Universidade Federal do Pampa
Jonathan Patrick Rosso

Análise de Desempenho de Aplicações Científicas em Ambiente de Nuvem Privada

Alegrete
2015

Jonathan Patrick Rosso

Análise de Desempenho de Aplicações Científicas em Ambiente de Nuvem Privada

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Claudio Schepke

Alegrete

2015

Jonathan Patrick Rosso

Análise de Desempenho de Aplicações Científicas em Ambiente de Nuvem Privada

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 01 de Dezembro de 2015

Banca examinadora:

Claudio Schepke
Orientador

Márcia Cristina Cera
Universidade Federal do Pampa

Dalvan Jair Griebler
PUCRS

Resumo

Infraestruturas de computação em nuvem vem sendo uma alternativa para execução de aplicações científicas, pois possuem uma alta flexibilidade na alocação de recursos a um custo relativamente baixo. Para sua implementação são disponíveis uma série de ferramentas *open source* capazes de gerenciar esses ambientes. No entanto, ambientes virtuais podem sofrer uma perda de desempenho devido ao *overhead* causado pelo *software* de virtualização. Neste contexto, este trabalho teve por objetivo avaliar o desempenho de duas aplicações científicas reais (*Ocean-Land-Atmosphere Model* (OLAM) e Método de Lattice Boltzmann (MLB)), escolhidas pela sua relevância no contexto científico e por apresentarem um elevado custo computacional. A avaliação foi feita considerando duas nuvens privadas, uma gerenciada pela ferramenta OpenStack e outra pela ferramenta OpenNebula. Os resultados mostraram maiores quedas de desempenho da nuvem privada em relação ao ambiente nativo, nas execuções que necessitaram de uma maior comunicação entre processos. Os resultados também mostraram que as ferramentas podem impactar em uma perda de desempenho.

Palavras-chave: Computação em Nuvem. IaaS. Nuvem Privada, Computação Científica.

Abstract

Cloud computing infrastructures are becoming a good alternative to execute scientific applications. The reason of this is that they have a high resources allocation flexibility for a cheap price. For its implementation, there are available many open source tools for the management of these environments. However, virtual environments can suffer of a loss of performance caused by the overhead of the virtualization software. This problem becomes worse when some classes of application have a high communication demand among the processes. In this context, this project has as its objective the evaluation of the performance of private clouds infrastructures. In order to do this analysis, two scientific applications were utilized. These applications have a high computational demand, and low communication among the processes. The evaluation was made considering two private clouds, which were managed by OpenNebula and OpenStack. The results showed a bigger loss on the executions that demanded more communication among the processes. Therefore, it lead to the conclusion that the overload of network virtualization has a bigger impact. The results also showed that the tools have an impact on the performance, where OpenNebula had slight better results.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API *Application Programming Interfaces*

CPU *Central Processing Unit*

DFC Dinâmica dos Fluídos Computacionais

EC2 *Elastic Computing Cloud*

GPL *Generical Public License*

HPC *High Performance Computing*

IaaS *Infrastructure as a Service*

LARCC Laboratório de Pesquisas Avançadas para Computação em Nuvem da SETREM

MLB Método de Lattice Boltzmann

MPI *Message Passing Interface*

NFS *Network File System*

NIST *National Institute of Standards and Technology*

OLAM *Ocean-Land-Atmosphere Model*

OpenMP *Open Multi-processing*

PaaS *Platform as a Service*

PC *Personal Computer*

SaaS *Software as a Service*

SLA *Service Level Agreement*

SO Sistema Operacional

SPMD *Single Program Multiple Data*

SSH *Secure Shell*

TI Tecnologia da Informação

VMM *Virtual Machine Monitor*

VMs *Virtual Machines*

Sumário

1 Introdução

A computação em Nuvem é hoje um dos principais paradigmas da computação, figurando cada vez mais nas infraestruturas das organizações atuais. Seu modelo propõe a integração de diversos modelos tecnológicos, com o objetivo de proporcionar serviços de Tecnologia da Informação (TI) sob demanda, utilizando hardware compartilhado para processamento e armazenamento (MELL; GRANCE, 2011). Ao utilizar um ambiente de nuvem, organizações tem a possibilidade de adquirir recursos de forma dinâmica e elástica. Essa flexibilidade é possível graças ao uso da virtualização que prove um ambiente de processamento independente e isolado através da criação de *Virtual Machines* (VMs) que podem ser facilmente instanciadas e destruídas.

O paradigma da Computação em Nuvem pode ser elaborado com diferentes níveis da pilha de TI. Segundo NIST, uma nuvem pode distribuir seus recursos em três modelos de serviço (*Software as a Service* (SaaS), *Platform as a Service* (PaaS), *Infrastructure as a Service* (IaaS)) e quatro formas de implementação (público, privado, comunitário e híbrido). Este trabalho aborda especificamente o modelo IaaS. Este, por sua vez, é o nível mais baixo da pilha, onde o usuário tem a sua disposição toda uma infraestrutura virtual com controle total sobre as máquinas virtuais, aplicativos instalados e armazenamento. Para implementação de IaaS são disponíveis ferramentas *open source* de administração de nuvem, capazes de gerenciar os ambientes virtualizados de forma eficiente, facilitando a manutenção e o gerenciamento dos recursos (BUYYA et al., 2009).

A computação em Nuvem criou uma série de vantagens se comparado a infraestruturas tradicionais de computação. Elasticidade e redução nos custos vem atraindo cada vez mais o interesse da comunidade científica. Para muitos pesquisadores, a computação em nuvem pode ser uma alternativa viável na execução de aplicações científicas, beneficiando desde usuários que possuem pequenas aplicações, até aqueles que executam seus experimentos em grandes centros de supercomputação.

Apesar de todas essas vantagens, IaaS ainda possuem uma série de desafios que precisam ser considerados na execução de aplicações científicas. Uma importante questão é que, como os ambientes são virtualizados, existe o problema do *overhead* causado pela virtualização (SUBRAMANIAN et al., 2011). Esse problema aumenta quando determinadas classes de aplicações necessitam de grande comunicação entre os processos, fazendo com que a perda de desempenho seja significativa. Outro fator que pode impactar no desempenho, são as ferramentas de administração de nuvem. Segundo Maron et al. (2015) ferramentas de computação em nuvem possuem componentes particulares e características próprias de implementação, além de oferecer diferentes funcionalidades para tratar a

camada de virtualização.

1.1 Motivação e Objetivos

Muitos centros de pesquisa possuem um grande número de recursos, na maioria das vezes subutilizado. Segundo Taurion (2009), infraestruturas de TI tradicionais apresentam níveis de utilização muito baixos, chegando a usar apenas de 5% a 10% da capacidade de processamento total disponível. Com isso, a computação em nuvem surge como uma forma de diminuir essa ociosidade de processamento, otimizando a utilização dos recursos através da construção de ambientes virtuais.

O objetivo geral deste trabalho foi **Avaliar o Desempenho de aplicações Científicas em Ambiente de Nuvem Privada**. Para realizar a análise foram escolhidas duas aplicações científicas reais, ambas de uma classe de aplicações da área da dinâmica dos fluídos. A escolha se deve a relevância no contexto científico e por apresentarem um elevado custo computacional.

1.2 Estrutura do Texto

O restante deste trabalho esta estruturado da seguinte forma: No Capítulo 2 são introduzidos os principais conceitos de computação em nuvem, dando uma visão geral para o entendimento do restante do trabalho. O Capítulo 3 dá uma visão geral da computação científica e a possibilidade de utilização de nuvens para resolver problemas que demandam alto processamento. No final do capítulo são mostrados os principais trabalhos relacionados ao tema. No Capítulo 4 é apresentada a metodologia para avaliar o desempenho de nuvens na execução de aplicações científicas. Os resultados dos testes feitos baseados na metodologia proposta são mostrados no Capítulo 5. Por fim, no Capítulo 6, são apresentadas as conclusões e os trabalhos futuros.

2 Computação em Nuvem - Visão Geral

Este capítulo tem por objetivo introduzir os principais conceitos relacionados a Computação em Nuvem. As seção 2.1 e seção 2.2 trazem as principais definições e as características essenciais que formam a computação em nuvem. Nas seção 2.3 e seção 2.4 são apresentados os modelos serviço e implementação. Em seguida, a seção 2.5 mostra a arquitetura da computação em nuvem dividida em camadas. Por fim nas seção 2.6 e seção 2.7 são apresentadas a virtualização e as principais ferramentas de administração de nuvem.

2.1 Definições

Segundo Taurion (2009) o termo computação em nuvem surgiu em 2006, em uma palestra de Eric Schmidt do Google descrevendo como sua empresa gerenciava seus próprios *datacenters*. Alguns meses depois o termo *cloud* tornou-se mais popular, quando a Amazon anunciou sua oferta de *Elastic Computing Cloud* (EC2). Abrindo mercado para outras grandes empresas investirem na ideia.

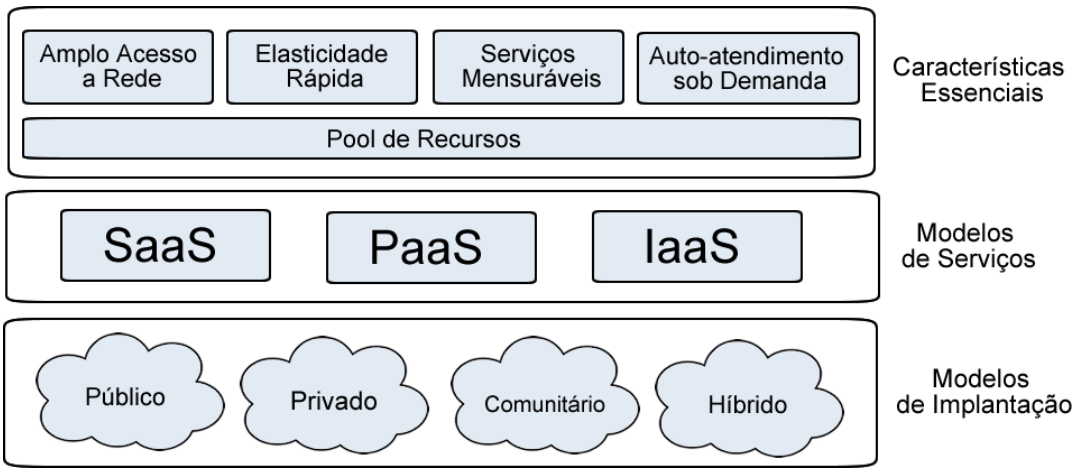
Atualmente pode ser encontrado na literatura inúmeras definições para Computação em Nuvem. Vaquero et al. (2008) faz um estudo de mais de 20 definições encontradas na literatura e a define da seguinte forma:

> "Computação em Nuvem é um conjunto de recursos virtuais facilmente usáveis e acessíveis tais como hardware, plataformas de desenvolvimento e serviços. Estes recursos podem ser dinamicamente configurados para se ajustarem a uma carga variável, permitindo a otimização do uso dos recursos. Este conjunto de recursos é tipicamente explorado através do modelo pague pelo uso com garantias oferecidas pelo provedor através de acordos de nível de serviço (*Service Level Agreement* (SLA))."

NIST – (Instituto Nacional de Padrões e Tecnologia) publicou em setembro de 2011 a versão hoje mais aceita da definição de computação em nuvem, na qual a define como um modelo que possibilita acesso, de modo conveniente e sob demanda, a um conjunto de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente adquiridos e liberados com mínimo esforço gerencial ou interação com o provedor de serviços (MELL; GRANCE, 2011).

