”.

dados que executam *backups* incrementais de imagens de VMs (BUYYA; BROBERG; GOSCISNSKI, 2011).

2.8 PRINCIPAIS GESTORES DA COMPUTAÇÃO EM NUVEM

Nesse tópico serão abordados alguns dos principais gestores de infraestrutura virtual e suas características básicas.

2.8.1 APACHE VCL

Segundo VLC (2012), o *Apache VLC* é um laboratório de computação virtual, *open-source* utilizado para fornecer acesso remoto para um ambiente de computação dedicado para o usuário final através de uma interface *Web*, foi projetado em 2004 por pesquisadores da *North Carolina State University*, como forma de proporcionar ambientes personalizados para usuários do laboratório de informática.

Em resumo, *Apache VCL* oferece os seguintes recursos: controlador multiplataforma, baseado no *Apache/PHP*⁵, portal *Web* e interface *XML-RPC*⁶, suporte para *hypervisor VMware* (*ESX*, *ESXi*, e servidor), redes virtuais, *clusters* virtuais, e reserva antecipada de recursos.

2.8.2 CITRIX ESSENTIALS

O *Citrix Essentials* designado para *Microsoft Hyper-V* fornece aos clientes virtualização de servidores com um conjunto de serviços avançados de gerenciamento de virtualização, permitindo estender essas capacidades de gerenciamento para o ambiente *Windows Server 2008*, ajudando a tornar o *Hyper-V* e o *System Center* mais escaláveis, gerenciáveis e ágeis (CITRIX, 2012).

Segundo Buyya, Broberg e Goscisnski (2011), a ferramenta oferece os seguintes recursos: controlador gráfico *Web*, *CLI* (*Command Line Interface*),

⁵ *Apache*: servidor *Web open-source* e *PHP* (*HyperText PreProcessor*): linguagem de programação destinada ao desenvolvimento de páginas *Web*.

⁶ *XML-RPC*: chamada de procedimento remoto que utiliza *XML*.

portal *Web*, e *XML-RPC* (*eXtensible Markup Language – Remote Procedure Call*), apoio para *XenServer* e *Hyper-V*; Armazenamento *Citrix*; redes virtuais; alocação dinâmica de recursos; três níveis de alta disponibilidade (recuperação pelo reinício da *VM*, recuperação ativando uma *VM* duplicada pausada e *VM* duplicada rodando continuamente) e proteção de dados com o *Citrix Consolidated Backup*.

2.8.3 EUCALYPTUS

Segundo *Eucalyptus* (2012), o *Eucalyptus* se concentra na construção de nuvens *IaaS*. Ele foi desenvolvido com a intenção de fornecer uma implementação *open-source* quase idêntica em funcionalidade com a *API* do *Amazon Web Services*. Portanto, os usuários podem interagir com uma nuvem *Eucalyptus* usando as mesmas ferramentas que eles usam para acessar o *Amazon EC2*⁷.

Segundo Buyya, Broberg e Goscisnski (2011), ele também se distingue de outras ferramentas, pois fornece uma nuvem de armazenamento emulando a *API* do *Amazon S3*⁸ para armazenar dados do usuário e imagens gerais da *VM*. Em resumo, *Eucalyptus* oferece os seguintes recursos: administração *Web*; *EC2* (*SOAP*, *Query*) e *S3* (*SOAP*⁹, *REST*) *CLI*¹⁰ e interfaces *Web*, *Xen*, *KVM* e *VMWare backends*; *Amazon EBS*¹¹ compatíveis com dispositivos de armazenamento virtuais, interface com o *EC2* da *Amazon*, redes virtuais.

2.8.4 NIMBUS3

O conjunto de ferramentas *Nimbus* é construído em cima do *Globus Framework*. Ele fornece a maioria das características dos outros gestores como uma *API* para acesso ao *Amazon EC2* com suporte ao *Xen*. No entanto,

⁷ *Amazon EC2*: o *EC2* (*Elastic Compute Cloud*) é um serviço que permite o usuário alugar os recursos computacionais da *Amazon* e rodar máquinas virtuais sobre os centros de dados da empresa (AMAZON, 2012A).

⁸ O *Amazon Simple Storage* fornece uma interface simples de serviço *Web* que pode ser usada para armazenar e recuperar qualquer quantidade de dados, a qualquer momento, de qualquer lugar na *Web*. (AMAZON, 2012B)

⁹ *SOAP* (*Simple Object Access Protocol*) é um protocolo simples para troca de informação entre computadores, baseado no *XML*.

¹⁰ *Command Line Interface* é uma interface de linha de comando para entrada e saída de dados.

¹¹ O *Amazon Elastic Block Store* (*EBS*) fornece volume de armazenamento em bloco para instâncias do *Amazon EC2*. Os volumes *Amazon EBS* são armazenamentos fora da instância que

distingue-se dos outros, fornecendo uma interface de acesso para o *framework Globus Web Services Resource (WSRF)*. Em resumo, *Nimbus* fornece as seguintes características: baseado em *Linux*; *EC2 (SOAP)* e interfaces *WSRF*; suporte a *Xen* e *KVM* (BUYYA; BROBERG; GOSCISNSKI, 2011).

2.8.5 OPENNEBULA

O *OpenNebula* é um *open-source* rico em recursos para gestores de nuvem. Ao todo, quatro *APIs* de programação estão disponíveis: *XML-RPC* e *libvirt* para interação local; um subconjunto de *EC2* (consulta) e *APIs* da nuvem *OpenNebula* para acesso público. Sua arquitetura é modular e conectável, o módulo principal orquestra os servidores físicos e seus *hypervisors*, nós de armazenamento e de rede. O módulo *Scheduler* está encarregado de atribuir os pedidos pendentes de *VM* para hosts físicos, oferecendo alocação de recursos dinâmicos (OPENNEBULA, 2012B).

Segundo Buyya, Broberg e Goscisnski (2011), em resumo, *OpenNebula* oferece os seguintes recursos: baseado em *Linux*; *CLI*, *XML-RPC*, *EC2* e *OpenNebula Cloud API*; *Xen*, *KVM* e *VMware backend*; interface para as nuvens públicas (*Amazon EC2*, *ElasticHosts*); redes virtuais; alocação dinâmica de recursos; reserva antecipada de capacidade.

2.8.6 OPENPEX

Segundo Venugopal et al. (2009), *OpenPEX (Open Provisioning and EXecution Environment)* foi construído em torno da noção de usar reservas antecipadas como o principal método para alocar instâncias de *VM*. Distingue-se de outros gestores, pois incorpora um protocolo de negociação bilateral que permite usuários e provedores chegar a um acordo sobre troca de ofertas e contra ofertas quando seus pedidos originais não podem ser satisfeitos.

Segundo Buyya, Broberg e Goscisnski (2011), em resumo, *OpenPEX* oferece os seguintes recursos: multiplataforma (*Java*); portal *Web* e *Web Services (REST)*; *Citrix Backend XenServer*; reserva antecipada com capacidade de negociação.

persistem independentemente da vida de uma instância (AMAZON, 2012B).

2.8.7 OVIRT

Segundo OVIRT (2012), *oVirt* é um gestor *open-source*, patrocinado pela *Red Hat Emergent*. Ele fornece a maior parte das características básicas dos gestores, incluindo suporte para o gerenciamento de *pools* de servidores físicos, *pools* de armazenamento, contas de usuário e VMs. Todos os recursos são acessíveis através de uma interface *Web*, o nó de administração do *oVirt*, que também é uma VM, fornece um servidor *Web*, serviços de autenticação baseadas em *FreeIPA* (sistema integrado de segurança e gerenciamento de informações) e serviços de provisionamento para gerenciar imagem de VM e sua transferência para os nós gerenciados.

Em resumo, *oVirt* oferece os seguintes recursos: baseado no *Fedora Linux*, o controlador é empacotado como um dispositivo virtual, interface *Web*; *KVM backend* (BUYYA; BROBERG; GOSCISNSKI, 2011).

2.8.8 VMWARE VSPHERE

Segundo Buyya, Broberg e Goscisnski (2011), o *VMWare vSphere* é uma ferramenta destinada a transformar a infraestrutura em uma nuvem privada. Distingue-se de outros gestores como um dos mais ricos em recursos. Na arquitetura *vSphere*, servidores executam na plataforma *ESXi*. Um servidor a parte executa *vCenter Server*, que centraliza o controle da infraestrutura. Através do *software* cliente *vSphere*, os administradores conectam-se ao *vCenter Server* para executar várias tarefas. O *Distributed Resource Scheduler* (*DRS*) toma decisões de alocação com base em regras pré-definidas e políticas. Ele monitora continuamente a quantidade de recursos disponíveis para VMs e, se necessário, faz mudanças de alocação para atender aos seus requisitos. *VMFS vStorage* é um sistema de arquivos de cluster para agregar vários discos em um único volume. *VMFS* é especialmente otimizado para armazenar imagens de VM e discos virtuais, suportando equipamentos que utilizam *Fibre Channel* ou *iSCSI SAN*. O provisionamento de recursos aos usuários finais é fornecido através da *API vCloud*, que interage com o *vCenter Server*. Nesta configuração, *vSphere* pode ser usado por prestadores de serviços para construir as nuvens públicas.

Em resumo, *vSphere* oferece os seguintes recursos: baseado no

Windows (vCenter Server); CLI, GUI, Web portal, e interfaces de serviços Web; VMware ESX, ESXi backend; VMware; vStorage VMFS; interface para nuvens externas (VMware vCloud); redes virtuais (VMware Distributed Switch); alocação dinâmica de recursos (VMware DRM); alta disponibilidade; proteção de dados (VMware Consolidated Backup) (BUYA; BROBERG; GOSCISNSKI, 2011).

2.8.9 OPENSTACK ESSEX

Segundo OpenStack (2012), o gestor oferece *software* para criar nuvens públicas e privadas. Contém uma coleção de projetos *open-source*, incluindo *OpenStack Compute* (codinome *Nova*), armazenamento de objetos *OpenStack* (codinome *Swift*), serviço de imagem *OpenStack* (codinome *Glance*), serviço de armazenamento para uso das instâncias (codinome *Cinder*) e serviço de rede (codinome *Quantum*). O *OpenStack Essex* fornece uma plataforma operacional e um conjunto de ferramentas para gerenciamento das nuvens.

Em resumo o *OpenStack Essex* oferece gerenciamento do ciclo de vida das instâncias, gerenciamento dos recursos computacionais; rede e autorização; *API REST*; comunicação assíncrona, suporte a múltiplos *hypervisors*; *Xen*, *XenServer/XCP*, *KVM*, *UML*, *VMware*; *vSphere* e *Hyper-V*.

O *OpenStack Essex*, atualmente, é um dos mais importantes gestores de nuvem com a participação de mais de 200 empresas, incluindo a *NASA*, a qual cedeu o código-fonte do *Nebula* para construção da arquitetura.

As principais características dos gestores de nuvem são demonstradas na Tabela 1, em que é possível notar as diferenças entre os gestores proprietários e *open-source* estudados nesse capítulo.

TABELA 1 – COMPARAÇÃO ENTRE OS GESTORES DE NUVEM.

	Licença	Plataforma	Interface Cliente, API	Hypervisor	Virtualizaçã o Storage	Interfac e Cloud Público	Rede Virtual	Alocação Dinâmica de Recursos	Reserva Antecipa da	Alta Disponibilida de	Recuperação de Dados
Apache VCL	Apache v2	Multiplataforma (Apache/PHP)	Portal, XML- RPC	VMware ESX, ESXi, Server	Não	Não	Sim	Não	Sim	Não	Não
Citrix Essentials	Proprietári o	Windows	GUI, CLI, Portal, XML- RPC	XenServer, Hyper-V	Citrix Storage Link	Não	Sim	Sim	Não	Sim	Sim
Eucalyptus	BSD	Linux	EC2 WS, CLI	Xen, KVM	Não	EC2	Sim	Não	Não	Não	Não
Nimbus3	Apache v2	Linux	EC2 WS, WSRF, CLI	Xen, KVM	Não	EC2	Sim	Integrado com OpenNebula	Não	Não	Não
OpenNebula	Apache v2	Linux	XML-RPC, CLI, Java	Xen, KVM	Não	Amazon EC2, Elastic Hosts	Sim	Sim	Sim	Não	Não
OpenPEX	GPL v2	Multiplataforma (Java)	Portal, WS	XenServer	Não	Não	Não	Não	Não	Não	Não
oVirt	GPL v2	Fedora Linux	Portal	KVM	Não	Não	Não	Não	Não	Não	Não
VMWare vSphere	Proprietári o	Linux, Windows	CLI, GUI, Portal, WS	VMware ESX, ESXi	VMware vStorage VMFS	VMware vCloud	Sim	Sim	Sim	Sim	Sim
OpenStack	Apache v2	Linux	EC2 WS, CLI	KVM, LXC, QEMU, UML, VMWare ESX/ESXi 4.1 e	Sim	EC2, Vmware	Sim	Sim	Sim	Sim	Não

Fonte: adaptado de Buyya, Broberg e Goscinski (2011).

2.9 LIMITAÇÕES ENCONTRADAS COM OS GESTORES

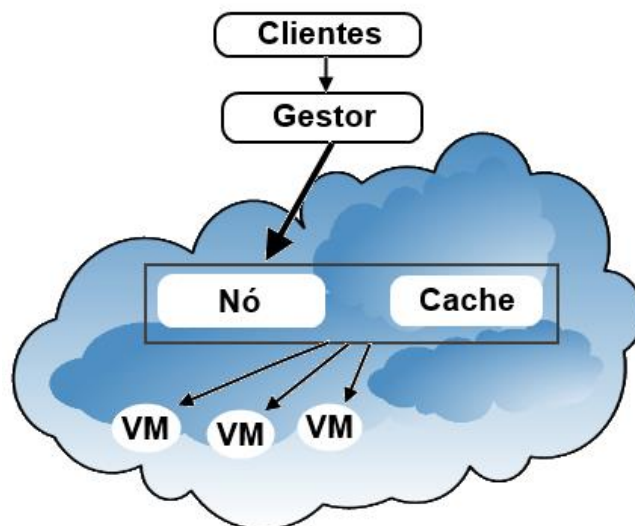
Para um melhor entendimento do conceito e arquitetura da nuvem, foram testados três gestores, *Eucalyptus* (Apêndice A - Instalação do *Eucalyptus*), *OpenNebula* (Apêndice B - Instalação do *OpenNebula*) e *OpenStack Essex* (Apêndice C - Instalação do *OpenStack Essex*), escolhidos pela popularidade e por serem *open-source*. Nos testes foi verificado que os gestores se comportam da mesma forma e possuem praticamente os mesmos serviços com algumas diferenças, na qual o *OpenStack Essex* mostrou melhor eficiência, sendo assim, aqui será demonstrada sua arquitetura e funcionamento.

Após a realização de testes foi possível levantar os principais problemas da arquitetura *open-source*.

- **Failover:** o *Eucalyptus* não tem *failover*, tanto para o gestor quanto para o nó. Por exemplo, se o gestor sofrer algum problema, os nós continuarão funcionando normalmente, porém se o nó apresentar alguma indisponibilidade, a execução daquela instância será perdida, devendo um operador humano intervir, carregando a instância em outra VM, o mesmo foi observado com o *OpenNebula* e *OpenStack Essex*. Segundo Chauhan et al (2012), uma solução para contornar o problema de indisponibilidade é criar um sistema de *cache* para nuvem, conforme Figura 7. Porém essa solução aumentaria o consumo energético, uma das principais vantagens da Computação em Nuvem.
- **Algoritmo de Escalonamento:** o *Eucalyptus* possui três algoritmos para escalonamento de instâncias, o primeiro é o *ROUNDROBIN* (escolhe um nó de cada vez), o segundo *GREEDY* (aloca a instância no primeiro nó encontrado) e o terceiro é o *POWERSAVE* (desliga os nós que não estão executando instâncias de máquinas virtuais), tanto o *OpenNebula* quanto o *OpenStack Essex* também possuem algoritmos locais similares. Nesse caso, o gestor poderia determinar de modo eficiente e técnico quem são seus nós e realizar o escalonamento de acordo com o consumo da aplicação ou reescalonar caso perceba que um nó esteja excedendo quantidade utilizada de memória e CPU.