Analisando essas e outras diversas definições podemos concluir que nenhuma delas é unânime e definitiva. Para a maioria dos autores o conceito de Nuvem esta fortemente ligado a um modelo de negócio onde se contrata serviços de processamento e armazenamento de acordo com a demanda e paga-se apenas pelo que for consumido. O NIST

Figura 1 – Definição de NIST para Computação em Nuvem



Baseado em Veras e TOZER (2012)

como mencionado acima, propõe uma definição mais abrangente incluindo a possibilidade de uma única organização realizar a virtualização de seus recursos para uso próprio. A Figura 1 resume os conceitos definidos pelo NIST que serão detalhados a seguir.

2.2 Características essenciais

Um modelo de Computação em Nuvem deve apresentar características essenciais que em conjunto as definem e diferenciam de outros paradigmas. A seguir são apresentadas as cinco principais características definidas pelo NIST.

Auto-atendimento sob demanda: permite ao usuário adquirir recursos computacionais (tempo de processamento no servidor ou armazenamento) na medida em que necessitar, sem precisar da iteração humana com os provedores de cada serviço.

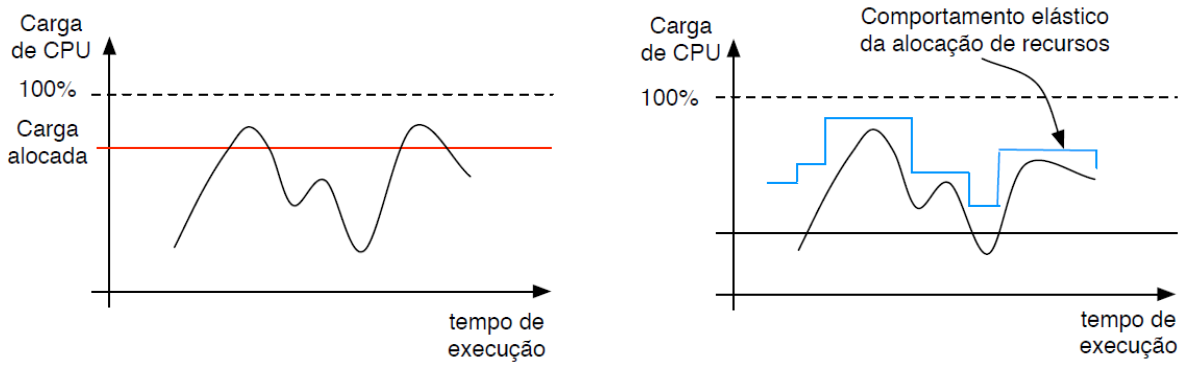
Amplo acesso a serviços de rede: recursos são disponibilizados por meio da internet e acessados através de mecanismos padronizados, que possibilitam o uso por plataformas do tipo *thin client* tais como celulares, *notebooks*, etc.

***Pool* de recursos:** recursos computacionais do provedor são dinamicamente atribuídos e ajustados de acordo com a demanda dos clientes. Estes clientes não precisam ter conhecimento da localização física dos recursos computacionais.

Elasticidade rápida: capacidade de adquirir e liberar recursos de forma rápida e elástica. Para o consumidor, a nuvem parece ilimitada e recursos podem ser adquiridos em qualquer quantidade a qualquer momento.

Serviços Mensuráveis: todos os serviços são controlados e monitorados automa-

Figura 2 – Comportamento de Infraestruturas: Tradicional e Elástica



(a) Infraestrutura Tradicional

(b) Infraestrutura Elástica.

Referência: Righi (2013)

ticamente pela nuvem de maneira que tudo fique transparente, tanto para o consumidor quanto para o fornecedor.

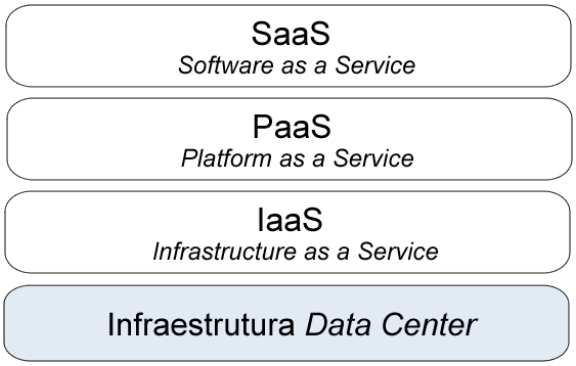
A proposta da computação em Nuvem é criar um ambiente com todas essas características envolvidas, provendo agilidade e economia na utilização do recursos. A Figura 2 ilustra bem essa vantagem. No modelo tradicional, na hora da aquisição de uma infraestrutura sempre é considerado o pior caso, com picos de utilização. No entanto, essa estimativa pode ser baixa e ocorrer momentos de falta de recursos como podemos ver em dois pontos na Figura 2(a), sem contar com o grande desperdício de recursos, dado que esses períodos de pico não representam a carga média do sistema. Já no modelo de nuvem, como ilustrado na Figura 2(b), a alocação de recursos é feita conforme a demanda requerida pelo serviço (aplicação), adicionando ou liberando recursos. Esse método traz benefícios tanto para o usuário quanto para o provedor.

2.3 Modelos de Serviços

Os modelos de serviços definidos pelo NIST são três: *Software* como Serviço - SaaS, Plataforma como Serviço - PaaS e Infraestrutura como Serviço - IaaS. A Figura 3 ilustra essa divisão em forma de pilha, sendo IaaS o modelo de mais baixo nível.

IaaS: Capacidade que o provedor tem de prover uma infraestrutura virtual, como processamento, armazenamento, rede, etc.. O usuário não administra diretamente a infraestrutura física, no entanto, tem controle total sobre a infraestrutura virtual, como sistema operacional, aplicativos e componentes de rede. O controle geralmente é feito através de protocolos de acesso remoto (*Secure Shell* (SSH)). Atualmente existe um grande número de provedores de IaaS. Os principais provedores de nuvem pública são: Amazon (AMAZON, 2015), GoGrid (GOGRID, 2015) e Rackspace (RACKSPACE, 2015). Para a implementação de IaaS privadas são disponíveis ferramentas *open source*, entre elas

Figura 3 – Divisão em camadas dos modelos de serviços



se destacam: OpenNebula (OPENNEBULA, 2015), Eucalytus (EUCALYPTUS, 2015), OpenStack (OPENSTACK, 2015) e CloudStack (CLOUDSTACK, 2015).

PaaS: O provedor oferece uma plataforma com um sistema operacional, linguagens de programação e ambientes de desenvolvimento. Neste caso o cliente será o desenvolvedor com a possibilidade de implementar, executar e disponibilizar aplicações na nuvem para terceiros. O cliente controla apenas suas aplicações implantadas, ficando por conta do provedor a administração e manutenção do ambiente (Sistema Operacional (SO), rede e armazenamento). Microsoft Azure (AZURE, 2015) e Google App Engine (APPENGINE, 2015) são dois exemplos de provedores PaaS.

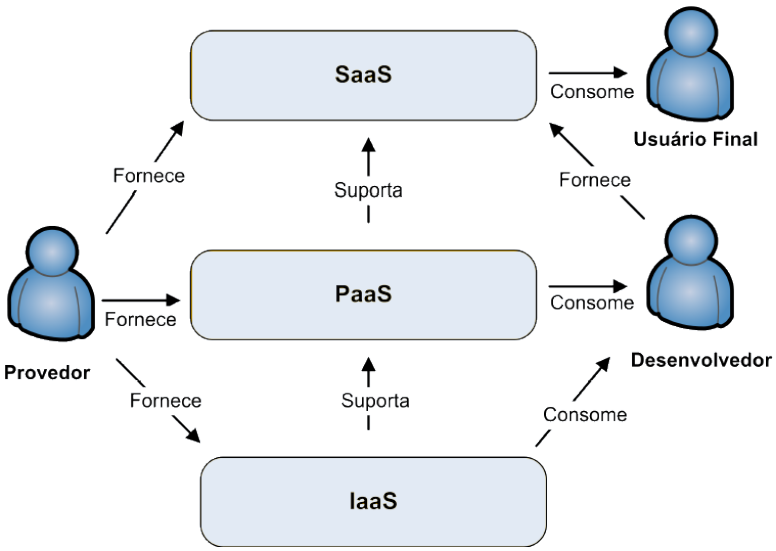
SaaS: Neste modelo os provedores oferecem aplicações como serviço. Tais aplicações podem ser acessadas a partir de diversos dispositivos (PC, *tablets* e *smartphones*) por intermédio de uma interface cliente, como um navegador Web, ou por uma *Application Programming Interfaces* (API). Este modelo representa os serviços de mais alto nível, onde o provedor tem controle total sobre a rede, sistema operacional e armazenamento. O Google Apps (GOOGLE, 2015) e o Sales Force (SALESFORCE, 2015) são exemplos de SaaS.

Para entender melhor os conceitos apresentados até aqui, podemos classificar a Computação em Nuvem em papéis (MARINOS; BRISCOE, 2009). A Figura 4 ajuda a identificar os atores e os seus diferentes interesses. O provedor pode fornecer serviços nas três modalidades (IaaS, PaaS e SaaS). Os desenvolvedores utilizam os recursos fornecidos pelo provedor e oferecem serviços para usuários finais.

2.4 Modelos de Implementação

Segundo NIST, uma nuvem pode ser implantada de quatro formas (pública, privada e híbrida) independente dos serviços prestados. Para realizar essa classificação deve-se levar em conta quem é o proprietário da infraestrutura, quem gerência, onde está

Figura 4 – Papéis de cada ator na computação em nuvem



Baseado em Marinos e Briscoe (2009)

alocada e quem acessa os serviços (MELL; GRANCE, 2011).

Nuvem Pública: No modelo de nuvem pública, toda a infraestrutura da nuvem pertence a uma organização que disponibiliza seus serviços publicamente através do modelo pague pelo uso. Este modelo deve suportar múltiplos usuários e geralmente disponibiliza apenas um modelo de serviço. A Amazon é um exemplo de nuvem pública.

Nuvem Privada: Nuvens privadas são construídas exclusivamente para uma única organização, a qual possui controle total sobre os recursos e aplicações implementadas. Diferentemente da nuvem pública, uma nuvem privada não disponibiliza seus serviços para uso geral.

Nuvem Comunitária: O modelo de implementação comunitário possui uma infraestrutura compartilhada por diversas organizações que normalmente possuem interesses comuns, como requisitos de segurança, políticas, aspectos de flexibilidade e/ou compatibilidade. Pode ser administrado pelas próprias organizações ou por terceiros.

Nuvem Híbrida: No modelo de implantação híbrido, existe uma composição de duas ou mais nuvens, que podem ser privadas, comunitárias ou públicas e que permanecem como entidades únicas e ligadas por uma tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicações.

A Tabela 1 faz uma breve comparação das principais características dos modelos de implantação. A escolha de um modelo depende da necessidade da organização, levando em consideração custo, segurança e controle dos recursos. As nuvens públicas são uma interessante alternativa para quem não possui uma infraestrutura mas precisa

Tabela 1 – Modelos de Implementação.

	Gerência	**Propriedade**	**Segurança**
Pública	Terceiros	Terceiros	Baixa
Privada	Própria	Própria ou Terceiros	Alta
Comunitária	Própria ou Terceiros	Própria ou Terceiros	Média
Híbrida	Própria ou Terceiros	Própria ou Terceiros	Média

de processamento ou armazenamento. Nas nuvens públicas não são necessários investimentos iniciais, nem em espaço físico, energia, ou pessoal especializado. Os recursos são adquiridos de forma rápida e liberados quando não forem mais necessários de acordo com a necessidade do usuário que paga apenas o que é consumido.