- **Distâncias Geográficas:** no caso de nuvens geograficamente distribuídas, o gestor deveria conseguir reescalonar uma determinada instância de VM de acordo com a região na qual tem mais acessos. Um dos problemas de virtualizar uma aplicação, é que na maioria das vezes ela não está preparada para ser acessada através de uma rede, sendo assim, acessar uma aplicação geograficamente distante, geraria custo e consumo de banda.
- **Consumo Energético:** nós que não estão executando instâncias deveriam estar desligados ou colocados em baixo consumo energético, para isso, o gestor deve fornecer algoritmos de escalonamentos que consigam ativar ou desativar nós de acordo com a demanda. No *Eucalyptus* foi analisado que o algoritmo *PowerSave* realiza esse procedimento.
- **Proteção de Dados:** no *Eucalyptus* e em qualquer outra nuvem *open-source*, não existe política e controle de *backup*. Outro problema encontrado é que a nuvem *open-source* utiliza de soluções de terceiros para armazenamento, como *NFS* ou *SAN*. Nesse caso garantir a redundância das informações torna-se muito dispendioso para a rede, que no mínimo deverá trabalhar a 10 Gbps para ter um desempenho considerável.

Figura 7 – Modelo de nuvem utilizando cache.



Fonte: CHAUHAN et al., 2012.

2.10 CONSIDERAÇÕES FINAIS

Nesse capítulo foram apresentados os gestores analisados em laboratório, *Eucalyptus*, *Nebula* e o *OpenStack Essex*. Dos três gestores estudados, o *Eucalyptus* e o *Nebula* contêm as mesmas funcionalidades e limitações apresentadas na Seção 2.9. O *OpenStack Essex* foi o que apresentou os melhores resultados com alguns pontos positivos como, documentação completa, comunidade ativa, preocupação com correção de *bugs*, instalação facilitada e uma variedade de imagens prontas para instâncias.

Os três gestores apresentaram deficiências em disponibilizar recursos para virtualização de armazenamento, alta disponibilidade, recuperação de dados e algoritmos alternativos para escalonamento de máquinas virtuais, objeto desse estudo.

3 ESCALONAMENTO DE RECURSOS

Nesse capítulo são analisados os principais algoritmos de escalonamento e os algoritmos de escalonamento utilizados nos principais gestores *open-source* da Computação em Nuvem (*Eucalyptus*, *OpenNebula* e *OpenStack Essex*).

3.1 CONSIDERAÇÕES INICIAIS

Um dos problemas mais desafiadores da computação paralela e distribuída é conhecido como problema de escalonamento. O objetivo do escalonamento é determinar uma atribuição de tarefas a elementos de processamento com o intuito de otimizar certos índices de desempenho, como tempo processamento, leitura e escrita em disco ou memória (EL-REWINI; ABD-EL-BARR, 2005).

Deve-se ressaltar que existem duas espécies de escalonamento dentro da arquitetura:

- **Escalonador do gestor:** os algoritmos de escalonamento trabalham com a finalidade de escalonar instâncias de máquinas virtuais para nós computacionais, responsáveis pelo processamento.
- **Escalonador do *hypervisor*, dentro do nó computacional:** o algoritmo de escalonamento está presente no sistema operacional da máquina física, compartilhando o processamento entre seus processos.

Conforme pode ser observado na Figura 8, para uma nuvem *IaaS* que utiliza o *KVM* como *hypervisor*, as instâncias de máquinas virtuais são carregadas no *Linux* como um processo normal de usuário. Nesse caso, o escalonador padrão do *Linux* é que determina a execução das instâncias, conforme seu algoritmo de escalonamento.

Figura 8 – Listagem de processos do KVM no Linux.

107	2395	1	2	May27	?	03:24:06	/usr/bin/kvm	-S
root	2396	2	0	May27	?	00:00:00	[kvm-pit-wq]	
root	2676	2217	0	May22	pts/1	00:00:00	sg libvirtd /opt	
stack	2685	2150	0	May22	pts/2	00:00:00	/bin/bash	
107	2841	1	2	May27	?	03:18:10	/usr/bin/kvm	-S
root	2842	2	0	May27	?	00:00:00	[kvm-pit-wq]	

Fonte: Autor.

3.2 ESCALONAMENTO

Segundo Silva (2012), em sistemas que empregam a multiprogramação, processos concorrem pelo processamento, em que se faz necessário dividir o tempo de execução de cada um deles, sendo feito de duas maneiras: escalonamento dos processos a curto prazo e escalonamento dos processos a longo prazo. No caso do escalonamento dos processos a curto prazo, o escalonador decide quando o processo será criado. Ele pode esperar, por exemplo, a carga de processamento da máquina diminuir para criar novos processos.

Esse tipo de escalonamento pode não ser atraente para usuários que estão esperando a efetivação daquele processo, ou no caso da nuvem, usuários que estão esperando pela sua instância de máquina virtual ou outro serviço.

Ainda segundo Silva (2010), no escalonamento de curto prazo, o processo é executado sempre que o processador ficar livre, nesse caso, o escalonador é requisitado com grande frequência.

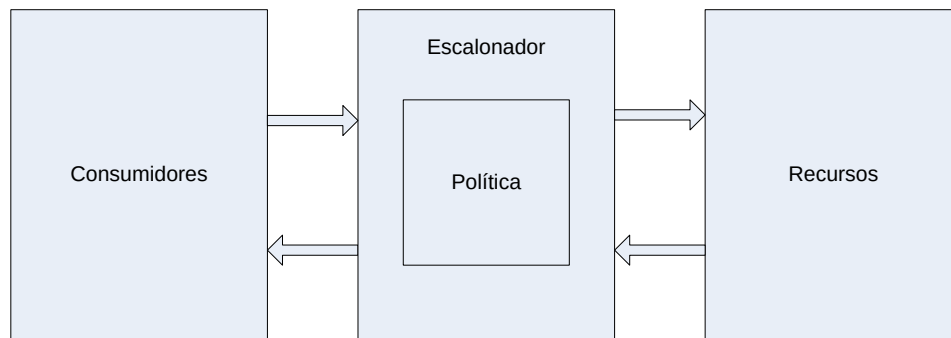
O escalonador caracteriza-se como um mecanismo de gestão, utilizado para gerenciar de forma eficiente e eficaz o acesso aos recursos por seus vários consumidores. Segundo a Figura 9, este mecanismo é constituído por três componentes principais: consumidores, recursos e política (CASAVANT; KUHL, 1988).

Entretanto, há duas propriedades que devem ser avaliadas em quaisquer sistemas de escalonamento de recursos:

- A gerência de recursos deve satisfazer os consumidores;
- Consumidores não devem ser prejudicados por problemas associados

ao uso do escalonador.

Figura 9 – Sistema de escalonamento.



Fonte: Adaptado de: CASAVANT; KUHL (1988).

Em relação às nuvens uma terceira propriedade deve ser adicionada para satisfazer também os provedores de serviços, que oferecem os serviços aos consumidores. Sendo assim:

- A gerência de recursos deve satisfazer os provedores de serviços;

Para que essas duas propriedades (de consumidores e provedores de serviços) não se sobreponham, devem existir acordos de nível de serviços entre elas, também conhecidos como *SLAs*. Esses acordos garantem que os recursos estarão disponíveis para os consumidores ou parte deles, no qual os consumidores aceitam participar de um ambiente compartilhado ou não, dependendo do tipo de acordo estabelecido.

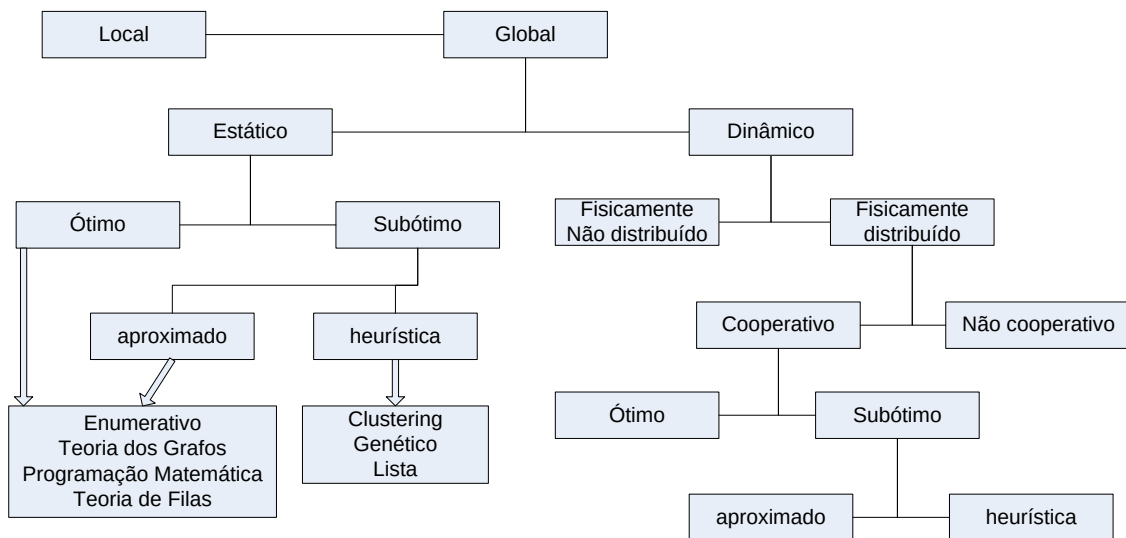
3.3 TAXONOMIA DE ESCALONAMENTO

Casavant e Kuhl (1988), conforme Figura 10, criou uma taxonomia para classificar os diferentes tipos de escalonamento, com o intuito de fornecer um mecanismo para permitir comparações entre trabalhos na área da computação distribuída de forma qualitativa ou em qualquer conjunto de sistemas de gestão de recursos:

- **Local:** atribuição de tarefas por slots de tempo para um único processador.
- **Global:** incide em onde executar as tarefas em um conjunto de processadores.

- **Estático:** também conhecida como programação determinística, todas as informações sobre as tarefas a serem executadas e suas relações com outras tarefas são totalmente conhecidas antes da execução do programa (EL-REWINI; ABD-EL-BARR, 2005).

Figura 10 – Taxonomia de Escalonamento.



Fonte: Adaptado de: CASAVANT (1988) apud RUSANOVA, KOROSCHKIN (1999).

- **Dinâmico:** conhecido também como escalonamento não-determinístico, algumas informações podem não ser conhecidas antes da sua execução (EL-REWINI; ABD-EL-BARR, 2005).
- **Ótimo:** quando é possível determinar todas as informações sobre a tarefa, como por exemplo, custo de comunicação, de execução e dependências de tarefas.
- **Subótimo:** quando essas informações sobre tarefa não são possíveis de determinar, porém encontra-se uma solução aceitável.
- **Aproximado:** é satisfeito quando é encontrada uma boa solução para um determinado problema, ao invés de procurar uma solução ótima em todo o conjunto de soluções.
- **Heurística:** estão presentes nessa categoria os algoritmos que fazem as suposições mais realistas a respeito das características dos processos e de carga do sistema.
- **Não distribuído:** a responsabilidade do escalonamento de tarefas

reside em um único processador.

- **Distribuído:** o trabalho envolvido na tomada de decisões deve ser distribuído entre os processadores.
- **Não cooperativo:** processadores tomam decisões individualmente, independentes do efeito no resto do sistema.
- **Cooperativo:** nesse caso, há uma cooperação entre os componentes distribuídos.

Existem ainda algumas classificações de escalonamento que segundo Casavant e Kuhl (1988), não se encaixam exclusivamente na taxonomia, mas são importantes como descrevem um escalonador:

- **Adaptativo:** quando os algoritmos e parâmetros mudam dinamicamente de acordo com o comportamento anterior para se ajustar as novas condições do sistema.
- **Não adaptativo:** ocorre quando o mecanismo de controle não sofre alteração, independente do comportamento do sistema.
- **Balanceamento de carga:** tenta prover o equilíbrio de carga entre os processadores do sistema.
- **Bidding:** nesse tipo de política, há uma cooperação entre os nós que executarão as tarefas, no sentido de troca de informações, conforme o sistema cooperativo.
- **Probabilístico:** analisar uma grande quantidade de tarefas antes de escalonar pode resultar em uma perda de tempo por parte do processador. Para compensar esse problema, processos são escolhidos aleatoriamente, e entre esses processos, os que apresentam os melhores resultados são escolhidos.

Existem diversas propriedades que podem fomentar um algoritmo de escalonamento. Nessas condições torna-se essencial analisar e investigar quais os objetivos que este deve possuir para atender um serviço ou tarefa. Atualmente, os algoritmos de escalonamento de nuvens *open-source* são determinísticos, as informações para determinar o nó computacional com mais recursos não levam em consideração as condições físicas do nó computacional

no momento do escalonamento.

Segundo Dandamudi (1994), o escalonamento de processos tem um impacto substancial no desempenho quando as estratégias de escalonamento não-adaptativo são utilizadas. Assim, se faz necessário criar um escalonador dinâmico e adaptativo, onde segundo El-Rewini; Abd-El-Barr (2005), este é normalmente implementado como uma espécie de heurística de balanceamento de carga. Porém, alguns problemas devem ser verificados, como a sobrecarga gerada para determinar o escalonamento enquanto o programa está sendo executado ou como medir o custo de cada processo.

3.4 BALANCEAMENTO DE CARGA

O balanceamento de carga tem como característica distribuir a carga de trabalho entre vários nós situados no *cluster*, com a finalidade de alcançar uma utilização simétrica ou ideal dos recursos, evitando sobrecarga de um ou mais nós computacionais. Na nuvem, o balanceamento de carga tem uma função importante em que ajuda a garantir a qualidade de serviços prestados aos usuários em relação à disponibilidade de recursos (SIDHU; KINGER, 2013).

3.4.1 CLASSIFICAÇÃO DOS ALGORITMOS DE BALANCEAMENTO DE CARGA

Segundo Sidhu e Kinger (2013), os algoritmos de balanceamento de carga são divididos em duas principais categorias:

- Dependendo da forma como a carga está distribuída e como os processos são atribuídos aos nós (carga do sistema):
 - **Centralizado**: um único nó é responsável por gerenciar a distribuição de carga;
 - **Distribuído**: cada nó é independente e as decisões são tomadas localmente de acordo com as informações de cargas de outros nós.
 - **Mista**: a combinação entre as duas formas anteriores.
- Dependendo das informações de estado dos nós (topologia do sistema).

- **Dinâmico:** leva em conta o estado atual do sistema durante as decisões de balanceamento;
- **Adaptativo:** nesse modelo, o balanceamento de carga se adapta quando o estado do sistema muda, alterando seus parâmetros e até seus algoritmos de forma dinâmica.

3.4.2 ALGORITMOS DE BALANCEAMENTO DE CARGA

Existem diversos algoritmos de balanceamento de carga propostos, Sidhu e Kinger (2013) apontam os principais:

- **Connection Mechanism:** nesse caso, o algoritmo de balanceamento de carga determina a quantidade de conexão por nó e faz o balanceamento de acordo com esse número.
- **Randomized:** nesse tipo, um processo pode ser tratado por um determinado nó n , com uma probabilidade p , para diferenciar cargas de diferentes complexidades.
- **Equally Spread Current Execution Algorithm:** nesse tipo, a carga de máquinas sobrecarregas é distribuída aleatoriamente através da checagem da carga.
- **Throttled Load Balancing Algorithm:** esse algoritmo é voltado para máquinas virtuais, em que é solicitado ao balanceador encontrar um nó adequado para realizar a operação desejada.
- **Task Scheduling Algorithm Based on Load Balancing:** é composto de um mecanismo de agendamento de tarefas em dois níveis baseado em balanceamento de carga para atender às necessidades dinâmicas dos usuários e obter alta utilização de recursos.
- **Max-Min Algorithm:** a máquina que possui o tempo mínimo de conclusão de todos os trabalhos é selecionada.
- **Scheduling strategy on load balancing of virtual machine resources:** faz o balanceamento de processos utilizando dados históricos e dados do estado atual do sistema para escalonar VMs (KANSAL, CHANA, 2012).