Caso a organização preze por maior controle, ela poderá implementar uma nuvem privada. Na implantação deste modelo de nuvem, a organização deve adquirir ou já possuir os recursos computacionais. Ao contrário de uma nuvem pública, os recursos são disponibilizados apenas para os membros da organização. Este modelo compreende todas as características de uma nuvem pública, porém se restringe apenas aos recursos pertencentes a própria infraestrutura, fazendo com que seu limite de escalabilidade seja sua capacidade total. O modelo privado consegue melhorar significativamente a utilização dos recursos, diminuindo a ociosidade média dos servidores (superior a 85%), gerando economia de energia, aumentando a segurança e a velocidade com que os recursos são provisionados para seus usuários. As nuvens híbridas podem trazer os benefícios de ambos modelos, ou ainda pode-se implementar nuvens comunitárias, voltadas para organizações com interesses em comum.

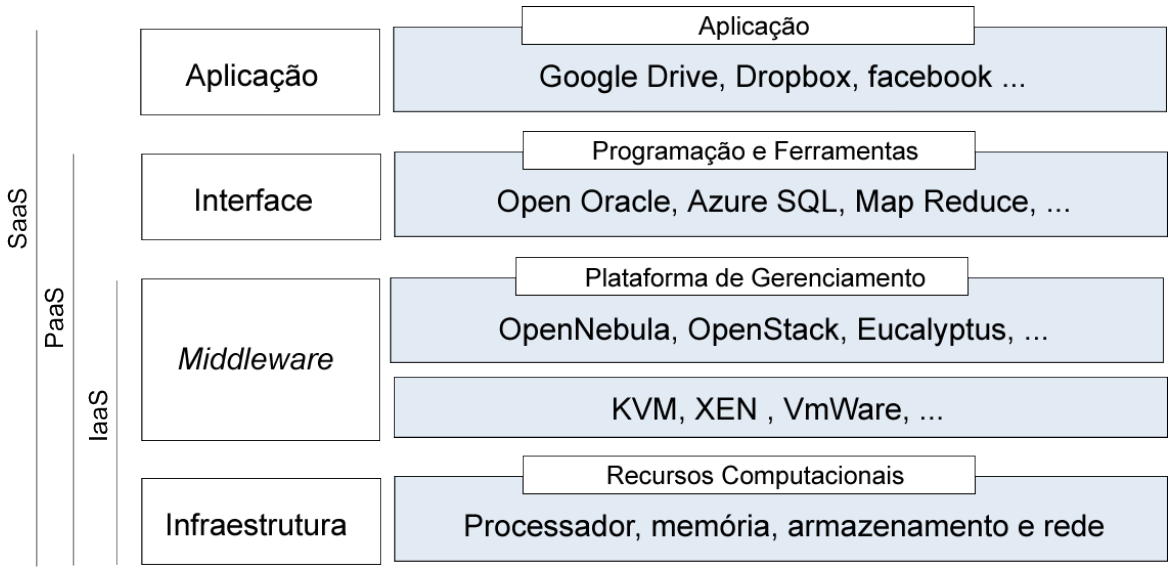
2.5 Arquitetura de Nuvem

A Computação em Nuvem é baseada em uma arquitetura dividida em camadas, onde cada uma delas atua independente e fica responsável por uma parte da infraestrutura (VECCHIOLA; PANDEY; BUYYA, 2009). A Figura 5 ilustra essa divisão, sendo que a camada de mais baixo nível é composta pela infraestrutura de hardware, seguindo pela camada de *middleware*, interface e aplicação de usuário.

A camada denominada infraestrutura é composta pelos recursos físicos, memória, armazenamento, processamento e rede. Esses recursos físicos geralmente são formados por grandes *datacenters* centralizados ou compostos por recursos heterogêneos formados por servidores, organizados em *clusters*. Desta forma, organizações com uma infraestrutura já estabelecida podem migrar facilmente para nuvem.

O *middleware* é uma camada de *software* responsável por fornecer um núcleo lógico e gerenciar toda infraestrutura de hardware. Seu objetivo é explorar de maneira eficaz

Figura 5 – Arquitetura em camadas da computação em nuvem



Referência baseada em (CARISSIMI, 2015)

a utilização dos recursos. Na Figura 5 observa-se que a camada de *middleware* pode ser dividida em duas subcamadas: a primeira, através de um *software* denominado *hipervisor*, fica responsável pela virtualização dos recursos; a segunda, é composta por uma ferramenta de administração de nuvem, na qual tem a responsabilidade de autenticar usuários, permitir a solicitação de recursos virtuais e configurá-los (CARISSIMI, 2015).

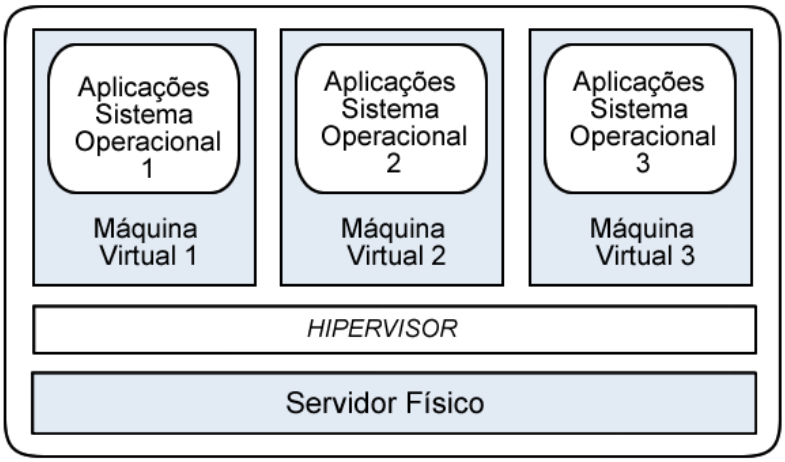
A camada plataforma de Computação em Nuvem oferece todos os recursos necessários para o desenvolvimento de aplicações, como compiladores, linguagem de programação e bibliotecas. Windows Azure e Google AppEngine são exemplos de plataformas. Por fim a camada de aplicação oferece *softwares* como serviços.

2.6 Virtualização

A virtualização é uma camada de abstração localizada entre o *hardware* e o sistema operacional, capaz de controlar o acesso direto do *software* aos recursos físicos (VERAS; TOZER, 2012). A camada de virtualização também conhecida como *hipervisor* ou monitor de máquina virtual (*Virtual Machine Monitor* (VMM)) permite que um servidor físico virtualizado possa executar vários servidores virtuais, também chamados de máquinas virtuais (VMs). A Figura 6 ilustra um exemplo genérico de virtualização.

Cada máquina virtual tem seu próprio *hardware* virtual (*Central Processing Unit* (CPU), memória e disco) com um sistema operacional e aplicações executando sobre ele. Segundo Carissimi (2008) as máquinas virtuais podem ser implementadas como uma

Figura 6 – Exemplo genérico de virtualização



aplicação de um sistema operacional (máquina virtual Java), ou formarem uma camada de *software* posicionada entre o *hardware* e o sistema operacional (*hipervisor*).

O *hipervisor* pode prover a virtualização de duas formas: virtualização total ou paravirtualização. Na virtualização total, o *hipervisor* realiza uma réplica de toda a estrutura de *hardware*, de tal forma que o sistema operacional e aplicações possam executar como se estivessem diretamente no *hardware* original (LAUREANO, 2006). A vantagem é o fato de o sistema a ser virtualizado não sofrer qualquer tipo de alteração. No entanto, essa abordagem acarreta em uma perda de desempenho, pois todos acessos ao *hardware* são intermediados pelo *hipervisor* (CARISSIMI, 2008).

A paravirtualização é utilizada como uma alternativa para problemas de desempenho encontrados na virtualização total. Na paravirtualização, o sistema é modificado para chamar o *hipervisor* sempre que executar uma instrução que possa alterar o estado do sistema. Assim não é necessário que o *hipervisor* teste instrução por instrução, como é o caso da virtualização total. Outro fator é o acesso de *drivers* da própria máquina virtual, ao contrário da virtualização total que utiliza *drivers* genéricos e diminuem o desempenho.

A virtualização é a tecnologia base da Computação em Nuvem, pois proporciona um alto grau de portabilidade e de flexibilidade, permitindo que várias aplicações, de sistemas operacionais diferentes, executem em um mesmo *hardware*. Essa execução é denominada consolidação de servidores (SMITH; NAIR, 2005). Essa abordagem é bastante interessante, pois ao invés de alocar uma máquina física para cada cliente, o provedor pode instanciar uma máquina virtual.

Outra fator importante possível com a virtualização do *datacenter* é realizar o balanceamento de carga. Se uma máquina física estiver sob baixa carga de trabalho, é

possível migrar as máquinas virtuais para outras máquinas físicas, desligando as de baixa carga, gerando melhor utilização dos recursos e economia de energia.

Existem diversos *hipervisores* disponíveis no mercado. A seguir serão apresentados: Xen (XEN, 2015) utilizado pela Amazon, principal provedora pública; KVM (KVM, 2015) amplamente utilizado por ferramentas de *open source*; e VMware (VMWARE, 2015) *hipervisor* proprietário também muito utilizado.

2.6.1 Xen

Xen (XEN, 2015) é um *hipervisor* desenvolvido pelo laboratório de computação da Universidade de Cambridge, disponibilizado nos termos da GNU *Generical Public License* (GPL). Xen é nativo para plataformas x86, tradicionalmente implementa a paravirtualização e pode suportar uma variedade de sistemas operacionais (Windowns, GNU/Linux, Solaris, etc.).

Xen conta com a virtualização total, desde que as máquinas possuam processadores com suporte na paravirtualização, para evitar o desenvolvimento de *drivers* específicos, Xen utiliza uma abordagem alternativa, na qual o sistema operacional que possui o *hipervisor* é denominado Dom0 (domínio 0) e as máquinas virtuais são chamadas de Dom1 (domínio 1). O núcleo da máquina de Dom1 é modificado de modo que não possa acessar diretamente o *hardware*. Somente o Dom0 tem acesso direto ao *hardware*.

2.6.2 VMware

VMware (VMWARE, 2015) é um *hipervisor* proprietário para a plataforma x86. Possui uma implementação completa, possibilitando utilizar a tecnologia de paravirtualização e virtualização total. Os produtos disponibilizados pelo VMware dividem-se em três categorias: gerenciamento e automação, intra-estrutura virtual e plataformas de virtualização.

Na arquitetura do VMWare, a virtualização ocorre a nível de processador, o qual é executado como se fosse um programa no espaço de aplicação. Para ter acesso mais rápido aos dispositivos, VMWare instala um *driver* especial que suporta um amplo conjunto de dispositivos para a arquitetura x86.

2.6.3 KVM

O *hipervisor* KVM (KVM, 2015) foi criado em 2007 pela empresa Red Hat. Possui suporte a virtualização total sem necessidade de modificação especial. KVM é disponibilizado nos termos da GNU GPL.

KVM atua diretamente no *kernel* do sistema operacional, tratando o *kernel* do Linux como um *hipervisor*. KVM utiliza virtualização completa para virtualizar memória e CPU. Somente nos dispositivos de E/S é que a paravirtualização é usada. Isso acontece a fim de tentar melhorar o desempenho de tais dispositivos. O método aplicado a KVM se diferencia das ferramentas existentes no mercado devido ao local em que KVM opera seus serviços de virtualização e por ocupar pouco espaço no disco do servidor, pois existe uma grande reutilização de recursos do sistema operacional Linux.

2.7 Ferramentas *open source* de Administração de IaaS

Ambientes de Computação em Nuvem possuem uma grande quantidade de recursos computacionais, tais como: máquinas físicas, sistemas de armazenamento, *hipervisores* executando sobre máquinas físicas. Todos esses componentes devem ser instalados, configurados e gerenciados. Isso é feito através de ferramentas de administração de Nuvem.