Conforme analisado, há alguns algoritmos propostos para balanceamento de carga de processos em sistemas distribuídos. O gestor de nuvem, além de decidir sobre qual nó deverá alocar o novo pedido de instância de VM, deve também:

- Ter a capacidade de redistribuir a carga do *cluster*, garantindo a disponibilidade dos serviços de usuários.
- Ou realocar instâncias para nós com capacidades de processamento livre, desligando nós ociosos, ajudando na redução do consumo de energia e, consequentemente, na redução de emissão de carbono (KANSAL, CHANA, 2012).

3.4.3 MÉTRICAS PARA BALANCEAMENTO DE CARGA

Segundo Sidhu e Kinger (2013), existem alguns indicadores para balanceamento de carga na nuvem que devem ser analisados:

- **Escalabilidade:** é a capacidade de executar um algoritmo para equilíbrio de carga de um sistema com um número finito de nós.
- **Utilização de Recursos:** é utilizado para verificar a utilização de recursos. Ele deve ser otimizado para um balanceamento de carga eficiente.
- **Desempenho:** é usado para verificar a eficiência do sistema. Deve ser melhorado a um custo razoável, por exemplo, reduzir o tempo de resposta de tarefas, mantendo atrasos aceitáveis.
- **Tempo de Resposta:** é a quantidade de tempo necessário para responder através de um algoritmo de balanceamento de carga.
- **Sobrecarga Associada:** determina a quantidade de sobrecarga envolvida durante a implementação de um algoritmo de balanceamento de carga. Essa sobrecarga é gerada pelo movimento de VMs e comunicação entre nós.

3.5 ESCALONAMENTO E BALANCEAMENTO NO OPENSTACK

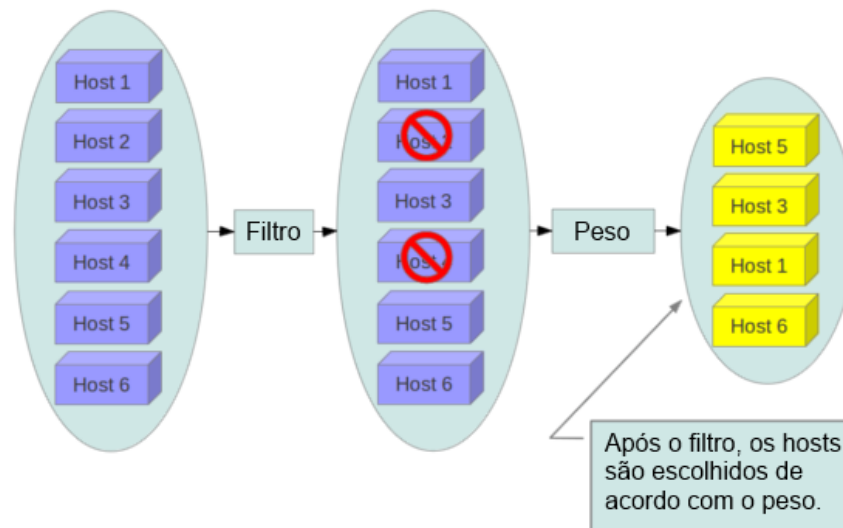
O *OpenStack Essex* implementa os algoritmos em linguagem Python, sendo o binário ***nova-scheduler***, responsável por calcular e decidir qual nó deverá lançar uma instância de imagem de máquina virtual, entre outras responsabilidades.

A importância do escalonador é que em tais ambientes de nuvem, muitas vezes há mais de um nó para carregar uma instância de imagem de máquina virtual. Como gerenciar e medir esses nós é um problema muito importante, assim como reagir a um pedido de usuário para alocação da instância também.

O *nova-scheduler* interage com outros componentes através de uma fila (*Queue*) e *nova-database* (banco de dados do gestor), no caso do agendamento, a fila é o centro de comunicação. Todos os nós publicam periodicamente o seu estado, os recursos disponíveis e as capacidades de *hardware* para *nova-scheduler* através da fila. Assim o *nova-scheduler* recolhe esses dados e os utiliza para tomar decisões, de acordo com as solicitações de recursos (OPENSTACK, 2012).

- **Filter Scheduler:** segundo OpenStack (2012), O *filter* (*nova.scheduler.filter_scheduler.FilterScheduler*) é o agendador padrão para escalonar instâncias de máquinas virtuais. Ele suporta filtragem e ponderação para tomar decisões de onde uma nova instância deverá ser criada. Conforme pode ser observado na Figura 11, quando o escalonador recebe um pedido de um recurso, primeiro aplica filtros para determinar quais nós são elegíveis para alocar uma instância. Nós que são aceitos pelo filtro são então processados por um algoritmo diferente para decidir qual alocará o pedido. Existem dezenas de filtros preexistentes no *OpenStack Essex*, alguns realizando filtragem pela quantidade de núcleos do processador, memória, rede entre outros recursos presentes nos nós.

Figura 11 – Seleção de nós – *nova-scheduler*.



Fonte: OPENSTACK, 2012.

- Custos e Pesos:** o *nova-scheduler* seleciona os nós que permaneceram após a filtragem e aplica uma ou mais funções de custo para obter pontuação numérica para cada nó. Se houver múltiplas funções de custo, as pontuações são somadas. O escalonador seleciona o nó que tem o mínimo custo ponderado. De acordo com OpenStack (2012), o gestor tem três funções de custo:
 - `compute_fill_first_cost_fn_weight`: esta função de custo calcula a quantidade de memória livre (RAM) disponível no nó.
 - `nova.scheduler.least_cost.retry_host_cost_fn`: esta função de custo acrescenta um valor adicional para repetir o escalonamento para um nó que já foi usado para uma tentativa de agendamento prévio.
 - `nova.scheduler.least_cost.noop_cost_fn`: esta função de custo retorna 1 para todos os nós.
- Outros Escalonadores:** o gestor possui outros escalonadores que atuam em conjunto com os filtros para escalonar instâncias de máquinas virtuais:
 - `Chance Scheduler` (`nova.scheduler.chance.ChanceScheduler`): seleciona aleatoriamente um nó a partir de uma lista de anfitriões filtrados. É o padrão do gestor.

- *Multi Scheduler* (*nova.scheduler.multi.MultiScheduler*): contém diversas sub-rotinas de programação para escalonamento de nós. É o padrão de escalonamento de volumes no *OpenStack Essex*.
- **Agregadores de Nós:** os agregadores dividem os nós em uma zona de disponibilidade, para permitir uma programação avançada na criação de *pools* de recursos ou para definir grupos lógicos para a migração de nós.

3.6 CONSIDERAÇÕES FINAIS

Existem diversos algoritmos de escalonamento utilizados para determinar um melhor balanceamento de processamento e distribuição de recursos. Em nuvens *open-source*, os principais algoritmos são determinísticos, utilizando da pontuação para determinar o nó que será utilizado para processamento. Essa pontuação não leva em consideração as condições de recursos disponíveis na nuvem, podendo prejudicar seu desempenho e afetar os serviços entregues aos clientes pelos prestadores de serviços.

O escalonador atual da *OpenStack Essex* utiliza o algoritmo *roundrobin* para distribuir as instâncias de *VMs* entre o nós da nuvem. Caso deseje equilibrar essa carga, pesos são adicionados aos nós com mais recursos, porém fica a critério do operador decidir quais são esses nós.

Essa escolha pode influenciar diretamente no desempenho de uma *VM*, devendo esta ser realizada por medições constantes e com base técnica para decidir o modo de balanceamento. Simplesmente atribuir um peso, pode não ser a uma métrica eficaz e que condiz com a realidade. Por esse motivo, o algoritmo de escalonamento deve ser alimentado constantemente (*feedback*) com a situação da nuvem, para determinar a forma como vai atribuir uma *VM* a um determinado nó.

4 METODOLOGIA E PROTÓTIPO

Nesse capítulo é apresentada a metodologia e um protótipo de um algoritmo para escalonamento dinâmico de instâncias de máquinas virtuais na nuvem.

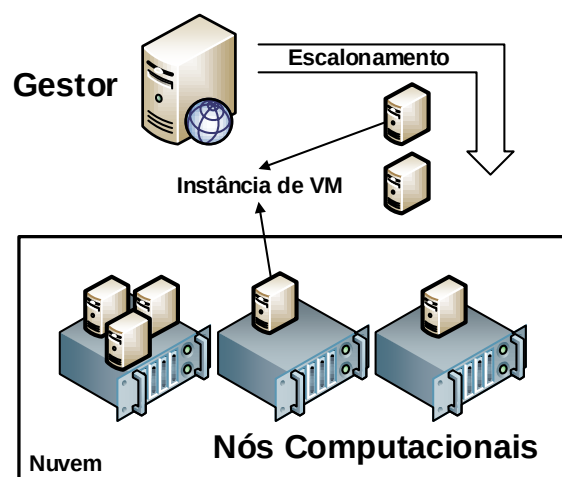
4.1 CONSIDERAÇÕES INICIAIS

O objetivo desse capítulo é discutir um algoritmo de escalonamento para nuvem e apresentar um algoritmo protótipo que seja capaz de verificar a carga de cada nó existente na nuvem privada, antes de alocar um pedido de instância de máquina virtual.

4.2 METODOLOGIA

Para o desenvolvimento desse trabalho foi construída uma nuvem. Conforme Figura 12, foram utilizados quatro computadores físicos, onde uma máquina foi utilizada como gestor (responsável por gerir a nuvem) e as demais máquinas como nós computacionais (máquinas físicas que abrigam e processam as máquinas virtuais).

Figura 12 – Escalonamento no gestor de nuvem.



Fonte: Autor.

A Tabela 2 descreve a situação inicial de cada nó, quantidade de memória, rede, *CPU*, quantidade de núcleos e o cache de processador em cada nó físico.

TABELA 2 – ESPECIFICAÇÕES DOS NÓS.

Nós	Escalonador	Rede	Memória	CPU Mhz	Cache Size	Núcleo
Nó 01	<i>Simple</i>	100 Mbps	4 GB	2000	3073 KB	4
Nó 02	<i>Simple</i>	100 Mbps	4 GB	2000	3073 KB	4
Nó 03	<i>Simple</i>	100 Mbps	4 GB	2000	3073 KB	4

A Tabela 3 demonstra o estado inicial dos nós em relação aos recursos disponíveis antes do teste de escalonamento. Conforme Tabela 3, todos os nós apresentam 99% de *CPU* disponível (levando em consideração que cada processador é composto de quatro núcleos), 100% de disponibilidade de disco e rede e média de 90% de disponibilidade de memória para alocação de instâncias de máquina virtual.

TABELA 3 – RECURSOS DISPONÍVEIS ANTES DO TESTE.

Nós	CPU %	Rede %	Memória %	Disco %
Nó 02	99.75	100	91	100
Nó 03	99.875	100	91.2	100
Nó 04	99.875	100	90	100

Para uma melhor observação dos resultados e prover uma comparação mais detalhada na fase de desenvolvimento, os algoritmos foram testados com três comportamentos de máquinas virtuais, na tentativa de simular um ambiente real de produção.

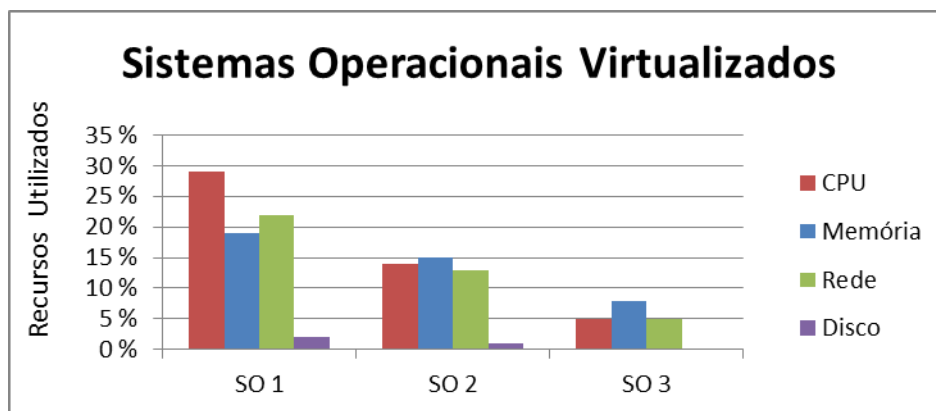
4.2.1 COMPORTAMENTO 1: MÁQUINAS VIRTUAIS DE CONSUMO CONSTANTE COM CARGAS DISTANTES

Foram criadas três imagens de máquina virtual com o sistema operacional *Ubuntu Server 12.04* em que na sua inicialização é executado o programa abaixo com um número diferente de *threads* para cada sistema operacional.

```
$threads = threads->new(\&stressDisk)
$threads = threads->new(\&stressNet)
$threads = threads->new(\&stressCPU)
```

Cada *thread* disparada pelo programa escrito em *Perl*, encontrado com maiores detalhes no Anexo A – `test_node_fx.pl`, fornece um consumo constante de recursos (processamento, rede e disco) conforme Figura 13.

Figura 13 – Sistemas operacionais virtualizados com cargas distantes.



Fonte: Autor.

O programa não precisou gerar um consumo para memória, pois quando uma VM é carregada, o gestor separa no sistema operacional hospedeiro a quantidade de memória desejada. Nesse caso, foram criados perfis de memórias dentro do gestor, especificando no script de lançamento o perfil desejado para cada VM.

Figura 14 – Perfis de memória.

```
root@controller:/openstack# nova flavor-list
```

ID	Name	Memory_MB
389e2d3f-726f-44e7-8ec4-fbf5a8f3493d	tipo3	256
a76bd1f9-64bf-4e0b-b881-7b211a984e47	tipo2	192
bac15e7f-76b7-433a-82e2-a3a18b79a976	tipo1	128

Fonte: Autor.

Foram criados três perfis de memória (tipo1, tipo2 e tipo3) cada um com, respectivamente, 128MB, 192MB e 256MB de consumo. Após os sistemas operacionais com cargas diferentes foram carregados para o gestor, que os disponibilizou para escalonamento como visto na

Figura 15.

Figura 15 – Sistemas operacionais disponíveis para o gestor.

```
root@controller:/openstack# nova image-list
```

ID	Name	Status	Server
e9f34d89-687f-4fbd-870e-18a24068d21d	ubuntu-12.10_1	ACTIVE	
cb02cc2f-623c-4a97-b197-24658781e2d8	ubuntu-12.10_2	ACTIVE	
8461c078-405a-469e-a81e-d7d1d95ad01d	ubuntu-12.10_3	ACTIVE	

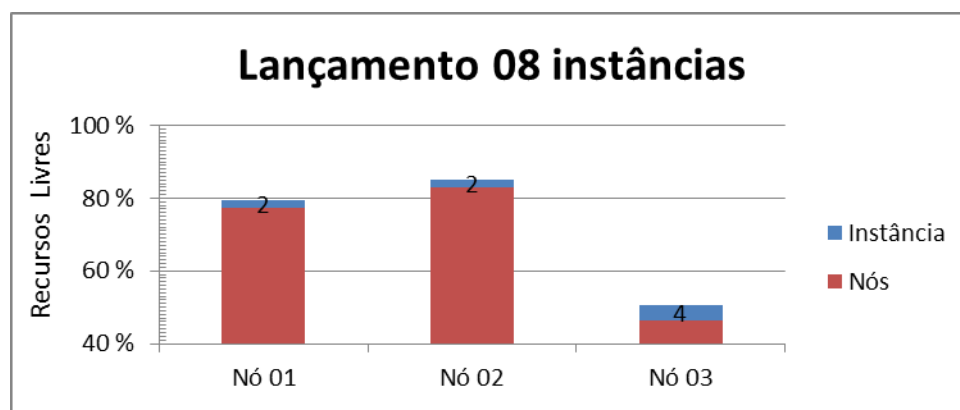
```
root@controller:/openstack#
```

Fonte: Autor.