Atualmente existem diversas ferramentas *open source* com o objetivo de prover o gerenciamento de ambientes virtualizados. Essas ferramentas tendem a ser uma ótima alternativa para implantação de uma nuvem privada, em organizações que desejam realizar a virtualização de seus *datacenters*. Cada ferramenta possui diferentes componentes e características. A seguir, duas ferramentas de gerenciamento são analisadas.

2.7.1 OpenStack

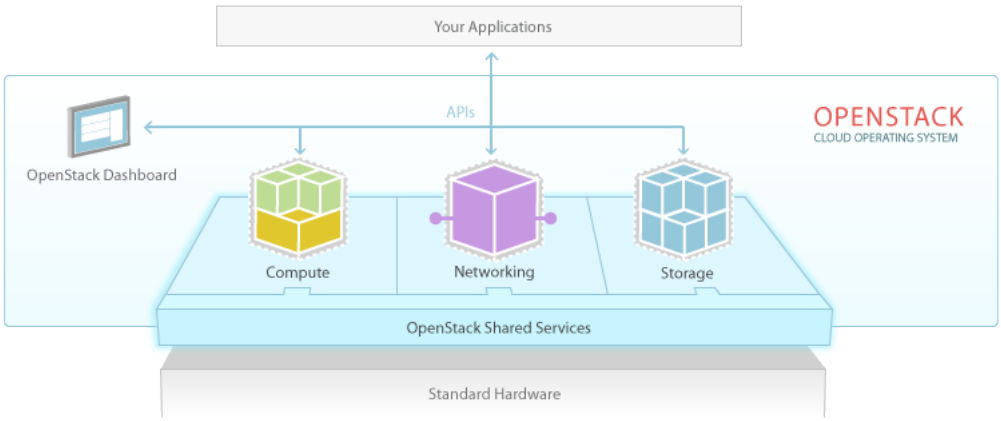
OpenStack (OPENSTACK, 2015) é um projeto entre Rackspace e NASA com objetivo de prover soluções completas para a implementação de uma infraestrutura de nuvem. A versão mais recente oferece uma série de facilidades para o gerenciamento de máquinas virtuais, capacidade de computação, armazenamento e recursos de rede.

A ferramenta OpenStack oferece ainda suporte aos principais *hipervisores* do mercado: Xen, KVM, VMware, Hyper-V. A administração de suas máquinas virtuais pode ser feita por linha de comando ou através de uma aplicação web (*Dashboard*).

A arquitetura do OpenStack é formada por uma série de componentes. Estes por sua vez podem ser divididos em três partes: *Compute*, *Storage* e *Networking*. Cada uma dessas partes possui um ou mais subcomponentes responsáveis por um papel dentro da nuvem, como podemos ver na Figura 7.

Compute: Composto por dois componentes: *Nova* e *Glance*. O componente *Nova* realiza o controle de todas as atividades necessárias para instanciar as máquinas virtuais. Para tanto, *Nova* possui vários sub-componentes ligados ao gerenciamento de instâncias de nuvem: *Nova-API* responsável pela comunicação privilegiada entre componentes e prover uma interface para comunicação com provedores externos; *Nova-Sheduler* aloca

Figura 7 – Arquitetura do OpenStack



Referência: (OPENSTACK, 2015)

e gerência novas instâncias; *Nova-Compute* controla o ciclo de vida das instâncias de máquinas virtuais; *Nova-Network* trata da configuração de rede das máquinas virtuais; *Nova-conductor* gerencia a iteração com banco de dados; *Nova-consoleauth* realiza as autorizações de acessos.

O componente *Glance* é responsável por fornecer serviços para registrar e recuperar imagens de máquinas virtuais. *Glace* fornece uma interface padrão para consultar informações sobre imagens de disco virtual armazenadas em diferentes formatos. Essas imagens são armazenadas no componente *Swift*.

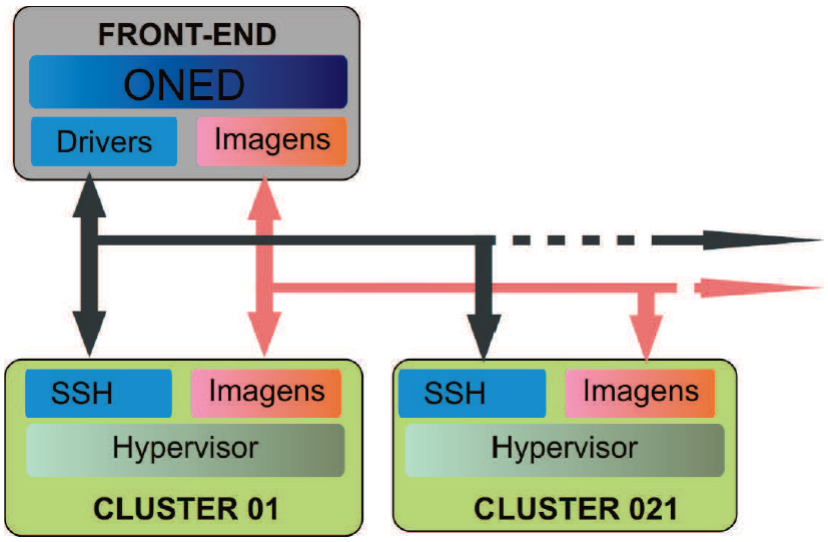
Storage: As unidades de armazenamento utilizadas pelo OpenStack são realizadas com *Cinder* e *Swift*. O componente *Cinder* implementa o armazenamento através de blocos. Os usuários acessam os dados montando um dispositivo remoto de forma semelhante como é montado um disco físico local. *Swift* implementa o armazenamento através de objetos, geralmente utilizado para grande volumes de dados.

Networking: O gerenciamenteo das funcionalidades de redes é realizado pelo componente denominado *Neutron*. Este, por sua vez define uma infraestrutura de rede própria (VLANs e VPNs), como também a integração a outros componentes de redes.

2.7.2 OpenNebula

OpenNebula (OPENNEBULA, 2015) é uma ferramenta para a administração de nuvem que surgiu em 2005. Pode ser instalada em qualquer distribuição GNU Linux a partir do seu código fonte ou através dos repositórios oficiais de algumas distribuições específicas (Debian, openSUSE, Ubuntu). É compatível com os *hipervisores* Xen, KVM e VMware.

Figura 8 – Arquitetura do OpenNebula



Referência: (OPENNEBULA, 2015)

A arquitetura de OpenNebula é semelhante as arquiteturas de *clusters* tradicionais, onde um nó do conjunto de máquinas é escolhido como controlador principal, denominado *front-end*. Este por sua vez fica responsável por controlar os demais nós, que são responsáveis por executar as máquinas virtuais solicitadas. A Figura 8 ilustra os componentes da arquitetura do OpenNebula.

Podemos observar na Figura 8, que o *front-end* será responsável por executar os principais componentes da ferramenta, relacionados a rede, armazenamento e *hipervisores*. O *front-end* é responsável por executar o OpenNebula *Daemon* (*oned*). Este por sua vez controla o ciclo de vida das VMs, rede e disco. O *front-end* executa ainda os drivers que tem a função de atuar como repositório de imagens das máquinas virtuais.

Para realizar o gerenciamento e monitoramento dos nós compostos pela nuvem, o OpenNebula *Daemon* utiliza suas operações através de comandos SSH. OpenNebula possibilita através do uso do *Network File System* (NFS) o armazenamento centralizado de arquivos e informações, oferecendo serviços de *storage*. Isso é possível pois o *front-end* compartilha uma unidade de rede entre todos os nós do *cluster*.

2.8 Conclusão do capítulo

O objetivo deste capítulo foi apresentar uma visão geral sobre a computação em nuvem, fornecendo conceitos básicos necessários para o entendimento do restante deste trabalho. Foram mostrados as características essenciais, os modelos de serviço e implanta-

ção. Foi apresentado também os principais virtualizadores e ferramentas de administração de nuvem, capazes de criar e gerenciar ambientes virtualizados.

3 Computação Científica em Nuvem

Este capítulo tem por objetivo apresentar os benefícios e problemas da computação em nuvem no meio científico. Para tanto a seção 3.1 apresenta uma revisão geral de conceitos sobre computação de alto desempenho, arquiteturas paralelas e programação paralela. A seção 3.3 estabelece essa relação destacando os benefícios e problemas encontrados. Por fim, na seção 3.4, são apresentados os principais trabalhos relacionados a esta pesquisa.

3.1 Computação de Alto Desempenho

Computação de alto desempenho (em inglês, *High Performance Computing* (HPC)) é uma importante área da computação que tem por objetivo reduzir o tempo de execução de aplicações que demandam alto poder de processamento (BUYYA, 1999). São inúmeras aplicações em diversas áreas de pesquisa que tem se beneficiado da HPC. Tais aplicações, denominadas científicas, podem simular eventos naturais tais como a previsão do tempo, dinâmica dos fluídos, genética ou até mesmo gerar imagens da subsuperfície da terra para localizar petróleo.

Por muitos anos, o principal alvo para ampliar o desempenho dos computadores era aumentar a frequência do processamento. Devido a limites físicos, problemas de calor e alto consumo de energia, a solução encontrada pela indústria foi investir em arquiteturas paralelas formadas por múltiplas unidades de processamento. No entanto, o aumento no número de processadores na arquitetura acaba dificultando a programação, visto que o problema deve ser dividido e distribuído entre as unidades de processamento disponíveis. Esta seção apresenta conceitos básicos sobre arquiteturas paralelas, o modelo de programação paralela e as principais interfaces de programação.

3.1.1 Arquiteturas Paralelas

Arquiteturas paralelas são exploradas em diferentes níveis de paralelismo e podem ser classificadas quanto ao compartilhamento de memória (WILKINSON; ALLEN, 2005). As arquiteturas que possuem memória compartilhada entre as unidades de processamento são chamados de multiprocessadores. Já as arquiteturas que possuem sua memória distribuída entre computadores independentes, interconectados através de uma rede, são denominados multicomputadores.

A comunicação entre os processos em multiprocessadores é realizada através da leitura e escrita (*load* e *store*) em um espaço compartilhado de memória. As vantagem

dessa arquitetura estão relacionadas a velocidade e simplicidade dos programas em relação ao modelo distribuído. A principal desvantagem é a escalabilidade limitada, uma vez construída a arquitetura não será mais possível adicionar processadores.

Atualmente o mercado de processadores vem investindo em arquiteturas *multi-core* formadas por múltiplos núcleos de processamento integrados em um único *chip*. Nesse tipo de arquitetura cada núcleo é replicado e desempenha a função de um processador, fazendo com que vários processos leves (*threads*) possam ser executados simultaneamente de acordo com o total de núcleos existentes. Atualmente é possível encontrar processadores com diversos núcleos (*cores*), como é o caso da arquitetura *Xeon Phi* da Intel com modelos de 57, 60 e 61 *cores*.

Em arquiteturas multicomputador, cada computador é formado por unidade de processamento e memória próprios, na qual apenas ele tem acesso (ROSE; NAVAUX, 2002). Como cada computador possui sua própria região de memória, a comunicação entre processadores é feita via troca de mensagens, através de uma rede de interconexão. A grande vantagem desta arquitetura é a possibilidade expandir ou reduzir quantas unidades de processamento forem necessárias.

Uma solução simples e viável para implementar multicomputadores são as arquiteturas denominadas *clusters* (ou agregados de computadores). Segundo Buyya (1999) uma arquitetura do tipo *cluster* pode ser definida como um conjunto de máquinas ou nós independentes que cooperam entre si na execução de aplicações utilizando troca de mensagens. *Clusters* geralmente são construídos com computadores convencionais, os quais podem ser adicionados várias máquinas simples para trabalhar em conjunto como se fossem uma única máquina de grande porte. Um tipo de *cluster* bastante conhecido é da classe *Beowulf*, na qual diversos nós escravos são gerenciados por um só computador.