Com as máquinas disponíveis para escalonamento, o próximo passo foi realizar os testes de escalonamento com o algoritmo atual do gestor *OpenStack Essex*. Para início do teste, foram lançadas oito instâncias de máquinas virtuais utilizando o programa abaixo que percorre um laço de um até oito, selecionando aleatoriamente um dos três sistemas operacionais criados com cargas diferentes e a quantidade de memória de acordo com a máquina selecionada.

Após a execução do programa, conforme pode ser observado pela Figura 16, os nós 1 e 2 são os que possuem mais recursos disponíveis, porém o nó 3 recebeu 4 instâncias, sobrecarregando-o mais que os outros.

Figura 16 – Teste de escalonamento – comportamento 1: 08 instâncias.



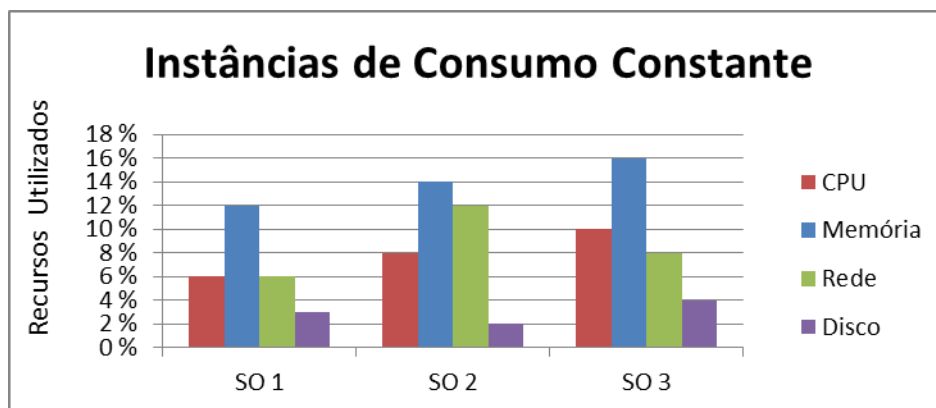
Fonte: Autor.

4.2.2 COMPORTAMENTO 2: MÁQUINAS VIRTUAIS DE CONSUMO CONSTANTE COM CARGAS APROXIMADAS

Para o teste com um segundo comportamento, foram criadas máquinas virtuais com consumo em pequena escala e com valores aproximados. Esse

teste foi criado para ser possível aumentar a quantidade de VMs escalonadas, visto que existiu uma restrição com o *hardware* utilizado nesse laboratório, em que as quatro máquinas físicas são de uso *desktop* e não servidores.

Figura 17 – Sistemas operacionais virtualizados com cargas aproximadas.

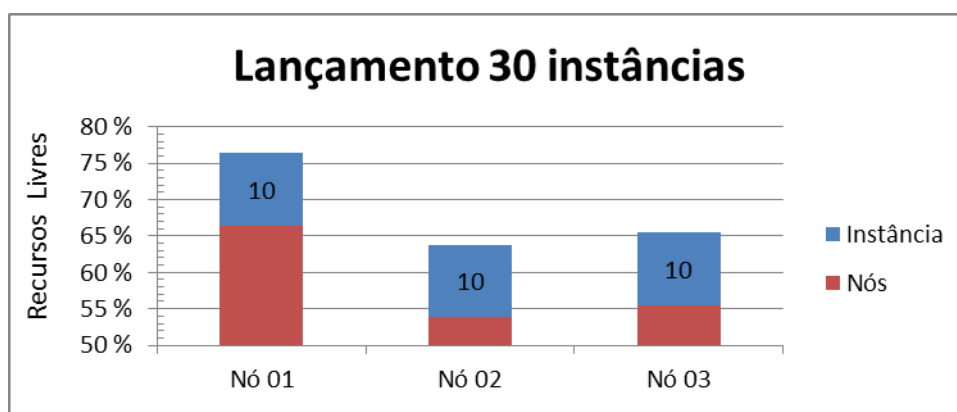


Fonte: Autor.

Conforme Figura 17, a diferença de carga entre os sistemas operacionais é pequena, permitindo que uma máquina física abrigue mais instâncias.

Após executar o pedido de escalonamento para trinta instâncias, foram obtidos os resultados vistos na Figura 18, onde todos os nós receberam a mesma quantidade de instâncias, porém o nó 02 teve seus recursos mais esgotados que o nó 1 e 3.

Figura 18 – Teste de escalonamento – comportamento 2: 30 instâncias.



Fonte: Autor.

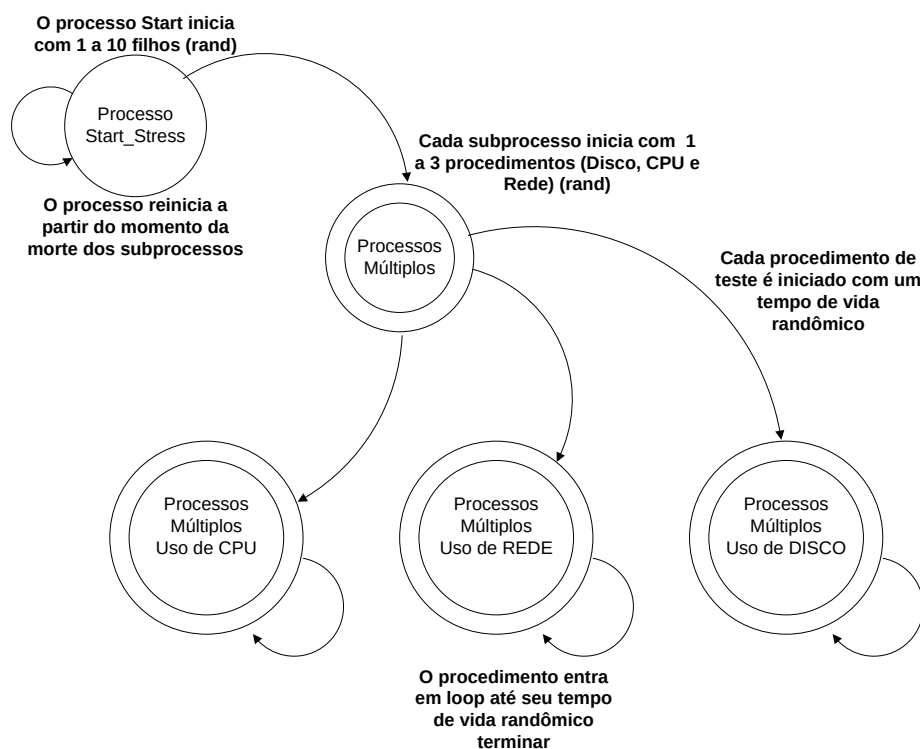
4.2.3 COMPORTAMENTO 3: MÁQUINAS VIRTUAIS DE CONSUMO VARIÁVEL

Nesse modelo de comportamento, foi criado um programa que fornece um consumo variável com o uso de *threads* que são disparados de modo randômico, conforme pode ser observado com maiores detalhes no Anexo B – test_node_rand.pl.

O programa gera uma quantidade de perfis de consumo (CPU, memória, disco e rede). Cada perfil inicia um novo *thread* com um determinado tempo de vida. Após o *thread* finalizar um novo programa é iniciado refazendo todo o processo.

Para facilitar a compreensão, foi criada a Figura 19, onde inicialmente o programa inicia de 1 a 10 *threads* e cada um desses *threads* iniciam de 1 a 3 novos *threads* contendo um consumo específico: CPU, disco e rede. Cabe ressaltar novamente que a memória não foi inserida na seleção, pois quando uma instância de máquina virtual é carregada, o KVM retira a porção de memória disponível da máquina física e aloca para a máquina virtual.

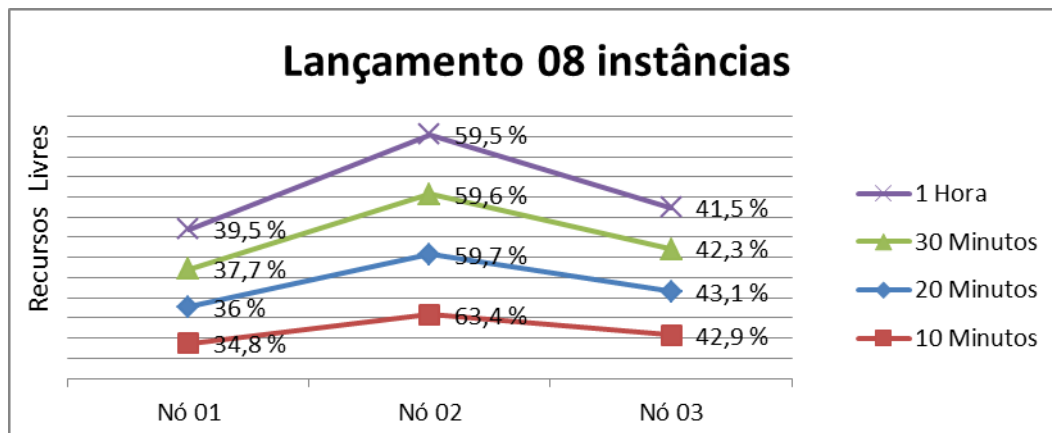
Figura 19 – Programa de simulação de consumo variável.



Fonte: Autor.

Após criar o sistema operacional virtualizado contendo o programa que gera o consumo aleatório, foram realizados oito escalonamentos de instâncias de máquinas virtuais.

Figura 20 – Teste de escalonamento – comportamento 3: 08 instâncias.



Fonte: Autor.

Conforme Figura 20, foram realizados medições aleatórias nos primeiros dez minutos, vinte minutos, trinta minutos e uma hora. Ocorreram variações de consumo de recursos, porém essas variações seguiram um padrão, permitindo estipular por média, a quantidade disponível de recursos dos nós.

4.3 O PROTÓTIPO

A nuvem possui um gestor de infraestrutura que conta com um escalonador de recursos. Diferentemente de um sistema operacional que, geralmente, lida com processos de baixa granularidade, o gestor da nuvem lida com instâncias de máquinas virtuais, se comparativamente a processos, pode-se dizer de alta granularidade. Assim, o escalonador da nuvem *IaaS* lida com pedido de alocação de instâncias de máquinas virtuais, devendo este determinar em qual nó alocar essa instância.

Em contrapartida de um processo comum de escalonamento, na nuvem a instância de máquina virtual permanece ativa, consumindo recursos ou não, até que uma ação (pedido do usuário ou falha *hardware/software*) interrompa seu processamento. Alguns itens devem ser avaliados antes do escalonador de nuvem tomar sua decisão sobre qual nó deverá alocar um novo pedido de

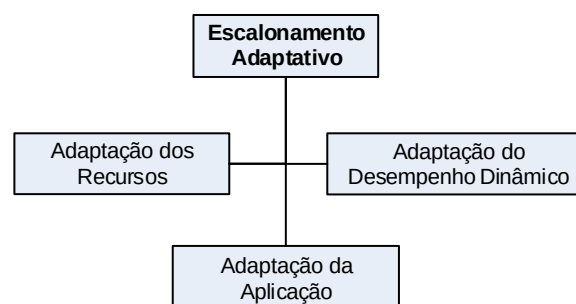
recursos, tais como:

- Capacidade livre de processamento do nó;
- Quantidade de memória total disponível;
- Quantidade de memória secundária disponível;
- Capacidade livre de leitura/gravação da memória secundária;
- Capacidade livre de *upstream* e *downstream* da rede;

Um dos principais problemas de escalonamento é determinar o custo de uma tarefa. Na nuvem encontra-se o mesmo problema de como determinar o custo de processamento, disco, memória e rede de uma instância de máquina virtual antes de ser escalonada. Nesses casos propõe-se utilizar um esquema adaptativo, em que os algoritmos e os parâmetros utilizados para decisões de escalonamento mudam dinamicamente de acordo com o estado de recurso anterior, corrente e/ou futuro (DONG;AKL, 2006).

Conforme pode ser observado na Figura 21, na computação em *Grid* que de certa forma torna-se similar com a nuvem, o escalonamento adaptativo é realizado com foco na heterogeneidade dos recursos de candidatos, o dinamismo do desempenho dos recursos, bem como a diversidade de aplicações (DONG;AKL, 2006):

Figura 21 – Taxonomia de escalonamento em *Grid*.



Fonte: DONG;AKL, 2006.

Segundo Casavant e Kuhl (1988), uma suposição de uma boa solução de escalonamento pode ser reconhecida nos casos em que está disponível uma métrica para avaliar uma solução, esta técnica pode ser utilizada para diminuir o tempo necessário para encontrar uma solução aceitável (programação). Os fatores que determinam esta abordagem incluem:

- A disponibilidade de uma função para avaliar uma solução.
- O tempo necessário para avaliar uma solução.
- A capacidade de julgar de acordo com algumas métricas o valor de uma solução ótima.
- A disponibilização de um mecanismo de inteligência para limitar o espaço soluções.

Para um ambiente de nuvem, o escalonador precisa avaliar as condições dos nós computacionais (aproximado ou heurístico) antes de alocar a próxima instância de máquina virtual (escalonamento estático), deverá selecionar o nó com mais recursos disponíveis, considerando que não é possível medir com precisão a quantidade de recursos que a nova instância necessitará (subótimo), medir periodicamente (adaptativo) e realocar as instâncias se necessário (balanceamento de carga), para não prejudicar o desempenho das demais instâncias presentes no nó em questão.

O algoritmo proposto é constituído de duas partes:

- A primeira deve avaliar constantemente os recursos livres de cada nó e guardar as informações.
- A segunda deve selecionar o nó com mais recursos no momento da instanciação de máquina virtual.

Assim, o algoritmo deve monitorar a quantidade livre de recursos em cada nó da nuvem privada, criar um índice para o gestor com os nós com mais recursos disponíveis e escalonar para nós com mais recursos, novas solicitações de instâncias de máquinas virtuais. Para que esse objetivo fosse atingindo, foi necessário dividir o algoritmo em dois módulos, um presente no gestor e o outro presente nos nós.

4.3.1 RECURSOS DOS NÓS

Foi criado um programa que monitora, em intervalos de tempo predefinidos a quantidade de recursos de cada nó (Anexo C – node_resources.py). Esse programa possui duas importantes funções:

- **resources:** essa função é executada apenas uma vez (quando o nó é iniciado) e através de análises de desempenho, determina

quais as capacidades máximas de transferência de dados do disco e da rede, quantidade de núcleos no processador, processamento e memória disponível.

- **check_node:** essa função é executada a cada cinco minutos armazenando em banco a quantidade de recursos disponíveis do nó.

O programa está presente em cada nó da nuvem e foi escrito em *Python* por ser a mesma linguagem utilizada pelo *OpenStack Essex*. Antes de executar as duas funções citadas acima, inicialmente o programa utiliza da função “*def check_net_velocity*” para analisar a taxa de transferência da *bridge*, necessária para comunicação entre nós e gestor e a função “*def check_disk_velocity*” para determinar a capacidade de leitura e gravação de disco. Essa análise ocorre apenas uma vez, quando a máquina física é ligada e o sistema operacional iniciado. Uma vez calculado esses parâmetros, não se faz necessário analisá-los novamente.

No caso específico do disco, para determinar a sua velocidade, é feito um teste de gravação e leitura para calcular as taxas de transferências. O valor referente a sua taxa de transferência foi armazenado em arquivo para não prejudicar a aferição dos recursos utilizados pelo nó, visto que há um consumo elevado justamente no momento de determinar sua taxa. Assim, esse processo ocorre uma vez quando o sistema operacional da máquina física é iniciado.

Após verificadas as capacidades de rede e disco, a função “*def resources*” informa para a função “*def check_node*” todos os recursos utilizados pelo nó, importantes para determinar o escalonamento. Para isso, foi necessária uma biblioteca do *Python* chamada *PSUTIL*, essa biblioteca contém funções que avaliam o uso de recursos como *CPU*, memória, rede e disco.

A função “*def check_node*” recebe por parâmetro os dados contendo o consumo atual do nó e através de operações simples, são calculadas porcentagens das quantidades livres de recursos para *CPU*, memória, disco e rede de acordo com a capacidade máxima do nó.

Com os resultados estabelecidos, as informações são armazenadas em um banco de dados, conforme estrutura apresentada na Figura 22.

Figura 22 – Tabela de armazenamento das informações dos nós.

```
mysql> desc node;
```

Field	Type	Null	Key	Default	Extra
id_node	int(5)	NO	PRI	NULL	auto_increment
node	varchar(30)	YES		NULL	
LA	float	NO		NULL	
CPU	float	NO		NULL	
MEM	float	NO		NULL	
NET	float	NO		NULL	
DISK	float	NO		NULL	
MED_LA	float	NO		NULL	
MED_CPU	float	NO		NULL	
MED_MEM	float	NO		NULL	
MED_NET	float	NO		NULL	
MED_DISK	float	NO		NULL	
MIN_LA	float	NO		NULL	
MIN_CPU	float	NO		NULL	
MIN_MEM	float	NO		NULL	
MIN_NET	float	NO		NULL	
MIN_DISK	float	NO		NULL	
record	datetime	YES		NULL	

18 rows in set (0.00 sec)

Fonte: Autor.

O algoritmo foi agendado para executar a cada cinco minutos. Esse tempo foi baseado no comando *uptime*, presente em sistemas *Linux* e *Unix*, que verifica a carga de processos nesses sistemas, conforme o último registro da Figura 23.

Figura 23 – Agendamento de execução de programa (*crontab*).