Com objetivo de agregar ainda mais desempenho aproveitando recursos distribuídos, arquiteturas do tipo *grid* (ou grades computacionais) foram desenvolvidas. Segundo Foster e Kesselman (2003) *grids* tem por objetivo agregar recursos heterogeneos e fisicamente distribuídos entre diferentes organizações, oferecendo grande capacidade de processamento (*TeraFlops*), que dificilmente seriam obtidos com sistemas centralizados. Se comparado a outras arquiteturas paralelas, ambientes de *grids* possuem uma enorme complexidade em sua implementação, dentre os fatores, se destacam a heterogeniedade e a dificuldade em gerenciar os recursos. Para tentar minimizar esses problemas e tornar a computação em *grid* mais popular, existem diversos projetos de pesquisa, sendo que o mais popular é o Globus (GLOBUS, 2015), um *middleware* capaz de gerenciar recursos, tarefas e usuários.

Atualmente existe um grande número de sistemas computacionais com uma grande poder computacional. Isso pode ser facilmente examinando na lista dos TOP 500 (TOP500, 2015) supercomputadores com maior poder de processamento da atualidade. Tais super-

computadores são construídos sob arquiteturas paralelas, com diversos níveis de paralelismo. O Tianhe-2, atual supercomputador mais rápido do mundo, possui 16 mil nós de processamento. Cada nó é composto por dois processadores Intel com 12 *cores* físicos e 3 coprocessadores Intel *Xeon Phi* com 57 *cores* cada, totalizando mais de 3,1 milhões de núcleos.

3.1.2 Programação Paralela

Tradicionalmente um programa é construido sequencialmente para ser executado em uma única unidade de processamento. A programação paralela surge para utilizar os múltiplos elementos de processamentos de arquiteturas paralelas. Isso é possível ao quebrar um programa em partes independentes de forma que cada elemento de processamento possa executar sua parte simultaneamente com outros. Os elementos de processamento devem cooperar entre si utilizando primitivas de comunicação.

Existem basicamente duas formas de explorar a programação paralela. São o paralelismo em nível de dados e em nível de tarefas (WILKINSON; ALLEN, 2005). O paralelismo de dados é feito através da divisão do conjunto de dados, cada parte dessa divisão forma um subconjunto menor para ser atribuído a cada processo. Esta é uma forma simples de ser implementada, pois não possui dependência entre as operações e geralmente utilizam pouca comunicação entre os processos. No paralelismo de tarefas o problema é dividido em tarefas independentes atribuídas a cada processo, nas quais operam sobre um mesmo conjunto de dados. Geralmente tarefa possuem dependência entre suas operações, tornando elevado a comunicação entre processos.

Ao construir programas paralelos deve-se ter o conhecimento da arquitetura computacional que será utilizada. Genericamente, os modelos computacionais são divididos em memória compartilhada (*multi-core*) e memória distribuída (*clusters* e *grids*). Para facilitar a implementação desses ambientes foram criadas interfaces de programação paralela. Uma interface popular que explora o paralelismo em memória compartilhada é a API *Open Multi-processing* (OpenMP), para memória distribuída temos a biblioteca de comunicação *Message Passing Interface* (MPI). Nas próximas seções serão detalhadas ambas interfaces.

Programar em paralelo não é uma tarefa trivial e requer certo nível técnico. Particionamento de código, divisão adequada de trabalho entre os recursos, comunicação e concorrência são alguns dos cuidados que devem ser tomados. Para facilitar a programação e fazer o uso eficiente de arquiteturas paralelas, podem ser adotadas diferentes modelos de programação (FOSTER, 1995), listadas a seguir:

Mestre-Escravo. Modelo formado por um processo mestre e múltiplos processos escravos. O mestre é responsável por dividir o problema em pequenas partes, e distribuir

entre os processos escravos. Os processos escravos recebem as tarefas, executam, e enviam o resultado ao processo mestre. A comunicação é feita geralmente entre mestre e escravos. Ao final o processo mestre reúne as mensagens recebidas e gera a solução do problema.

Decomposição de Dados ou Decomposição de Domínios. Neste modelo cada processo executa o mesmo trecho de código em partes diferentes de dados. Os dados a serem manipulados são divididos e distribuídos entre os processadores. Após a divisão, os resultados parciais podem precisar de comunicação pois podem haver dependências entre as partes segmentadas.

Pipeline. Neste caso os processos são organizados de uma forma que cada processo corresponde a um estágio de uma fila de execuções. Cada estágio opera sobre todo o problema, repassando as informações aos estágios subsequentes.

Divisão e Conquista. A ideia é dividir um problema em problemas menores. Estes problemas são distribuídos entre os processos disponíveis, no final o resultado é combinado para se obter a resposta.

Bag of Task. Conjunto de tarefas que podem ser executadas de forma independente, ou seja não há a necessidade de comunicação durante cada execução.

3.2 Interfaces de Programação Paralela

Para o desenvolvimento de programas paralelos costuma-se utilizar alguma interface de programação paralela. Entre estas, destacam-se OpenMP, para memória compartilhada, e MPI, para memória distribuída.

3.2.1 OpenMP

OpenMP (*Open Multi-Processing*) (CHAPMAN; JOST; PAS, 2007) é uma API criada para facilitar a programação paralela em memória compartilhada. Nasceu da cooperação de um grupo de grandes fabricantes de *hardware* e *software*, na qual se incluem Intel, AMD, IBM, HP, Sun, entre outras. Projetada para ser utilizada nas linguagens C/C++ e Fortran, OpenMP consiste em um conjunto de diretivas para o compilador, funções de biblioteca e variáveis de ambiente que permitem facilmente transformar um programa sequencial em um programa paralelo.

Os programas desenvolvidos com OpenMP utilizam um modelo chamado *fork-join*. O programa inicia a execução com uma *thread* principal (*master*). A *thread master* executa suas operações de forma sequencial até encontrar um construtor paralelo. Ao encontrar um construtor (*parallel*), o programa cria (*fork*) um grupo de *threads* que executam em um bloco de código estruturado. Após concluir a execução paralela (final do bloco) o grupo de *threads* sincroniza (*join*) e continua sua execução sequencial.

3.2.2 MPI

MPI (*Message Passing Interface*) é uma biblioteca padrão de comunicação para ambientes de memória distribuída que foi definida através da colaboração de um conjunto de fabricantes e instituições de pesquisa da área de processamento de alto desempenho (MPI-FORUM, 2015). A biblioteca MPI pode ser implementada para diversos tipos de máquinas, possui 125 funções para programação nas linguagens Fortran 77, Fortran 90, ANSI C e C++ (GROPP et al., 1996).

Um programa MPI é definido como uma coleção de processos que operam sobre um código fonte. Todo paralelismo fica de responsabilidade do programador, que deve identificar quem é o processo responsável pela execução de cada parte do código. É com essa identificação que os processos conseguem realizar trocas de mensagens para se comunicar. Atualmente existem inúmeras implementações do MPI, dentre elas se destacam OpenMPI (OPEN-MPI, 2015) e o Mpich (MPICH, 2015).

3.3 Computação em Nuvem para HPC

A demanda crescente por recursos computacionais devido a avanços científicos e tecnológicos, vem gerando uma busca constante por sistemas de alto desempenho. Como vimos na subseção 3.1.1 com advento das arquiteturas paralelas diversos modelos computacionais foram desenvolvidos ao longo do tempo. Modelos mais recentes tais como *clusters* e *grids*, e os tradicionais supercomputadores são exemplos de plataformas de alto desempenho amplamente utilizadas.

Um paradigma recente que vem ganhando grande visibilidade por parte da comunidade científica é computação em nuvem. Devido as suas características e seus inúmeros benefícios, como a elasticidade e a redução de custos, a computação em nuvem tem se mostrado uma interessante alternativa para execução de aplicações científicas, beneficiando desde usuários que possuem pequenas aplicações, até aqueles que executam seus experimentos em centros de supercomputação.

Se comparado a modelos tradicionais como *clusters* e supercomputadores, a computação em nuvem pode ser considerada mais barata e escalável, pois seus recursos podem ser facilmente adquiridos de acordo com os requisitos das aplicações e condições financeiras da organização (OSTERMANN et al., 2010). Quando comparada a *grids* a computação em nuvem possui alguns objetivos similares como o compartilhamento de recursos e redução de custos. No entando, nuvens são mais simples de implementar e serem configuradas, com a possibilidade de ajustar facilmente os recursos de acordo com a demanda da aplicação (VECCHIOLA; PANDEY; BUYYA, 2009).

Tabela 2 – Comparação de Trabalhos Relacionados.

Autores	**Foco**	**Ambiente**	**Avaliação**
Sousa et al. (2012)	Nuvem Privada	Eucalyptus e KVM	*Benchmarks*
AL-Mukhtar e Mardan (2014)	Nuvem Privada	CloudStack e KVM	*Benchmarks*
Beserra et al. (2015)	Virtualização	KVM e VirtualBox	*Benchmarks*
Chakthranont et al. (2014)	Nuvem Privada	Apache CloudStack e KVM	*Benchmarks*
Tudoran et al. (2012)	Nuvem Públida e Privada	Azure e Nimbus	Aplicação Real *Benchmarks*
Alves e Drummond (2014)	Nuvem Públida	Amazon EC2 e Microsoft Azures	Aplicação Real *Benchmarks*
(MARON et al., 2015)	Nuvem Privada	OpenNebula, OpenStack e KVM	*Benchmarks*

3.4 Trabalhos Relacionados

Atualmente inúmeros pesquisadores vem investigando o desempenho de IaaS. De um modo geral, a maioria dos trabalhos encontrados na literatura avaliam o desempenho de virtualizadores e ambientes públicos de nuvem. Podemos perceber também que para realizar seus experimentos geralmente utilizam *benchmarks* de forma isolada. A Tabela 2 faz uma breve comparação dos principais trabalhos relacionados com esta pesquisa, considerando o foco, ambiente e meio de avaliação.

Segundo Beserra et al. (2015) infraestruturas virtualizadas tendem a ser uma alternativa de baixo custo para implantar aplicações de alto desempenho. No entanto a virtualização traz alguns desafios, especialmente no que diz respeito à sobrecarga causada pelos *hipervisores*. Para tanto, os autores avaliam o desempenho de dois *hipervisores* (KVM e VirtualBox) para HPC. Para realizar a análise utilizaram o *benchmark* HPC *Challenge* (HPCC) para avaliar processador, RAM, comunicação entre processos e desempenho de comunicação de rede. Como resultados os autores concluiram que o KVM utilizando paravirtualização possui um desemepenho similar ao nativo.

Para Tudoran et al. (2012), os cientistas se deparam com a dificuldade em avaliar a variedade de tipo e modelo de nuvem para as necessidades de suas aplicações. Para tanto este trabalho faz uma avaliação de duas plataformas de nuvem, uma pública (nuvem Azure) e outra privada (ferramenta Nimbus). A análise é feita considerando as necessidades primárias de aplicações científicas (poder de computação, armazenamento, transferência de dados e custos). A avaliação é feita utilizando ambos os *benchmarks* sintéticos e uma aplicação na vida real. Os resultados mostraram que Nimbus possui menos variabilidade de desempenho e maior suporte a aplicações intensivas de dados, enquanto

Azure implanta mais rápido e com custo mais baixo.

Alves e Drummond (2014) realizaram uma análise de desempenho de um simulador de reservatórios de petróleo com o objetivo de avaliar o comportamento de tal aplicação científica em um ambiente de computação em nuvem providos pelas plataformas Amazon EC2 e Microsoft Azure. Para realizar os testes foi analisado métricas do Sistema Operacional e a execução de *benchmarks* específicos. Os resultados mostraram um decréscimo significativo no tempo devido ao *overhead* de virtualização e o compartilhamento de recursos.