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts
#
*/5 * * * * root    /usr/bin/python /root/node_resources.py
```

Fonte: Autor.

4.3.2 ESCOLHA DOS NÓS

O segundo algoritmo está inserido no gestor. Este deve capturar todos os registos inseridos no banco de dados pelo script anterior, com as informações de recursos de cada nó, analisá-los e pontuá-los (Anexo D – `choice_node.py`).

Para isso, o algoritmo avalia os registos das condições das últimas 24 horas de um nó ativo, levando em consideração que este é apenas um protótipo, revisões futuras podem levar em consideração a parametrização de comportamento dos nós utilizando redes neurais ou outros algoritmos dinâmicos, conforme mencionado nas seções anteriores.

Para determinar o nó com mais recursos, primeiramente é calculada uma média simples dos registos e aplicados pesos nos recursos que mais podem influenciar o desempenho de uma máquina virtual. Cabe ressaltar que a definição desses pesos foi arbitrária nesse protótipo, mas em futuras aplicações deve-se analisar o comportamento das VMs e definir os pesos de acordo com esse resultado. Nesse teste foram priorizados pesos maiores para CPU e memória. Com base nos resultados, o algoritmo escolhe o nó com mais recursos disponíveis antes do lançamento da instância de máquina virtual.

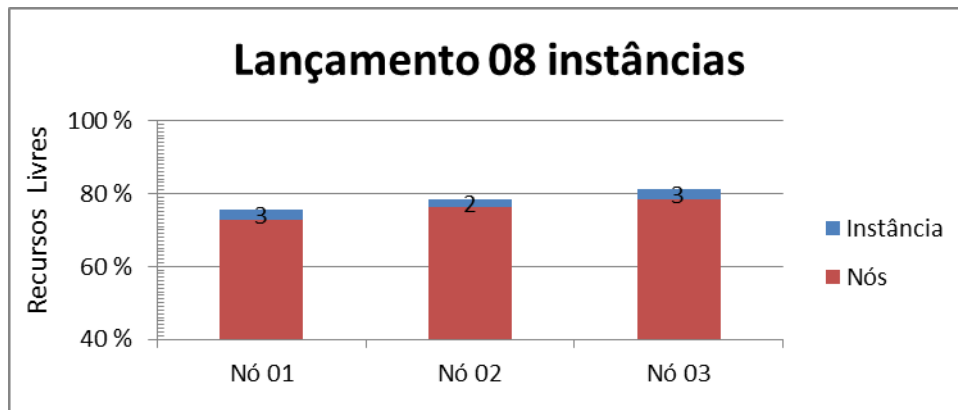
4.3.3 COMPORTAMENTO 1: MÁQUINAS VIRTUAIS DE CONSUMO CONSTANTE COM CARGAS DISTANTES

Foram realizados novos testes com o novo algoritmo, para isso o programa inicial de escalonamento do gestor foi alterado para instanciar a VM no nó com mais recursos. Sempre que uma solicitação de nó é requerida, o algoritmo desenvolvido é consultado para determinar o nó com mais recursos disponíveis.

Conforme pode ser observado na Figura 24, o protótipo conseguiu distribuir a carga entre os nós participantes da rede mais uniformemente do que o algoritmo atual da nuvem *OpenStack Essex*.

Assim, se faz necessário receber um retorno de cada nó participante da nuvem demonstrando sua capacidade real de recursos disponíveis. Para instâncias de consumo constante o protótipo atingiu seu objetivo em distribuir de maneira equacionada os recursos entre as instâncias.

Figura 24 – Teste do protótipo – comportamento 1: 08 instâncias.

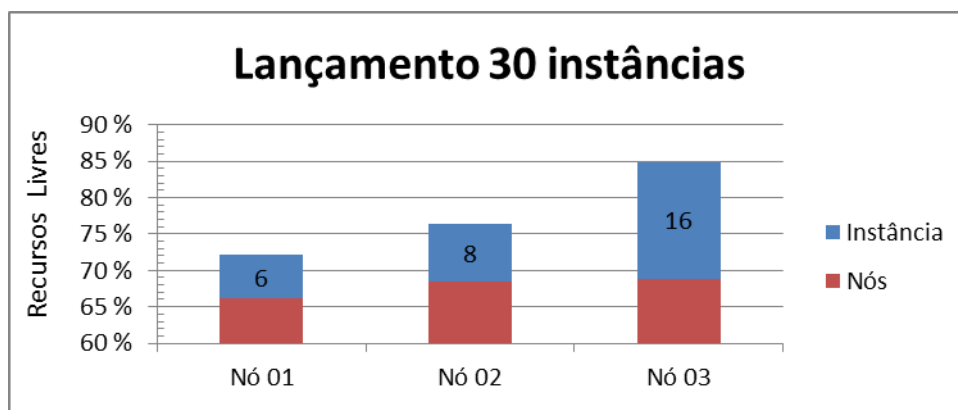


Fonte: Autor.

4.3.4 COMPORTAMENTO 2: MÁQUINAS VIRTUAIS DE CONSUMO CONSTANTE COM CARGAS APROXIMADAS

Nessa abordagem foram utilizadas trinta instâncias de VM com cargas aproximadas, com o intuito de aumentar a quantidade de instâncias nos testes. Foi escolhida essa quantidade para não sobrecarregar os nós, que são compostos por máquinas físicas de perfil *desktop*.

Figura 25 – Teste do protótipo – comportamento 2: 30 instâncias.



Fonte: Autor.

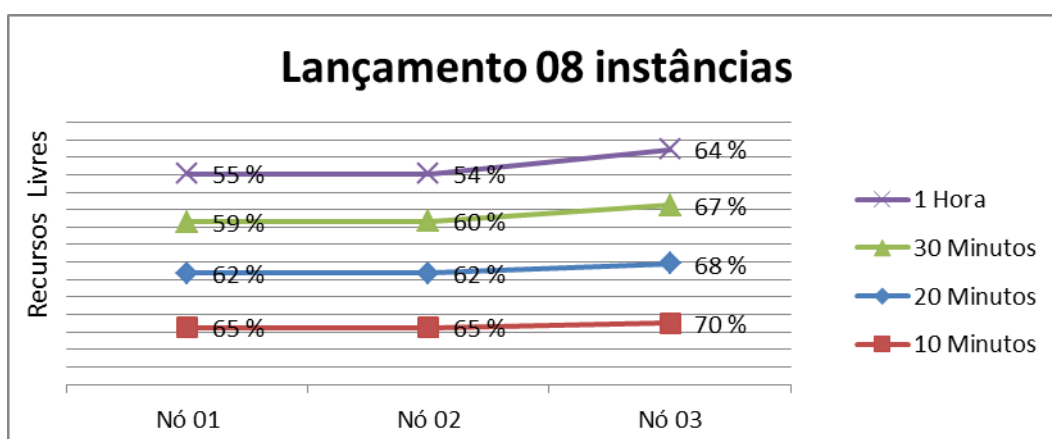
Assim como no teste do comportamento anterior, o protótipo funcionou conforme esperado, pois este avalia a condição dos nós antes de escalonar o pedido de VM. Conforme pode ser observado na Figura 25, o nó 03 recebeu 16

instâncias, o dobro se comparado ao nó 02, porém o custo de recursos para a máquina física foi equivalente para ambos os nós.

4.3.5 COMPORTAMENTO 3: MÁQUINAS VIRTUAIS DE CONSUMO VARIÁVEL

Para esse teste, foram utilizadas instâncias que alteram o padrão de consumo de recursos por um tempo aleatório, conforme já demonstrando no teste similar anterior na seção 4.2.3.

Figura 26 – Teste do protótipo – comportamento 3: 08 instâncias.



Fonte: Autor.

De acordo com a Figura 26, observa-se que calculando a média com pesos dos consumos das instâncias foi possível estabelecer um padrão que permitiu determinar a quantidade de recursos utilizados por cada nó. Ainda é possível verificar que o algoritmo conseguiu equilibrar a carga de recursos dentro da nuvem.

Assim, o algoritmo conseguiu avaliar as condições daquele nó para escalonamento no momento de um pedido de instância de VM.

4.4 CONSIDERAÇÕES FINAIS

Nessa seção foi demonstrada a metodologia utilizada para construção do ambiente de testes. Também foram demonstrados os testes com o algoritmo padrão do *OpenStack Essex* e o novo algoritmo com o intuito de prover uma comparação dos resultados.

Foram estabelecidos três comportamentos distintos de VMs para os testes de escalonamento no ambiente de nuvem. Após, as estruturas dos algoritmos foram apresentadas e em seguida realizados os escalonamentos de comparação. Conforme os resultados, o novo escalonador conseguiu atingir o objetivo inicial, avaliando as condições dos nós computacionais (aproximado ou heurístico) antes de alocar a próxima instância de máquina virtual (escalonamento estático).

5 CONCLUSÃO

5.1 CONSIDERAÇÕES INICIAIS

Esse trabalho apresenta a proposta e implementação de um algoritmo para escalonamento de máquinas virtuais dentro de uma nuvem privada. Assim, foi possível demonstrar que instâncias que se comportam como processos no sistema operacional, como é o caso do *hypervisor KVM* (que trata cada instância de *VM* como um processo de sistema), facilitam a análise e registro dos recursos consumidos, permitindo o cálculo da quantidade de recursos disponíveis para cada nó da nuvem.

Com esse algoritmo foi possível determinar pelo menos duas políticas base de escalonamento para pedidos de instâncias de máquinas virtuais, que justificam o uso da Computação em Nuvem:

- Distribuir a carga entre os nós da nuvem, melhorando assim a qualidade de serviço fornecido ou
- Alocar o máximo de instâncias para um nó, até o seu esgotamento de recursos, desligando nós não utilizados.

Dessa forma, esse protótipo inicial demonstrou que o retorno de processos que executam as instâncias de *VM* é fundamental para determinar com alguma precisão a capacidade de recursos de cada nó participante da nuvem, tornando possível para o gestor decidir entre qualidade de serviço e/ou economia energética como políticas de funcionamento do cluster.

Conforme apresentado na Seção 4.2.2 o algoritmo atual da nuvem *open-source* distribuir as *VMs* igualmente entre os nós da nuvem e não de acordo com a carga consumida por estas, sobrecarregando um ou mais nós da estrutura.

O protótipo desenvolvido demonstrou, satisfatoriamente de acordo com os resultados, que a carga foi balanceada dentro da estrutura criada. O que torna esse algoritmo mais atrativo para empresas ou organizações que desejam

maximizar os recursos ou garantir uma melhor qualidade de serviço para seus clientes.

Com esse trabalho, é possível criar novos algoritmos que façam análises das VMs como processos de sistema e integrá-los às nuvens *open-source*, tornando estas mais atrativas se comparadas às nuvens proprietárias, que se destacam pela gama de serviços oferecidos.

Além do desenvolvimento do protótipo, também foi apresentado o cenário da arquitetura de Computação em Nuvem no qual o algoritmo foi desenvolvido. A nuvem tem ganhado uma grande adesão, com um forte crescimento mundial. Foram definidos os tipos de nuvem, suas arquiteturas, modelos de funcionamento e serviços oferecidos. Alguns pontos sobre segurança foram abordados, pois são sensíveis a adoção integral da tecnologia em questão. As principais ferramentas de nuvem foram descritas focando em seus pontos mais relevantes. Após, foram criados três ambientes de testes, onde os gestores *Eucalyptus*, *OpenNebula* e *OpenStack Essex* foram instalados e configurados para análise. Com base nisso, alguns pontos de falhas importantes foram detectados e documentados. Também foram estudados os principais algoritmos de escalonamento de recursos e balanceamento de carga, auxiliando no desenvolvimento desse trabalho e na parametrização do novo algoritmo para nuvem.

Como bases nesses pontos, foi apresentada a metodologia e testes com o algoritmo de escalonamento atual do *OpenStack Essex*, demonstrando sua deficiência em distribuir, de modo simétrico, a carga entre os nós participantes da nuvem.

5.2 CONTRIBUIÇÕES

Com os estudos apresentados nesse trabalho, foi construído um protótipo de algoritmo de escalonamento de instâncias de máquinas virtuais para nuvem e os resultados obtidos demonstraram que é possível monitorar e controlar todas as instâncias da nuvem como processos e distribuí-las para melhorar a eficiência do serviço prestado ou agrupá-las por nó para minimizar o consumo

energético, um dos grandes problemas dos grandes centros de dados.

A pesquisa bibliográfica sobre Computação em Nuvem gerou dois artigos, um artigo apresentado em PDPTA'13 - The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications e o outro artigo apresentado na IV Escola Regional de Alto Desempenho de São Paulo.

A proposta e desenvolvimento do algoritmo gerou um terceiro artigo para PDPTA'14 - The 2014 International Conference on Parallel and Distributed Processing Techniques and Applications.

Como parte da contribuição desse trabalho, foi criado um fórum CCF (*Cloud Computing* Fórum – <http://www.cloudcomputingforum.com.br>) com o intuito de divulgar a arquitetura e disponibilizar material de apoio.

5.3 TRABALHOS FUTUROS

Para incentivar a continuação desse trabalho, alguns itens são apresentados para incrementar o protótipo proposto ou a criação de novas soluções:

- **Redes Neurais:** uma outra forma de avaliar o comportamento de cada instância de máquina virtual dentro da nuvem e determinar o melhor nó para realocação.
- **Auto desligamento dos nós inativos:** o protótipo poderia detectar nós inativos e desligá-los, economizando energia.
- **Auto reativação de nós inativos:** o protótipo deveria perceber a falta de recursos na nuvem e reativar nós inativos.
- **Escalonamento para nuvens públicas:** o algoritmo proposto visa trabalhar com nuvens privadas, não levando em consideração distâncias geográficas. Nesse caso, o protótipo deveria verificar a distâncias dos nós antes de alocar os novos pedidos de instâncias de máquinas virtuais.

REFERÊNCIAS

AMAZON. **Amazon Elastic Compute Cloud (Amazon EC2)**. Disponível em: <<http://aws.amazon.com/pt/ec2/>>. Acesso em: 30 abr. 2012A.

AMAZON. Amazon **Simple Storage Service (Amazon S3)**. Disponível em: <<http://aws.amazon.com/pt/s3/>>. Acesso em: 30 abr. 2012B.

BHADAURIA, Rohit; CHAKI, Rituparna. **A Survey on Security Issues in Cloud Computing**. *CORR, P.* abs/1109.5388, 2011.

BUYYA, Rajkumar; BROBERG, James; GOSCISNSKI, Andrzej M. **Cloud Computing: Principles and Paradigms**. John Wiley and Sons: San Francisco, 2011.

CASAVANT, Thomas L.; KUHL, Jon G. **A taxonomy of scheduling in general-purpose distributed computing systems**. *IEEE Transactions on Software Engineering*, New York, v. 14 n. 2, p.141-154, Feb. 1988.

CHAGANTI, Prabhakar. **Serviços em nuvem para sua infraestrutura virtual, Parte 1: Infrastructure-as-a-Service (IaaS) e Eucalyptus**. Disponível em: <<http://www.ibm.com/developerworks/br/library/os-cloud-virtual1/>>. Acesso em: 25 mar. 2012.

CHAGANTI, Prabhakar. **Xen Virtualization: A Practical Handbook**. Birmingham, UK: Packt Publishing Ltd., 2007.

CHAUHAN, V. Kumar; Bansal, K.; Alappanavar, P. **Exposing cloud computing as a failure**. *International journal of engineering science and technology* [0975-5462] Vikas Ano:2012 Vol:4 Nr:4

CITRIX. **Citrix Essentials for Hyper-V**. Disponível em: <http://www.citrix.com.br/products/citrix_essentials.php>. Acesso em: 30 abr. 2012.

CSA, *Cloud Security Alliance*. **SECURITY GUIDANCE FOR CRITICAL AREAS OF FOCUS IN CLOUD COMPUTING**. Disponível em: <<https://cloudsecurityalliance.org/about/>>. Acesso em: 25 jun. 2012

DANDAMUDI, S., "Performance implications of task routing and task scheduling strategies for multiprocessor systems," *Massively Parallel Computing Systems*, 1994., *Proceedings of the First International Conference on* , vol., no., pp.348,353, 2-6 May 1994 doi: 10.1109/MPCS.1994.367059

DONG, F.; AKL, S. G. **Scheduling Algorithms for Grid Computing: State of the Art and Open Problems**. Kingston, Ontario, Canada, Janeiro 2006.

EL-REWINI, Hesham; ABD-EL-BARR, Mostafa. **Advanced Computer Architecture and Parallel Processing**. [S.l.]: Wiley-Interscience, 2005. (Wiley Series on Parallel and Distributed Computing, ISBN 0-471-46740-5).

EUCALYPTUS. **WHAT IS EUCALYPTUS**. Disponível em: <<http://www.eucalyptus.com/learn/what-is-eucalyptus>>. Acesso em: 30 abr. 2012.

GOMES, Christian Lyra. Guia do Servidor Conectiva Linux 7.0. Conectiva S.A. cap 11. ISBN 85-87118-38-2. Disponível em <<http://www.dimap.ufrn.br/~aguiar/Manuais/Servidor/index.html>>. Acesso em: 30 abr. 2012.

HE, Zhonglin; HE, Yuhua. **Analysis on the security of cloud computing**. Proc. Spie, Qingdao, China, n., p.7752-775204, 2011.

JONES, M. Tim. **Anatomia de uma nuvem de software livre**. Disponível em: <<http://www.ibm.com/developerworks/br/opensource/library/os-cloud-anatomy/>>. Acesso em: 30 abr. 2012A.

JONES, M. Tim. **System emulation with QEMU**. Disponível em: <<http://www.ibm.com/developerworks/linux/library/l-qemu/l-qemu-pdf.pdf>>. Acesso em: 10 jul. 2012B.

KANSAL, N. J.; CHANA I. Cloud Load Balancing Techniques : A Step Towards Green Computing. IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 1, January 2012 ISSN (Online): 1694-0814

KIRKLAND, Dustin. **UEC/PackageInstall**. Disponível em: <<https://help.ubuntu.com/community/UEC/PackageInstall>>. Acesso em: 01 maio 2012.

KYI, H. M.; NAING, T. T. **An efficient approach for virtual machines scheduling on a private cloud environment**. *Broadband Network and Multimedia Technology* (IC-BNMT), 2011 4th IEEE International Conference on , vol., no., pp.365-369, 28-30 Oct. 2011. doi: 10.1109/ICBNMT.2011.6155958

LAUREANO, M. **Máquinas Virtuais e Emuladores – Conceitos, Técnicas e Aplicações**. São Paulo: Novatec, 2006.

LIN, C.; LIU, P., WU, J. **Energy-Aware Virtual Machine Dynamic Provision and Scheduling for Cloud Computing**. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing (CLOUD '11)*. IEEE Computer Society, Washington, DC, USA, 736-737. DOI=10.1109/CLOUD.2011.94 <http://dx.doi.org/10.1109/CLOUD.2011.94>

MICROSOFT. **O que é um driver?** Disponível em: <<http://windows.microsoft.com/pt-br/windows-vista/What-is-a-driver>>. Acesso em: 29 abr. 2012.

NEMETH, E.; SNYDER, G.; HEIN, T. R. **Manual completo do Linux: guia do**

administrador. 2 ed. São Paulo: Pearson Prentice Hall, 2007.

OLIVEIRA, R.; CARÍSSIMI, A.; TOSCANI, S. **Sistemas Operacionais**. 4. ed. Porto Alegre: Bookman, 2010.

OPENNEBULA. **Planning the Installation 3.0**. Disponível em: <<http://opennebula.org/documentation/archives:rel3.0:plan>>. Acesso em: 19 maio 2012A.

OPENNEBULA. **The Cloud Data Center Management Solution**. Disponível em: <<http://opennebula.org/start>>. Acesso em: 30 abr. 2012B.

OPENSTACK. **What is OpenStack?** Disponível em: <<http://docs.openstack.org/cactus/openstack-compute/admin/content/what-is-openstack.html>>. Acesso em: 30 abr. 2012.

POPOVIC, Kresimir; HOCENSKI, Zeljko. **Cloud computing security issues and challenges**. MIPRO, 2010 *Proceedings of the 33rd International Convention*. IEEE, Opatija, Croatia, 2010.

RED HAT. KVM – *KERNEL BASED VIRTUAL MACHINE*. Disponível em: <<http://www.redhat.com/rhecm/rest-rhecm/jcr/repository/collaboration/jcr:system/jcr:versionStorage/5e7884ed7f00000102c317385572f1b1/1/jcr:frozenNode/rh:pdfFile.pdf>>. Acesso em: 10 jul. 2012.

ROBERTS, John C.; AL-HAMDANI, Wasim. **Who can you trust in the cloud?: a review of security issues within cloud computing**. In *Proceedings of the 2011 Information Security Curriculum Development Conference (InfoSecCD '11)*. ACM, New York, NY, USA, 15-19, 2011.

RUSANOVA, O.; KOROCHKIN, A. **Scheduling problems for parallel and distributed systems**. In *Proceedings of the 1999 annual ACM SIGAda international conference on Ada (SIGAda '99)*. ACM, New York, NY, USA, 195-201. DOI=10.1145/319294.319323 <http://doi.acm.org/10.1145/319294.319323>

SABAHI, F. **Cloud computing security threats and responses**. *Communication Software and Networks (ICCSN)*, 2011 *IEEE 3rd International Conference on May 2011*.

SASIKALA, P. **Cloud computing: present status and future implications**. Disponível em: <<http://www.inderscience.com/storage/f123101246118579.pdf>>. Acesso em: 29 abr. 2012.

SIDHU, K. A.; KINGER, S. P. **Analysis of Load Balancing Techniques in Cloud Computing**. *International Journal of Computers & Technology* [0975-5462] Volume 4 No. 2, March-April, 2013, ISSN 2277-3061

SILBERSCHATZ, A.; GALVIN, P. B.; Galvin, e GAGNE, G. **Sistemas Operacionais com Java**. 7a. Ed. Rio de Janeiro: Elsevier/Campus, 2008.

SILVA, Peter. *Securing the Cloud*. Disponível em: <<http://forms.madisonlogic.com/FormConfirmation.aspx?pub=1&pgr=254&src=8758&cmp=5209&ast=18886&frm=1236&up=2-90111-33-6-57-123-0>>. Acesso em: 08 maio 2012.

SQLITE.ORG. **About SQLite**. Disponível em: <<http://www.sqlite.org/about.html>>. Acesso em: 12 jul. 2012.

SUMTER, La'Quata. **Cloud computing: security risk**. In *Proceedings of the 48th Annual Southeast Regional Conference (ACM SE '10)*. ACM, New York, NY, USA, , Article 112 , 4 pages. 2010.

TANENBAUM, A. S. *Sistemas Operacionais Modernos*. 2ª. ed. São Paulo: Prentice-Hall, 2003.

THANGARAJ, Ayyappan. **The Cloud – Hybrid Cloud – Part 6**. Disponível em: <<http://sqlserverrider.wordpress.com/2011/09/10/the-cloud-hybrid-cloud-part-6/>>. Acesso em: 29 abr. 2012.

TRENDS, Google. **Google Trends - Cloud Computing**. Disponível em: <<http://www.google.com.br/trends/?q=Cloud+Computing>>. Acesso em: 29 abr. 2012.

VCL, Apache. **Apache VCL**. Disponível em: <<https://cwiki.apache.org/VCL/>>. Acesso em: 30 abr. 2012.

VENUGOPAL, S.; BROBERG, J.; BUYYA, R. **OpenPEX: An open provisioning and EXecution system for virtual machines**. 17th *International Conference on Advanced Computing and Communications (ADCOM 2009)*, Bengaluru, India, 2009.

VMWARE. **Understanding Full Virtualization, Paravirtualization, and Hardware Assist**. Disponível em: <http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf>. Acesso em: 09 jul. 2012.

YANG, Z.; YIN, C.; LIU, Y. **A Cost-Based Resource Scheduling Paradigm in Cloud Computing**. In *Proceedings of the 2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT '11)*. IEEE Computer Society, Washington, DC, USA, 417-422. DOI=10.1109/PDCAT.2011.1 <http://dx.doi.org/10.1109/PDCAT.2011.1>

ZHENG, Z.; WANG, R.; ZHONG, H.; ZHANG, X. **An approach for cloud resource scheduling based on Parallel Genetic Algorithm**. *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, vol.2, no., pp.444-447, 11-13 March 2011. doi: 10.1109/ICCRD.2011.

GLOSSÁRIO

API (Application Programming Interface): É uma interface disponibilizada por uma aplicação para acesso externo por outras aplicações através de comandos preestabelecidos.

AWS (Amazon Web Services): Oferece um conjunto de serviços que juntos formam uma plataforma de computação confiável, escalável e de baixo custo na nuvem.

EC2: É um serviço *Web* da empresa *Amazon* que fornece uma capacidade de computação redimensionável na nuvem. É projetado para tornar a escalabilidade computacional de nível de *Web* mais fácil para desenvolvedores.

Escalonador (scheduler): É utilizado dentro de um gestor e tem a finalidade de selecionar um nó dentro da nuvem para receber a instância de máquina virtual.

Faiolver: O processo no qual uma máquina assume os serviços de outra, quando esta última apresenta falha.

Gestor: É um programa ou conjunto de programas que gerenciam o ambiente de nuvem.

Hypervisor: É uma camada de *software* entre o *hardware* e o sistema operacional, que controla o acesso dos sistemas operacionais convidados aos dispositivos de *hardware*.

IaaS (Infraestrutura como Serviço): É um modelo de disposição em que uma organização terceiriza o equipamento utilizado para apoiar as operações, incluindo armazenamento, *hardware*, servidores e componentes de rede. O prestador de serviços possui o equipamento e é responsável pela habilitação, execução e manutenção. O cliente geralmente paga por aquilo que utilizar.

KVM: É uma solução de virtualização completa para *Linux* na plataforma x86 contendo extensões de virtualização (*Intel VT* ou *AMD-V*). Usando *KVM*, é possível executar várias máquinas virtuais rodando *Linux* ou *Windows* sem precisar alterar essas imagens. Cada máquina virtual tem *hardware* virtualizado privado: uma placa de rede, disco, placa gráfica, entre outros.

Libvirt: É uma *API open-source* para gerenciamento de ferramentas de virtualização, podendo ser usado para gerenciar *Linux KVM*, *Xen*, *VMware ESX*, entre outros *hypervisors*.

Nuvem Pública: A infraestrutura é fornecida para muitos clientes e é gerenciada por terceiros. Várias empresas podem trabalhar simultaneamente na mesma infraestrutura. O usuário paga por aquilo que usar.

Nuvem Privada: Infraestrutura disponibilizada apenas para um cliente específico e gerenciada pela própria organização ou por terceiros. Este utiliza o conceito de virtualização de máquinas, usando rede proprietária.

Open-Source: É definido por aqueles programas em que é possível ter acesso ao código fonte, alterar e redistribuir.

OVF: É um padrão de empacotamento para distribuir máquinas virtuais. *OVF* permite a distribuição eficiente, flexível e segura de *software*, facilitando a mobilidade de máquinas virtuais, independente de plataforma.

PaaS: Neste modelo, o consumidor cria o *software*, utilizando ferramentas e bibliotecas do provedor de serviços. O consumidor também controla a distribuição do *software* e configurações. O provedor fornece as redes, servidores e armazenamento.

Paravirtualização: É uma alternativa à virtualização total. Nesse modelo de virtualização, o sistema operacional é modificado para chamar o *VMM* sempre que executar uma instrução que possa alterar o estado do sistema, uma instrução sensível. Isso acaba com a necessidade de o *VMM* testar instrução por instrução, o

que representa um ganho significativo de desempenho.

QEMU: É um emulador de máquina genérica e *open-source*. Pode executar sistemas operacionais e programas feitos para uma máquina (por exemplo, uma placa *ARM*) em uma máquina diferente (por exemplo, seu próprio *PC*).

SaaS: É um modelo de distribuição de *software* em que os aplicativos são hospedados por um fornecedor de serviços e disponibilizados aos clientes, através de uma rede, geralmente a Internet.

S3 (Amazon S3): É o armazenamento para a Internet. Fornece uma interface simples de serviço *Web* que pode ser usada para armazenar e recuperar qualquer quantidade de dados, a qualquer momento, de qualquer lugar na *Web*.

VIM: Esse tipo de *software* se assemelha a um sistema operacional, porém este agrega recursos de vários computadores apresentando uma visão uniforme e aplicações para usuário.

Virtualização Total: Tem por objetivo fornecer ao sistema operacional visitante uma réplica do *hardware* subjacente.

VMM: É um monitor de máquina virtual que é responsável por criar máquinas virtuais. Cada *VMM* em execução no *hypervisor* implementa a abstração de *hardware* da máquina virtual e é responsável por executar um sistema operacional convidado. Cada *VMM* compartilhar a *CPU*, memória e dispositivos de entrada/saída para virtualizar o sistema com sucesso.

VMWARE: É um *software* que visa criar ambientes para instalação de sistemas distintos. Ele permite a instalação e utilização de um sistema operacional dentro de outro dando suporte real a *softwares* de outros sistemas.

XEN: É um *hypervisor*, *open-source* para virtualização, oferece suporte para as arquiteturas *x86_64*, *x86*, *IA64*, *ARM* e outras arquiteturas de *CPU*. Ele suporta uma ampla gama de sistemas operacionais convidados incluindo *Windows*, *Linux* e *Solaris*.

APÊNDICE A - INSTALAÇÃO DO *EUCALYPTUS*

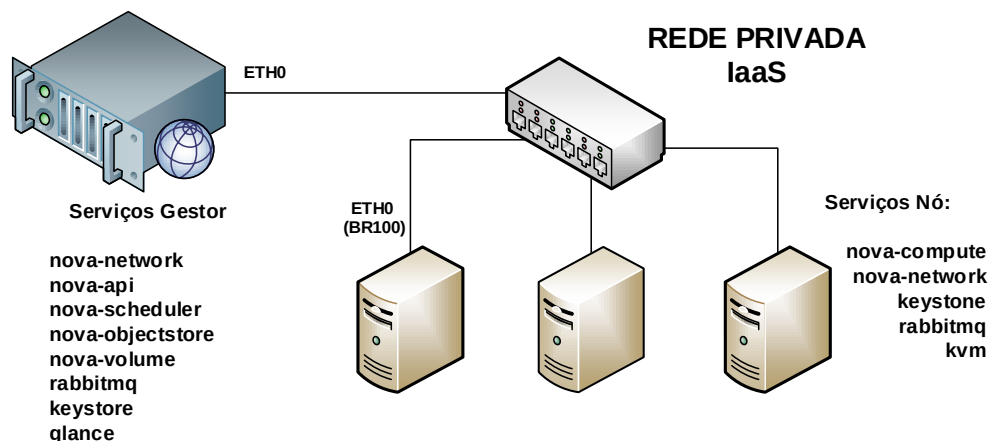
Para analisar os gestores, foram utilizadas quatro máquinas com a mesma configuração de *hardware*, conforme Tabela 4. Todas as máquinas possuem o mesmo sistema operacional, *Ubuntu Linux Server*. Uma máquina comporta-se como o gestor da nuvem e as demais como nós computacionais.

TABELA 4 – ESPECIFICAÇÃO DE *HARDWARE* PARA A NUVEM.

Hardware	Especificação
CPU	Intel Core 2 Quad 2.66GHz
Memória	4 GB
Disco	SATA 7200 rpm
Espaço em disco	500 GB
Rede	100 Mbps

As máquinas estão na mesma rede e não há comunicação com nuvens públicas, caracterizando-se uma arquitetura privada.

Figura 27 – *Layout* de instalação do *OpenStack Essex*.

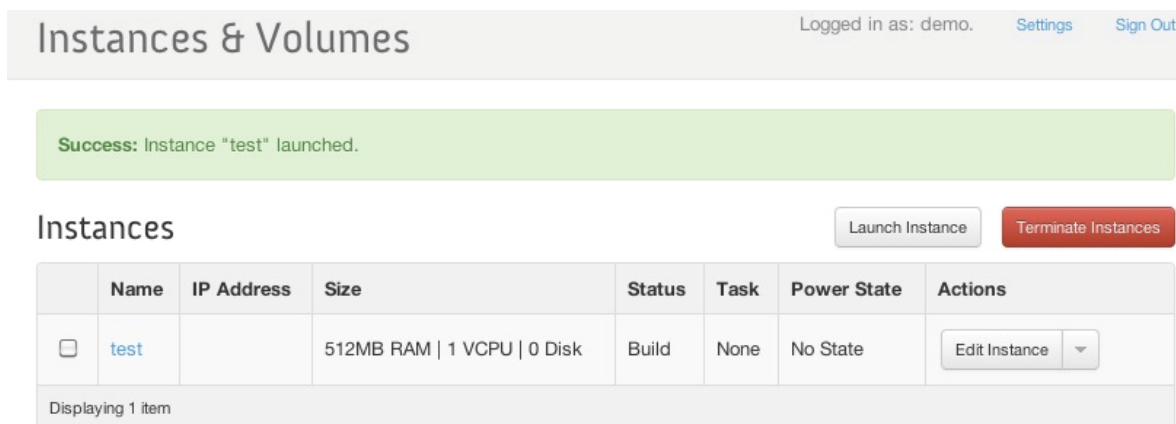


Fonte: OPENSTACK, 2012.

O *OpenStack Essex* tem diversas opções de instalação, pode ser instalado em um ou mais servidores com ou sem suporte de virtualização (quando não há suporte de virtualização, é feito emulação com *QEMU*). Dos quatro computadores utilizados, um executou todos os serviços do gestor como

nova-compute, *nova-api*, *nova-volume*, *nova-network*; e os demais computadores executaram o *hypervisor*, responsável pelo processamento das máquinas virtuais, conforme pode ser observado na Figura 27.

Figura 28 – Interface *Web* de gerenciamento – *Horizon*.



Fonte: OPENSTACK, 2012.

As instâncias e recursos são visualizados por uma interface *Web* de gerenciamento chamada *Horizon*, conforme Figura 28. A interface permite administrar volumes, criar instâncias, definir parâmetros de acesso e segurança, configurar a rede e acesso direto a instância de máquina virtual.

A instalação do *Eucalyptus* divide-se basicamente em duas partes, o gestor e os nós. Após instalação do sistema operacional (*Ubuntu 10.10 64bits Eucalyptus Cloud*) deve-se atualizar o sistema nas duas máquinas:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

Instalação Gestor