O objetivo de Sousa et al. (2012) foi realizar uma avaliação de desempenho de uma nuvem privada gerenciada pela ferramenta Eucalyptus. A avaliação foi feita considerando CPU e armazenamento. Para realizar a análise utilizaram diferentes cargas de trabalho baseadas nos *benchmarks* Bonnie++ (Disco) e Linpack (Processador). Segundo os autores a computação em nuvem permite que as organizações concentrem seus esforços nos negócios, em vez da gestão de infraestruturas complexas.

AL-Mukhtar e Mardan (2014) constroem uma nuvem privada afim de avaliar a ferramenta CloudStack. Os autores fazem uma análise de máquinas virtuais gerenciadas pela ferramenta OpenStack quanto ao uso de CPU (Linpack), largura de banda de memória (*Stream*), rede (*Iperf*) e disco (*Bonnie++*). A análise é feita utilizando diferentes operações de gerenciamento de máquinas virtuais, tais como adicionar, excluir e migrar. Com os resultados os autores concluiram que CloudStack possui uma arquitetura interna bem definida com estabilidade e desempenho no gerenciamento das VMs.

Em Chakthranont et al. (2014) criam uma nuvem privada utilizando um supercomputador. A criação das máquinas virtuais foi feita com o *hipervisor* KVM combinado com a ferramenta OpenStack para gerencia do ambiente. A análise foi feita comparando o ambiente nativo com o virtual, utilizando *micro-benchmarks* (Intel *Micro Benchmark*) e *benchmarks* baseados em aplicações reais (HPC Challenge (HPCC), OpenMX e Graph500). O desempenho é medido por *micro-benchmarks* HPC em ambas plataformas (*cluster* físico e virtual). Os resultados dos *micro-benchmarks* HPC indicaram que *clusters* virtuais sofrem o problema da escalabilidade em quase todas as funções MPI coletivas com o aumento dos nós. No entanto, outros *benchmarks* baseados em aplicações reais mostraram que a sobrecarga de virtualização é de cerca de 5%, mesmo quando o número de nós aumentar a 128.

Maron et al. (2015) mostraram através de experimentos que cada ferramenta de administração de nuvem possui características específicas que podem influenciar no desempenho. Foram realizados testes de desemepenho nas ferramentas OpenStack e OpenNebula, com OpenNebula alcançando melhores resultados. Segundo os autores as possíveis perdas de desempenho do OpenStack nas execuções com OpenMP se deve ao componente *Nova-compute-KVM*, que na versão *Havana*, grava em tempo real informações das ins-

tâncias em um banco de dados SQL, o qual também necessita do auxilio do *middleware RabbitMQ* para realizar esta tarefa. No MPI além do *Nova-compute-KVM* outro fator que pode ter influenciado está no componente *Neutron* e o *plugin OpenvSwitch*, que cria interfaces virtuais e um tunelamento exclusivo para comunicação de alguns serviços, inserindo camadas adicionais na comunicação de rede. Enquanto o OpenNebula usa apenas um driver básico (*Dummy*), o qual não exerce regras específicas para o tráfego de rede.

A computação em nuvem vem sendo alvo de inúmeras pesquisas, no entanto, em termos de avaliação de alto desempenho ainda é um assunto que precisa ser explorado. Assim como os trabalhos relacionados apresentados, este visa contribuir com análise de desempenho de IaaS para HPC. Como podemos observar na Tabela 2 poucos trabalhos utilizam aplicações científicas reais para avaliação, sendo os que utilizam, realizam seus experimentos em nuvens de domínio público. Diferente dos demais, avaliamos duas aplicações científicas reais da área da dinâmica dos fluídos em ambiente de nuvem privada.

3.5 Conclusão Capítulo

Este capítulo teve por objetivo apresentar a computação em nuvem no meio científico. Inicialmente foi feita uma revisão geral de conceitos sobre computação de alto desempenho, arquiteturas paralelas, programação paralela e ainda descrevemos brevemente as interfaces de programação mais populares. Após, apresentamos os benefícios e desafios da computação em nuvem para resolver problemas que demandam alto desempenho. Por fim, foram apresentados os principais trabalhos relacionados, destacando o diferencial dessa pesquisa e suas contribuições.

4 Metodologia de Avaliação

Neste capítulo é apresentada a metodologia utilizada para avaliação de desempenho. Como proposto anteriormente, o objetivo deste trabalho é avaliar o desempenho de aplicações científicas em ambientes de computação em nuvem. Para tanto na seção 4.1 são descritas as aplicações científicas e suas características. Na seção 4.2 é apresentado os ambiente computacional escolhidos para realizar os testes. Na seção 4.3 é apresentado as configurações dos testes. Por fim no seção 4.4 é concluído o capítulo.

4.1 Aplicações Científicas

Com o objetivo de entender o desempenho dos recursos providos pela nuvem, foram escolhidas duas aplicações científicas reais. Embora a maioria dos trabalhos encontrados na literatura utiliza *benchmarks* para realizar seus experimentos, aplicações científicas tendem a simular um ambiente real, permitindo avaliar o sistema computacional como um todo. Ambas aplicações são da área da dinâmica dos fluídos, foram escolhidas pela sua relevância no contexto científico e por apresentarem um elevado custo computacional. Nas duas próximas subseções serão detalhadas as características de cada uma delas.

4.1.1 *Ocean-Land-Atmosphere Model*

Modelos atmosféricos foram desenvolvidos para simular o comportamento da atmosfera terrestre e prever o tempo e clima em períodos variados de tempo. Estes modelos realizam cálculos sobre uma grande quantidade de dados, como temperatura, umidade do ar, etc. (VASQUEZ, 2006). Quanto mais precisa for a previsão nos resultados, maior será o volume de dados a ser computado, o que consequentemente demanda maior poder e tempo de processamento.

O OLAM foi desenvolvido por Roni Avissar e Robert Walko na Duke University (WALKO; AVISSAR, 2008). OLAM utiliza uma malha global não estruturada para representação do globo terrestre. Para que esta malha seja criada, inicialmente, o globo é dividido em 20 triângulos formando um icosaedro, ilustrado na Figura 9. Cada um desses triângulos originais é subdividido em outros triângulos, de acordo com uma resolução horizontal pré-estabelecida.

Cada uma das 20 faces do icosaedro é representada por um triângulo equilátero. A divisão de cada uma depende de um parâmetro que representa a resolução da malha, denominado nxp. A Figura 10 ilustra a divisão de uma face do icosaedro em duas, três e quatro partes. O tamanho da resolução implica diretamente na simulação, sendo

Figura 9 – Representação de um icosaedro

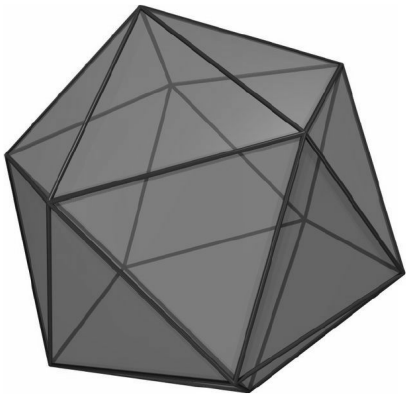


Figura 10 – Exemplo de subdivisão de uma face do icosaedro em 2, 3 e 4 partes

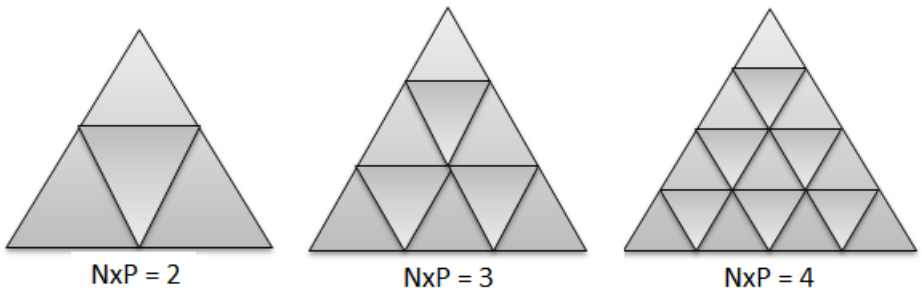


Tabela 3 – Divisões do Globo.

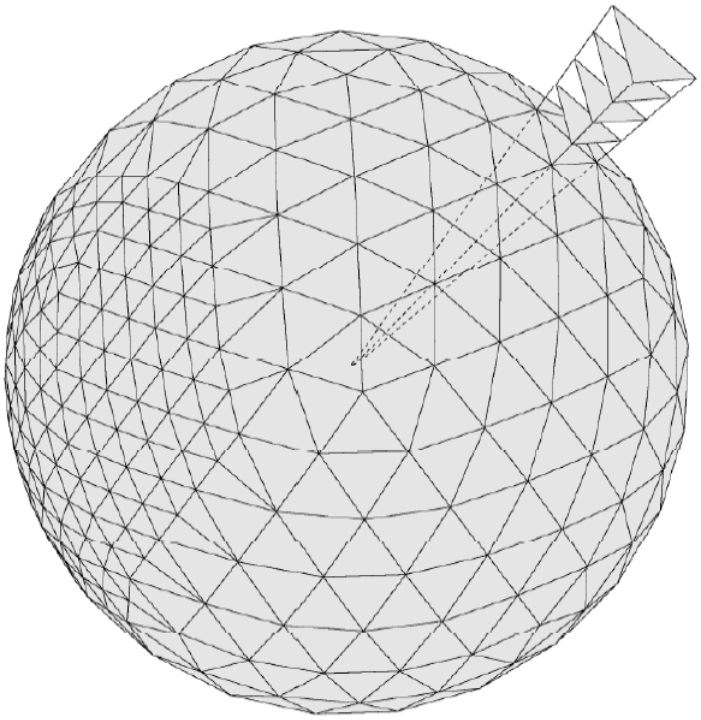
Elemento	**Número de elementos**
Vértices M	$10NxP^2 + 2$
Faces U	$30NxP2^2 + 2$
Triângulos W	$20NxP^2 + 2$

que, quanto maior a resolução da malha, mais preciso será o resultado das previsões, consequentemente mais pontos e mais dados para serem processados. Após esta etapa, é feita a divisão vertical em diversas camadas formando pontos semelhantes a prismas, como podemos observar na Figura 11.

Após a construção da malha, cada prisma corresponde a um elemento da grade. Em cada um desses elementos são armazenadas informações adicionais dos Vértices (M), Faces (U) e pontos centrais dos triângulos (W). A Tabela 3 demonstra como calcular a quantidade de pontos M, U e W a partir do parâmetro NxP.

Uma outra característica do modelo OLAM é a possibilidade de aumentar a resolução em uma determinada região do globo terrestre durante sua execução. Esse aumento, também chamado de refinamento de malha em tempo de execução, ocorre de forma hori-

Figura 11 – Representação de pontos em forma de prisma da malha do OLAM



Referência: (WALKO; AVISSAR, 2008)

zontal utilizando mais pontos discretos em regiões onde uma maior precisão é requerida, como podemos ver na Figura 12. Esta é uma ótima solução para previsões que envolvam iteração de longos períodos de tempo, onde o tempo de execução é grande.

O algoritmo do modelo OLAM envolve diversas etapas. Inicialmente é feito um pré-processamento no qual são lidas as entradas (topologia, vegetação, solos e mar) e feitas as configurações do ambiente para realizar a simulação. O restante do algoritmo é composto por etapas iterativas com base nas informações iniciais. Ao final de cada uma é feita a atualização da passagem do tempo. Caso seja desejado, durante a execução, pode ser feito o refinamento de malha. Após feito o refinamento, o algoritmo continua a execução até concluir todas as etapas (WALKO; AVISSAR, 2008).

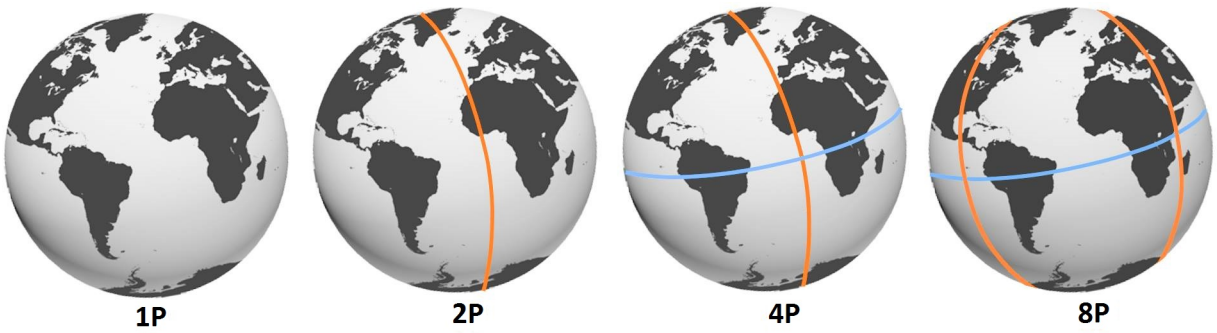
Para realizar os testes de avaliação foi utilizada uma versão simplificada do modelo OLAM, implementada em C (SCHEPKE, 2012), versão esta que possui as principais funções da aplicação original, como o refinamento dinâmico e todas as estruturas de dados e funções necessárias para seu correto funcionamento. A implementação paralela foi feita utilizando a interface de programação MPI com o modelo de programação *Single Program Multiple Data* (SPMD).

Considerando uma grade global, a mesma precisa ser dividida entre os processa-

Figura 12 – Refinamento local em uma região do globo

Referência: (WALKO; AVISSAR, 2008)

Figura 13 – Exemplo de particionamento em blocos do OLAM



dores, a fim de realizar o paralelismo dos dados. A Figura 13 ilustra a divisão do globo terrestre em duas, quatro e oito partes. Cada processo fica responsável por operar uma dessas partes. A comunicação entre os processos ocorre em cada iteração, apenas entre as regiões vizinhas.

A Tabela 4 faz uma representação do número de pontos que cada processo deverá comunicar. O cálculo do número de pontos de cada comunicação pode ser feito através do parâmetro NxP. Pegando como exemplo 2 processos, nxp igual a 50, teremos 50 + 50

Tabela 4 – Pontos de Comunicação do MLB.

	1 Processo	**2 Processos**	**4 Processos**	**8 Processos**
Processos Vizinhos	0	1	2	2
Pontos Comunicação	0	$n + n * \sqrt{3}$	$(n + n * \sqrt{3})/2$	$(n + n * \sqrt{3})/4$

$* \sqrt{3}$ com um total de aproximadamente 136 pontos de borda que deverão ser enviados aos processos vizinhos em cada iteração.

Devido a borda ser não estruturada, o OLAM empacota em um *buffer* o que for necessário para envio. O empacotamento utiliza tabelas auxiliares preenchidas pelas rotinas de divisão de domínio. Após o empacotamento, o *buffer* com os dados é enviado utilizando mensagens MPI não bloqueantes. O processo que recebe este vetor realiza o procedimento inverso de desempacotar os dados e colocá-los na grade. Este processo de empacotar, enviar, receber e desempacotar ocorre com auxílio de diversas tabelas, a fim de assegurar que os dados sejam copiados e gravados na grade em seus lugares corretos.

4.1.2 Método de *Lattice Boltzmann*

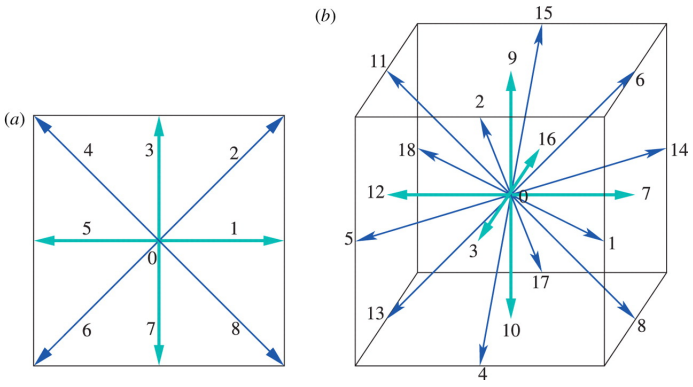
A Dinâmica dos Fluídos Computacionais (DFC) é uma importante área de pesquisa da computação científica que estuda o movimento de líquidos e gases através de simulações numéricas. Os trabalhos desenvolvidos nessa área possibilitam a simulação de diversos sistemas físicos, como furacões, maremotos, aerodinâmica, aeroacústica e gerenciamento térmico (SCHEPKE, 2007).

O MLB é um método numérico iterativo para simulação de propriedades físicas em fluídos, onde o espaço, tempo e velocidade são discretos (SUCCI, 2001). O fluído deste modelo é representado por meio de uma estrutura em forma de malha, também chamada de *latice* ou reticulado, composta por um conjunto de pontos ou células. Cada ponto representa as propriedades físicas de moléculas com valores reais para a propagação das informações pelo reticulado. A atualização desses pontos ocorre simultaneamente em intervalos de tempos discretos (CHEN; DOOLEN, 1998).

O número de direções que cada partícula pode se movimentar depende do modelo de reticulado. Na literatura podemos encontrar diversos modelos, neste trabalho usaremos o modelo tridimensional pois é o modelo que mais se aproxima da realidade. Como podemos observar na Figura 14 no modelo tridimensional, também chamado de D3Q19, cada partícula tem a possibilidade de se movimentar para 19 direções.

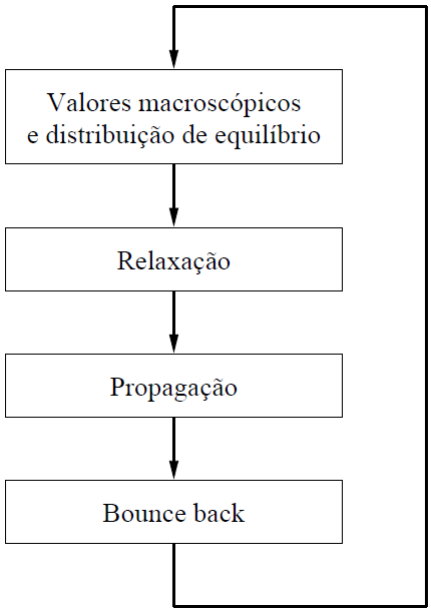
Após determinar as condições iniciais para todos os pontos do reticulado (velocidade, densidade, iterações, etc.), o MLB pode ser representado por um laço iterativo conforme a Figura 15. Em cada iteração os valores da função de equilíbrio são calculados

Figura 14 – Modelo de reticulado D3Q19



Referência: (ROLOFF et al., 2012)

Figura 15 – Estrutura do Algoritmo MLB

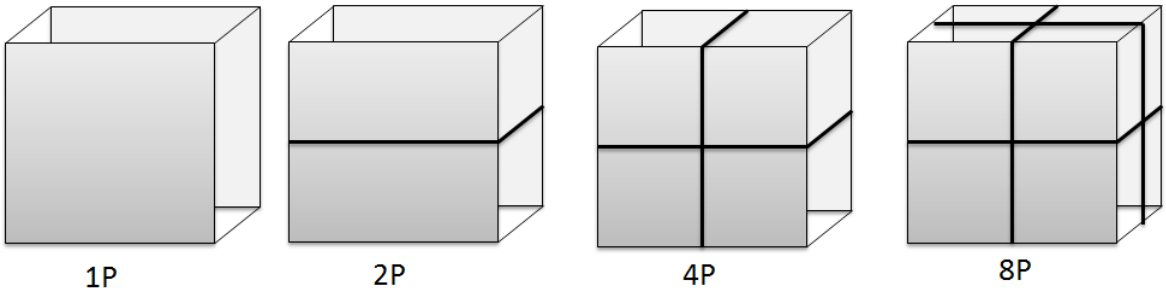


Referência: (SCHEPKE, 2007)

e passados para o cálculo da função de relaxação. A função relaxação utiliza o valor de equilíbrio para calcular a função de distribuição para cada ponto. Por fim, é preciso modelar o comportamento das partículas para satisfazer as condições de contorno, o que é feito pela função *bounceback*.

O MLB é um aplicação que exige uma grande carga de processamento. Uma de suas características é a possibilidade de aumentar seu desempenho através da paralelização de suas operações (SCHEPKE, 2007) (SUCCI, 2001). Para realizar os testes foi utilizada a versão paralela do método MLB, para fluxos tridimencionais, implementada por Schepke

Figura 16 – Exemplo de particionamento em blocos do MLB



(2007).

A implementação das funções do MLB foram escritas usando a linguagem de programação C. Para armazenar e manipular as informações foram usadas três estruturas de dados. Uma para representar as propriedades físicas, uma para a estrutura do reticulado e outra para representar os parâmetros referentes a execução paralela. Essas estruturas contém diversas informações utilizadas para iniciar e computar as variáveis do reticulado.

A versão paralela do método foi feita utilizando a interface de programação MPI, baseada no modelo de programação SPMD. Este modelo faz com que o mesmo código atue em cada processador sobre uma parcela distinta de dados. O particionamento dos dados foi feito dividindo a estrutura do reticulado entre os processadores disponíveis. Assim cada parte ou sub-reticulado é controlada por um único processo e realiza seus cálculos de forma independente. A Figura 16 ilustra a divisão em 2, 4 e 8 partes de um reticulado tridimensional, onde cada uma das partes deverá ser atribuída para um processo independente.

Após cada iteração é necessário que haja trocas de informações entre as regiões vizinhas, devido a propagação das informações do fluído. Como relatado anteriormente o modelo tridimensional possui 18 direções diferentes de deslocamento além da posição estática. Essas 18 direções podem ser agrupadas de maneira que sejam formadas 6 faces ortogonais. Cada uma delas é composta por 5 direções de deslocamento diferentes: uma direção ortogonal e 4 diagonais. Assim a comunicação é feita entre as faces e as diagonais dos sub-reticulados.

A Tabela 5 faz uma representação do número de pontos de comunicação que cada processo deverá comunicar com seu vizinho. Pegando como exemplo a divisão de dois processos com um reticulado de tamanho 128 x 128 x 128. Neste caso teríamos a comunicação apenas entre uma das faces dos sub-reticulados, com 128 x 128 = 162384 pontos. Agora, pegando como exemplo quatro processos, teríamos cada processo se comunicando com dois processos vizinhos através de suas faces e com um terceiro através de sua diagonal.

Tabela 5 – Pontos de comunicação OLAM.

	1 Processo	**2 Processos**	**4 Processos**	**8 Processos**
Processos Vizinhos	0	1	2 + 1	3 + 3
Pontos Comunicação	0	n * m	(n * m)/2 + n/2	(n * m)/4 + 3(n/2)

Quando é feita a divisão do reticulado são criados um conjunto adicional de pontos para armazenar valores temporários utilizados na comunicação. Para realizar a comunicação inicialmente é criada uma estrutura de comunicação cartesiana, onde são definidos o número de dimensões e o número de partes em cada dimensão. Após realizar o armazenamento temporário é feita a comunicação entre as regiões vizinhas. A troca de informações em si é feita utilizando mecanismo de envio assíncrono não bloqueante.