```
sudo apt-get install eucalyptus-cloud eucalyptus-cc eucalyptus-walrus eucalyptus-sc
```

Nome do *cluster*: *cluster1*

Range de *IPs* disponíveis para rede pública: 192.168.1.200-192.168.1.249

Instalação Nó

```
sudo apt-get install eucalyptus-nc
```

A comunicação entre o gestor e os nós é feita pela interface de rede “br0”:

```
auto br0
iface br0 inet static
    address 192.168.70.1
    network 192.168.70.0
    netmask 255.255.255.0
    broadcast 192.168.70.255
    bridge_ports eth0
    bridge_fd 9
    bridge_hello 2
    bridge_maxage 12
    bridge_stp off
```

Após, configurar o eucalyptus.conf com a interface de rede “br0”:

```
sudo /etc/init.d/eucalyptus-nc restart
```

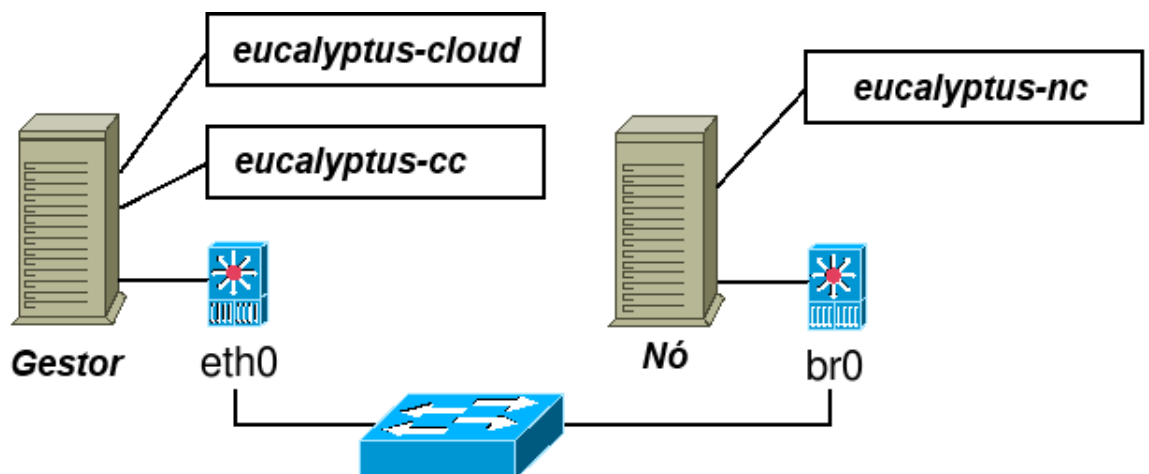
Definir uma senha para o usuário “eucalyptus” no gestor:

```
sudo passwd eucalyptus
```

No gestor, é necessário autorizar o usuário “eucalyptus” a acessar os nós diretamente:

```
sudo -u eucalyptus ssh-copy-id -i ~eucalyptus/.ssh/id_rsa.pub
eucalyptus@<IP_OF_NODE>
```

Figura 29 – Ambiente de Instalação – Eucalyptus.



Fonte: KIRKLAND, 2012.

Conforme Figura 29, o gestor e os nós estarão ligados por uma rede física e cada um receberá os pacotes correspondentes aos seus serviços.

Em seguida, no gestor, é necessário criar as credenciais de usuário para utilizar as instâncias de VM:

```
unzip -d ~/.euca mycreds.zip
mkdir -p ~/.euca
chmod 700 ~/.euca
cd ~/.euca
sudo euca_conf --get-credentials mycreds.zip
unzip mycreds.zip
ln -s ~/.euca/eucarc ~/.eucarc
cd -
```

Para validar o *cluster* e verificar se a agregação de instâncias está funcionando:

```
. ~/.euca/eucarc
euca-describe-availability-zones verbose
```

No gestor, é necessário realizar as operações abaixo para carregar em uma instância a imagem do euca-ubuntu-9.04-x86_64.tar.gz:

```
tar zxvf euca-ubuntu-9.04-x86_64.tar.gz
euca-bundle-image -i euca-ubuntu-9.04-x86_64/kvm-kernel/vmlinuz-2.6.28-11-generic --kernel true
euca-upload-bundle -b ubuntu-kernel-bucket -m /tmp/vmlinuz-2.6.28-11-generic.manifest.xml
euca-register ubuntu-kernel-bucket/vmlinuz-2.6.28-11-generic.manifest.xml

euca-bundle-image -i euca-ubuntu-9.04-x86_64/kvm-kernel/initrd.img-2.6.28-11-generic --ramdisk true
euca-upload-bundle -b ubuntu-ramdisk-bucket -m /tmp/initrd.img-2.6.28-11-generic.manifest.xml
euca-register ubuntu-ramdisk-bucket/initrd.img-2.6.28-11-generic.manifest.xml

euca-bundle-image -i euca-ubuntu-9.04-x86_64/ubuntu.9-04.x86-64.img --kernel $EKI --ramdisk $ERI
euca-upload-bundle -b ubuntu-image-bucket -m /tmp/ubuntu.9-04.x86-64.img.manifest.xml
euca-register ubuntu-image-bucket/ubuntu.9-04.x86-64.img.manifest.xml
```

Comando para executar a instância:

```
euca-run-instances -k mykey --kernel <eki-XXXXXXX> --ramdisk <eri-XXXXXXX> <emi-XXXXXXX>
```

Para acompanhar a inicialização da instância:

```
watch -n5 euca-describe-instances
```

APÊNDICE B - INSTALAÇÃO DO *OPENNEBULA*

A instalação do *OpenNebula* não difere da instalação de outros gestores de nuvem, divide-se basicamente em duas partes, o gestor e os nós. Após instalação do sistema operacional (*Ubuntu 10.10 64bits Server*) deve-se atualizar o sistema nas duas máquinas:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

Instalação Gestor

```
sudo apt-get install opennebula
```

Nome do *cluster*: opennebula

Instalação Nó

```
sudo apt-get install opennebula-node
```

A comunicação entre o gestor e os nós é feita pela interface de rede “br0”:

```
auto br0
iface br0 inet static
    address 192.168.70.1
    network 192.168.70.0
    netmask 255.255.255.0
    broadcast 192.168.70.255
    bridge_ports eth0
    bridge_fd 9
    bridge_hello 2
    bridge_maxage 12
    bridge_stp off
```

Após, é necessário criar o sistema de chave pública para permitir comunicação direta entre nós e gestor:

```
sudo scp /var/lib/one/.ssh/id_rsa.pub
```

```

oneadmin@node01:/var/lib/one/.ssh/authorized_keys
sudo scp /var/lib/one/.ssh/id_rsa.pub
oneadmin@node02:/var/lib/one/.ssh/authorized_keys
sudo sh -c "cat /var/lib/one/.ssh/id_rsa.pub >> /var/lib/one/.ssh/authorized_keys"

```

É necessário criar o diretório que irá alocar as imagens de instâncias de VMs:

```

sudo mkdir /var/lib/one/images
sudo chown oneadmin /var/lib/one/images/

```

Em seguida, é necessário registrar os nós participantes do *cluster*:

```

onehost create node01 im_kvm vmm_kvm tm_ssh
onehost create node02 im_kvm vmm_kvm tm_ssh

```

Para que a instância de VM possa comunicar na rede, é necessário criar uma configuração de rede virtual para o *cluster* (*vnet01.template*):

```

NAME = "LAN"
TYPE = RANGED
BRIDGE = br0
NETWORK_SIZE = C
NETWORK_ADDRESS = 192.168.0.0

```

É necessário carregar o *template* de rede criado:

```

onevnet create vnet01.template

```

Após, é criado o *template* da instância de VM, informando todos os parâmetros necessários para a inicialização da máquina virtual (*vm01.template*):

```

NAME = vm01
CPU = 0.5
MEMORY = 512
OS = [ BOOT = hd ]
DISK = [
    source = "/var/lib/one/images/vm01.qcow2",
    target = "hda",

```

```
readonly = "no" ]  
NIC = [ NETWORK="LAN" ]  
GRAPHICS = [type="vnc",listen="127.0.0.1",port="-1"]
```

Finalmente, deve-se criar a instância de VM:

```
onevm submit vm01.template
```

APÊNDICE C - INSTALAÇÃO DO OPENSTACK ESSEX

Na instalação do *OpenStack Essex* serão utilizados quatro computadores, cada um com uma função específica e todos com o sistema operacional (*Ubuntu 12.04 64bits Server*) devidamente atualizados:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install -y unzip
sudo apt-get install -y bridge-utils
brctl addbr br100
apt-get install iputils-arping -y
apt-get install arping -y
apt-get install git -y
```

Servidor Gestor

O gestor contém todos os serviços da nuvem, *nova-compute*, *nova-api*, *nova-volume*, *nova-network*, *Glance Image Service* e *Swift*. Configuração da rede, considerando:

- eth0: 192.168.100.0/24 (comunicação gestor/nó)
- eth1: 10.10.100.0/24 (comunicação externa, acesso *SSH* e *VNC*)

```
auto eth0
iface eth0 inet static
    address 192.168.100.1
    netmask 255.255.255.0

auto eth1
iface eth1 inet static
    address 10.10.100.100.1
    netmask 255.255.255.0
```

A instalação do servidor de *NTP* é importante para manter a sincronização de relógios entres os computadores participantes da nuvem.

```
#apt-get install -y ntp
#sed -i 's/server ntp.ubuntu.com/server ntp.ubuntu.com\nserver
127.127.1.0\nfudge 127.127.1.0 stratum 10/g' /etc/ntp.conf
```

Após, é necessário adicionar o usuário de comunicação gestor/nó:

```
#useradd -U -G sudo -s /bin/bash -m stack
#echo "stack ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
#mkdir ~/.ssh; chmod 700 ~/.ssh
#echo "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQACyYjfgYPazTvGpd8OaAvtU2utL8W6g
WC4JdRS1J95GhNNfQd657yO6s1AH5KYQWktcE6FO/xNUC2reEXSGC7ezy+sG
O1kj9Limv5vrvNHvF1+wts0Cmyx61D2nQw35/Qz8BvpdJANL7VwP/cFI/p3yhvx2ls
njFE3hN8xRB2LtLUopUSVdBwACOVUmH2G+2BWMJDjVINd2DPqRIA4Zhy09KJ
3O1Joabr0XpQL0yt/I9x8BVHdAx6l9U0tMg9dj5+tAjZvMAFfye3PJcYwwsfJoFxC8w/
SLtqlFX7Ehw++8RtvomvuipLdmWCy+T9hIkl+gHYE4cS3OIqXH7f49jdJf
jesse@spacey.local" > ~/.ssh/authorized_keys
```

Através do *GIT*, é realizado o clone do script *DevStack*, responsável pela instalação automática dos pacotes necessários para a nuvem:

```
# mkdir /opt/stack/
#cd ~
#git clone git://github.com/openstack-dev/devstack.git
#cd devstack
```

Na sequência, adicionar o gestor no */etc/hosts*:

```
192.168.100.1      controller
```

Configuração do *localrc* (em *~/devstack/*):

```
HOST_IP=192.168.100.1
FLAT_INTERFACE=eth0
FLAT_NETWORK_BRIDGE=br100
FIXED_RANGE=192.168.100.0/24
FLAT_NETWORK_DHCP_START==192.168.100.10
MULTI_HOST=1
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=cloudunesp
MYSQL_PASSWORD=cloudunesp
RABBIT_PASSWORD=cloudunesp
SERVICE_PASSWORD=cloudunesp
SERVICE_TOKEN=cloudunesp
ENABLED_SERVICES=g-api,g-reg,key,n-api,n-crt,n-obj,n-net,n-cond,cinder,c-
sch,c-api,c-vol,n-sch,n-novnc,n-xvnc,n-cauth,horizon,rabbit,tempest,mysql
```

Executar o *script* *stack.sh* (em *~/devstack/*):

```
#./stack.sh
```

Nós

Para os nós, é necessário repetir todos os passos do gestor, alterando o *IP* correspondente na interface de rede e o arquivo *localrc* (em *~/devstack/*):

```
HOST_IP=192.168.100.2
FLAT_INTERFACE=eth0
FLAT_NETWORK_BRIDGE=br100
FIXED_RANGE=192.168.100.0/24
FLAT_NETWORK_DHCP_START==192.168.100.10
MULTI_HOST=1
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=cloudunesp
MYSQL_PASSWORD=cloudunesp
RABBIT_PASSWORD=cloudunesp
SERVICE_PASSWORD=cloudunesp
SERVICE_TOKEN=cloudunesp
MYSQL_HOST=192.168.100.1
RABBIT_HOST=192.168.100.1
GLANCE_HOSTPORT=192.168.100.1:9292
ENABLED_SERVICES=n-cpu,n-net
```

Executar o *script* *stack.sh* (em *~/devstack/*):

```
#!/stack.sh
```

Reservar os dez primeiros *IPs* para os nós:

```
#for i in `seq 2 10`; do /opt/stack/nova/bin/nova-manage fixed reserve
192.168.100.$i; done
```

Para carregar os serviços no nó:

```
#!/opt/stack/nova/bin/nova-compute start &
#!/opt/stack/nova/bin/nova-network start &
#nova-manage db sync
```

Acessar o gestor com *root*, executar o comando abaixo para checar a comunicação entre gestor e nó:

```
#nova-manage service list
```

O comando acima deve retornar:

Binary	Host	Zone	Status	State	Updated_At
nova-cert	controller	nova	enabled	:-)	26/11/2012
nova-compute		controller	nova	enabled	:-)
26/11/2012					
nova-scheduler		controller	nova	enabled	:-)
26/11/2012					

nova-network	controller nova	enabled :-)	26/11/2012
nova-consoleauth	controller nova	enabled :-)	26/11/2012
nova-compute	node nova	enabled :-)	26/11/2012
nova-network	node nova	enabled :-)	26/11/2012

Como pode ser observado acima, o nó apareceu em host com status “:-)”. Para finalizar, acessar a interface gráfica *Horizon* e carregar as instâncias de máquinas virtuais.

ANEXO A – TEST_NODE_FX.PL

```
#!/usr/bin/perl
use warnings;
use threads;
use threads::shared;
use IO::Socket;
use Data::Dumper;

my $rand = shift;
$rand = 0 if (! $rand);
$rand = lc($rand);

$lifetime = 1;

if ($rand) {
    my $threads;
    sub subGT {
        $threads = threads->new(\&stressDisk) if ($rand == 1);
        $threads = threads->new(\&stressNet) if ($rand == 2);
        $threads = threads->new(\&stressCPU) if ($rand == 3);
    }

    sub stressDisk {
        while ( $lifetime > 0 ) {
            `dd if=/dev/urandom of="/tmp/testfile" count=60 bs=1024k`;
            sleep 1;
        }
        return 0;
    }

    sub stressNet {
        while ( $lifetime > 0 ) {
            `wget --limit-rate=100k http://www.cloudcomputingforum.com.br/testfile -O /tmp/testfile.net`;
        }
        return 0;
    }

    sub stressCPU {
        while ( $lifetime > 0 ) {
            `md5sum /tmp/testfile`;
            sleep 0.01;
        }
        return 0;
    }
}

# MAIN PROGRAM
#####
while (1) {
    $time = 60;
    `echo "Sleeping $time ..." >> /tmp/log`;
    &subGT($time);
    sleep $time;
    $threads->kill('STOP')->detach();
}
#####
}

#Lancamentos
#####
sub subLanc {
    my ($lanc) = @_ ;
    do `usr/bin/perl /home/cloud/stress_node_fx.pl $lanc`;
    return 0;
}

$lanc = 3;
while (1) {
    print "$lanc lancamentos \n";
    $threads = threads->new(\&subLanc, $lanc);
    $lanc--;
}
```

```
    if (!$lanc) {  
        sleep 60;  
        $lanc = 3  
    }  
}  
exit(1);
```

ANEXO B – TEST_NODE_RAND.PL

```
#!/usr/bin/perl
use warnings;
use threads;
use threads::shared;
use IO::Socket;
use Data::Dumper;

my $rand = shift;
$rand = 0 if (! $rand);
$rand = lc($rand);

if ($rand) {
    my $threads;
    `echo "Thread: $rand" >> /tmp/log`;
    sub subGT {
        my ($lifetime) = @_ ;
        $sleep = int(rand(60)) + 1;

        $choice = int(rand(6)) + 1;
        `echo "Choice: $choice" >> /tmp/log`;
        if ($choice == 1 or $choice == 4 or $choice == 6) {
            $threads = threads->new(\&stressDisk, $sleep, $lifetime);
        }
        if ($choice == 2 or $choice == 5 or $choice == 6) {
            $threads = threads->new(\&stressNet, $sleep, $lifetime);
        }
        if ($choice == 3 or $choice == 4 or $choice == 5 or $choice == 6) {
            $threads = threads->new(\&stressCPU, $sleep, $lifetime);
        }
    }

    sub stressDisk {
        my ($sleep, $lifetime) = @_ ;
        while ( $lifetime > 0 ) {
            `dd if=/dev/urandom of="/tmp/testfile" count=$sleep bs=1024k`;
            `echo "stressDisk|Sleep: $sleep |Tempo Vida: $lifetime" >> /tmp/log`;
            sleep 2;
            $lifetime--;
        }
        return 0;
    }

    sub stressNet {
        my ($sleep, $lifetime) = @_ ;
        while ( $lifetime > 0 ) {
            `echo "stressNet|Sleep: $sleep |Tempo Vida: $lifetime" >> /tmp/log`;
            `wget --limit-rate=100k http://www.cloudcomputingforum.com.br/testfile -O /tmp/testfile.net`;
            sleep $sleep;
            $lifetime--;
        }
        return 0;
    }

    sub stressCPU {
        my ($sleep, $lifetime) = @_ ;
        while ( $lifetime > 0 ) {
            `echo "stressCPU|Sleep: $sleep |Tempo Vida: $lifetime" >> /tmp/log`;
            `md5sum /tmp/testfile`;
            sleep $sleep;
            $lifetime--;
            $lifetime=0 if ( ! -e "/tmp/testfile" );
        }
        return 0;
    }
}

# MAIN PROGRAM
#####
while (1) {
    $time = ( int(rand(5)) + 1 ) * 60;
```

```

        `echo "Sleeping $time ..." >> /tmp/log`;
        &subGT($time);
        sleep $time;
        $threads->kill('STOP')->detach();
    }
    #####
}

#Lancamentos
#####
sub subLanc {
    my ($lanc) = @_ ;
    do `usr/bin/perl /home/cloud/stress_node.pl $lanc`;
    return 0;
}

$lanc = int(rand(9) + 1) if (!$lanc);
while (1) {
    print "$lanc lancamentos \n";
    $threads = threads->new(\&subLanc, $lanc);
    $lanc--;
    if (!$lanc) {
        sleep 60;
        $lanc = int(rand(9) + 1);
    }
}
exit(1);

```

ANEXO C – NODE_RESOURCES.PY

```
#!/usr/bin/env python

import MySQLdb
import os
import subprocess as sp
import sys
import platform
import time
import curses
import atexit
import commands
from datetime import datetime, timedelta
import psutil
if os.name != 'posix':
    sys.exit('platform not supported')

# create connection with database
con = MySQLdb.connect('192.168.100.200', 'unesp', 'unespcloud')
con.select_db('unesp')
cursor = con.cursor()

def resources(interval):
    """Check Resources as network and disk"""
    time.sleep(interval)

    # coletando dados transferidos da rede e disco
    network_before = psutil.network_io_counters()
    disks_before = psutil.disk_io_counters()

    time.sleep(interval)

    # coletando dados transferidos da rede e disco
    network_after = psutil.network_io_counters()
    disks_after = psutil.disk_io_counters()

    # verificando a taxa de transf. por segundo do disco
    disks_read_per_sec = disks_after.read_bytes - disks_before.read_bytes
    disks_write_per_sec = disks_after.write_bytes - disks_before.write_bytes

    # verificando a taxa de transferência por segundo da rede
    net_recv_per_sec = network_after.bytes_recv - network_before.bytes_recv
    net_sent_per_sec = network_after.bytes_sent - network_before.bytes_sent

    # retornando valores
    return (disks_read_per_sec, disks_write_per_sec, net_recv_per_sec, net_sent_per_sec)

def check_node(disks_read, disks_write, net_recv, net_sent):
    """Check status of node."""
    # cpu usage
    cpu_avail = 0
    for cpu,perc in enumerate(psutil.cpu_percent(interval=0, percpu=True)):
        cpu_avail += 100 - perc
    cpu_avail = (100 * cpu_avail) / (psutil.NUM_CPUS * 100)

    # me usage
    mem = psutil.virtual_memory()
    mem_avail = 100 - mem.percent

    # load usage
    av1, av2, av3 = os.getloadavg()
    av1 = int(av1)

    # disk usage
    try:
        f = open("/tmp/cc_resources", "r")
        vel_disk = int(f.read())
        if (vel_disk < 80 and vel_disk > 180):
            disk_cap = 140
    except:
```

```

else:
    disk_cap = vel_disk
    disk_avail = (( disks_read + disks_write) * 100 ) / (disk_cap * 1024 * 1024)
    disk_avail = int ( (100 - disk_avail) )
except():
    disk_avail = 0

# net usage
f = open("/tmp/cc_resources", "r")
vel_net = int(check_net_velocity())
if (vel_net < 100 and vel_net > 1000):
    net_cap = 100
else:
    net_cap = vel_net

net_avail = int((( net_rcv + net_sent) * 100 ) / (((net_cap) / 8)*1024)*1024)
if (net_avail > 100):
    net_avail = 100
net_avail = int ( (100 - net_avail) )

# put values in database
SQL_QUERY = "SELECT SUM(LA),SUM(CPU),SUM(MEM),SUM(NET),SUM(DISK),MIN(MIN_LA),MIN(MIN_CPU),MIN(MIN_MEM),MIN(MIN_NET),MIN(MIN_DISK),COUNT(*) FROM node WHERE node = '%s' AND TIMEDIFF(NOW(), record) > '00:59:59'" % platform.uname()[1]
cursor.execute(SQL_QUERY)
result_set = cursor.fetchall()
count = 0
if ( cursor.rowcount > 0 ):
    for row in result_set:

        if (row[10] <= 0):break
        else:count=row[10]

        if (av1 > int(row[5])):
            LA_MIN = av1
        else:
            LA_MIN = int(row[5])

        if (cpu_avail < int(row[6])):
            CPU_MIN = cpu_avail
        else:
            CPU_MIN = int(row[6])

        if (mem_avail < int(row[7])):
            MEM_MIN = mem_avail
        else:
            MEM_MIN = int(row[7])

        if (net_avail < int(row[8])):
            NET_MIN = net_avail
        else:
            NET_MIN = int(row[8])
        if (disk_avail < int(row[9])):
            DISK_MIN = disk_avail
        else:
            DISK_MIN = int(row[9])

if ( count > 0):
    LA_MED = int(row[0] / count)
    CPU_MED = int(row[1] / count)
    MEM_MED = int(row[2] / count)
    NET_MED = int(row[3] / count)
    DISK_MED = int(row[4] / count)
    SQL = "INSERT INTO node VALUES (NULL, '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', NOW() )" % ( platform.uname()[1], av1, cpu_avail, mem_avail, net_avail, disk_avail, LA_MED, CPU_MED, MEM_MED, NET_MED, DISK_MED, LA_MIN, CPU_MIN, MEM_MIN, NET_MIN, DISK_MIN )
else:
    # first SQL
    SQL = "INSERT INTO node VALUES (NULL, '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', NOW() )" % ( platform.uname()[1], av1, cpu_avail, mem_avail, net_avail, disk_avail, av1, cpu_avail, mem_avail, net_avail, disk_avail )

cursor.execute(SQL)
con.commit()
```

```

def check_net_velocity():
    """ Check the bridge of the cloud and your velocity """
    bridge_cloud = sp.check_output("brctl show | egrep -v interfaces | awk '{ print $4 }' | egrep -v \"^[[:space:]]*$\"",
    shell=True)
    bridge = "cat /sys/class/net/%s/speed" % bridge_cloud.rstrip("\n\t")
    veloc = sp.check_output(bridge, shell=True)
    return veloc.rstrip("\n\t")

def check_disk_velocity():
    """ Check the velocity's disk """
    t1 = time.time()
    exec_dd = sp.check_output("dd if=/dev/zero of=/tmp/output.img bs=8k count=256k", shell=True)
    t2 = time.time()
    veloc = int(2150.4 / ( (t2-t1) - 1))
    f = open("/tmp/cc_resources", "w")
    f.write("%s" % veloc)
    f.close()

def main():
    try:
        if ( not os.path.isfile("/tmp/cc_resources") ):
            check_disk_velocity()

        interval = 1

        args = resources(interval)
        check_node(*args)

        # close connection with database
        con.close()
    except (KeyboardInterrupt, SystemExit):
        pass

if __name__ == '__main__':
    main()

```

ANEXO D – CHOICE_NODE.PY

```
#!/usr/bin/env python
import MySQLdb
import os
import subprocess as sp
import sys
import platform
import time
import curses
import atexit
import commands
from datetime import datetime, timedelta
if os.name != 'posix':
    sys.exit('platform not supported')

# create connection with database
con = MySQLdb.connect('192.168.100.200', 'root', 'cloudunesp')
con.select_db('unesp')
cursor = con.cursor()

def choice_node():
    SQL_QUERY = "SELECT node,AVG(CPU),AVG(MEM),AVG(NET),AVG(DISK),record from node GROUP BY node
ORDER BY record DESC"
    cursor.execute(SQL_QUERY)
    result_set = cursor.fetchall()
    node_ant = 0;
    best_node = "";
    if ( cursor.rowcount > 0 ):
        for row in result_set:
            node = (row[1] * 0.6 + row[2] * 0.2 + row[3] * 0.1 + row[4] * 0.1)
            if (node > node_ant):
                node_ant = node
                best_node = row[0]
        return best_node
    return 0

def main():
    try:
        print choice_node()

        # close connection with database
        con.close()
    except (KeyboardInterrupt, SystemExit):
        pass

if __name__ == '__main__':
    main()
```

Autorizo a reprodução xerográfica para fins de pesquisa.

São José do Rio Preto, ____/____/____

Assinatura