4.2 Ambiente de Teste

Os experimentos foram executados em três ambientes computacionais distintos disponibilizada pelo grupo de pesquisa Laboratório de Pesquisas Avançadas para Computação em Nuvem da SETREM (LARCC) (Três de Maio/RS). Um *cluster* nativo e duas nuvens IaaS privadas gerenciadas pelas ferramentas de administração de nuvem OpenNebula e OpenStack, respectivamente. Ambos ambientes virtuais foram criados com o *hipervisor* KVM. Cada ambiente é formado por máquinas idênticas, compostas por um processador Intel *Core* i5 650 com 3.20GHz, memória RAM de 4 GB DDR3(1333MHz), disco de 500GB operando em *sata* II. O SO utilizado em todas as máquinas foi o Ubuntu *Server* 14.4.

4.3 Metricas de Desempenho

As métricas de desempenho utilizadas na avaliação das aplicações cientificas para cada conjunto de processos foram as seguintes:

Tempo Execução: O Tempo total de execução foi obtido através de uma média de 20 execuções onde foram descartadas os dois melhores e os dois piores tempos.

***Speed-Up*:** É uma métrica utilizada para expressar quantas vezes a execução do programa paralelo ficou mais rápido que a versão sequencial. O cálculo do *speed-up* é feito pela razão entre o tempo de execução sequencial e a versão paralela. A Equação 4.1 ilustra essa razão, onde $T(1)$ indica o melhor tempo de processamento da versão sequencial, e T(p) o tempo de processamento da versão paralela.

$$S(p) = \frac{T(1)}{T(p)} \tag{4.1}$$

Cada aplicação tem um número de unidades de processamento ideais para obtenção do melhor desempenho. Ou seja, nem sempre a adição de unidades de processamento aumentará o desempenho. Caso $S(p) > 1$ a versão paralela reduziu o tempo de execução, caso $S(p) < 1$ a versão paralela ficou mais lenta que a sequencial.

Eficiência: A eficiência é uma medida que mostra como foi a taxa de utilização média das unidades de processamento utilizadas. O cálculo da eficiência é feito pela razão entre o *speed-up* e as unidades de processamento utilizadas. A Equação 4.2 ilustra essa razão, onde $S(p)$ é o *speed-up* e p é o número de unidades de processamento.

$$E(p) = \frac{S(p)}{p} \tag{4.2}$$

A eficiência ideal seria quando cada unidade de processamento ficasse ativa 100% durante todo o tempo de execução. No entanto, geralmente essas unidades tem que aguardar por resultados de unidades vizinhas, isso faz com que reduza a taxa de utilização e consequentemente a eficiência.

4.4 Conclusão Capítulo

Neste capítulo, apresentamos a metodologia utilizada neste trabalho. Inicialmente apresentamos as duas aplicações científicas reais da área da dinâmica dos fluídos, seu funcionamento, características e motivo da escolha. Após apresentamos a configuração dos ambientes utilizados, seguido pelas métricas de avaliação.

5 Resultados dos Experimentos

Este capítulo apresenta a análise e comparação dos resultados obtidos através das execuções do MLB e OLAM nas IaaS privadas e ambiente nativo. Na primeira fase de testes, foi feita uma comparação entre o ambiente nativo e uma nuvem privada gerenciada pela ferramenta OpenStack. Na subseção 5.1.1 foi avaliado o MLB e na subseção 5.1.2 o OLAM. A segunda fase de testes, descrita na seção 5.2, foi feita uma breve comparação de desempenho das ferramentas OpenNebula e OpenStack. A análise foi feita utilizando o MLB. Por fim na seção 5.3 foram feitas as considerações finais do capítulo.

5.1 Comparação da IaaS OpenStack com Cluster Nativo

A primeira etapa de testes teve como objetivo comparar o desempenho de uma nuvem privada com o ambiente nativo, verificando o impacto que aplicações científicas, podem causar na utilização dos recursos virtualizados. A nuvem privada escolhida foi a gerenciada pela ferramenta OpenStack com *hipervisor* KVM. As aplicações utilizadas foram o OLAM e o MLB previamente descritas na seção 4.1.

Os ambientes de teste possuem as mesmas configurações de *hardware*. Neste primeiro caso ambos ambientes foram formados por 2 nós, cada um composto por 2 núcleos e 4 *threads*, totalizando 8 unidades de processamento. Foram feitas 20 execuções para cada caso de teste, sendo que as 5 melhores e os 5 piores tempos foram descartados, com isso o desfio padrão foi inferior a 1%. Os resultados apresentados a seguir correspondem ao tempo de execução, *speed-up* e eficiência das aplicações, sobre diferentes tamanhos de problemas e número de processos.

5.1.1 Avaliação Metodo Lattice Boltzmann (MLB)

A avaliação do MLB foi feita utilizando os seguintes tamanhos do reticulado: (A) 64 x 64 x 64, (B) 128 x 64 x 64, (C) 128 x 128 x 64 e (D) 128 x 128 x 128. A Tabela 6 ilustra os tamanhos utilizados com o número de pontos que deverão ser computados. É importante lembrar que cada ponto possui estruturas adicionais com informações sobre as propriedades físicas utilizadas para calcular o fluxo do fluído.

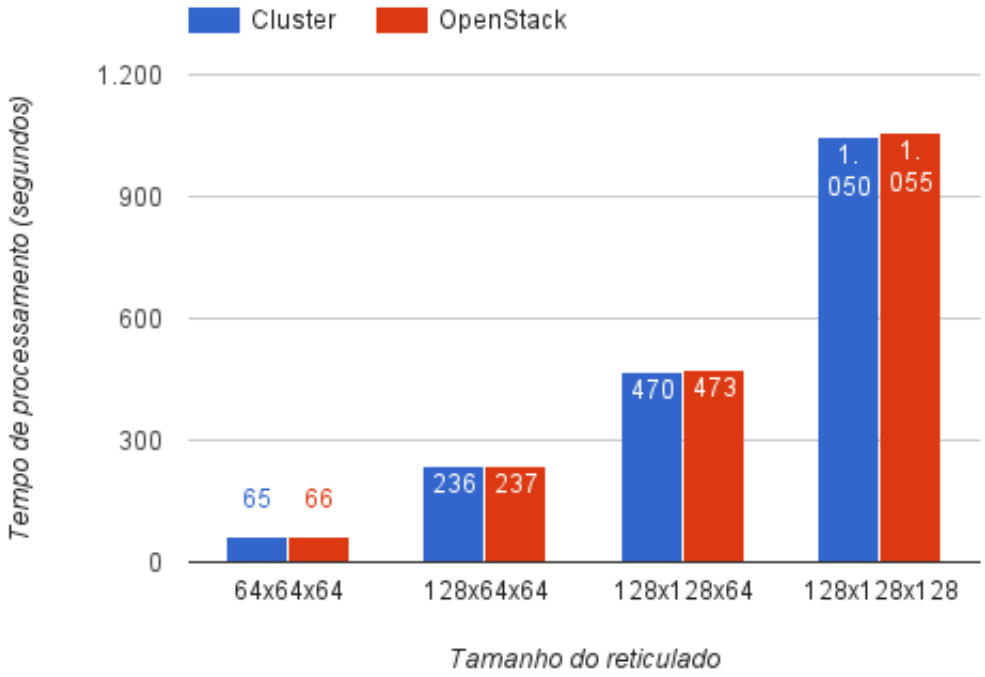
5.1.1.1 Avaliação Sequencial do MLB

Inicialmente, foi feita uma avaliação sequencial do MLB considerando tamanhos variados do reticulado. Como vimos na Tabela 6 quanto maior for o tamanho do reticulado, maior será quantidade de pontos a serem computados. A Figura 18 ilustra o tempo

Tabela 6 – Variação do Reticulado do MLB

Caso	Dimensões do reticulado	Número de pontos
A	64 x 64 x 64	262144
B	128 x 64 x 64	524288
C	128 x 128 x 64	1048576
D	128 x 128 x 128	2097152

Figura 17 – Tempo de Execução do MLB variando o tamanho do reticulado



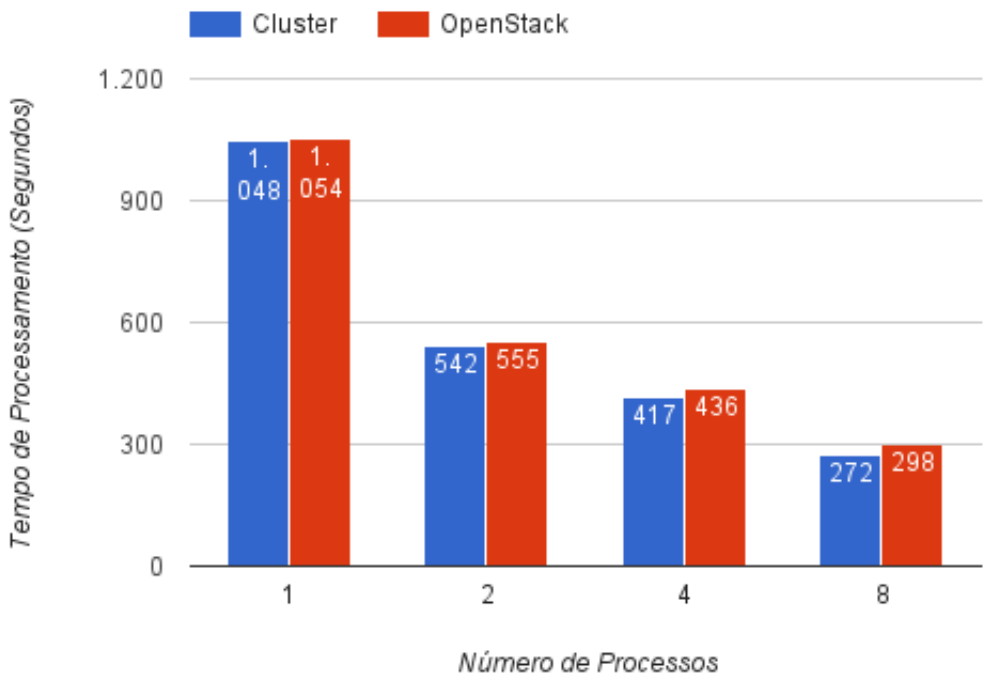
de execução para cada caso de teste. De acordo com o gráfico, podemos perceber que a diferença de desempenho, do MLB sequencial, entre a nuvem gerenciada pelo OpenStack e o ambiente nativo foi muito pequena, com apenas 1% para o caso A (64 x 64 x 64) de menor tamanho.

Para ficar mais claro a diferença de cada tamanho do reticulado, fizemos uma análise teórica do consumo de memória. Para tal, utilizamos o comando *top* disponibilizado pelo Linux para monitorar o desempenho do sistema. A Tabela 7 descreve cada caso de teste (reticulado variado) com seu consumo de memória. Com isso, fica claro que mesmo com tamanhos maiores do problema, o desempenho da nuvem OpenStack com a versão sequencial do MLB, não apresenta sobrecarga significativa de tempo.

Tabela 7 – Consumo de Memória aproximado do MLB

Reticulado	64 x 64 x 64	128 x 64 x 64	128 x 128 x 64	128 x 128 x 128
Memória	96 MB	192 MB	372 MB	720 MB

Figura 18 – Tempo de Execução do MLB paralelo com reticulado 128 x 128 x 128



5.1.1.2 Avaliação Paralela do MLB

Para realizar a avaliação da versão paralela do MLB utilizamos a infraestrutura completa de experimentação, formada por 2 nós, cada um composto por 2 núcleos e 4 *threads*, totalizando 8 unidades de processamento. A análise foi feita variando o número de processos entre 2, 4 e 8. Desta forma cada processo recebeu uma parte resultante do particionamento do reticulado, efetuado pelo MLB paralelo.

A Figura 18 ilustra o tempo de execução do MLB paralelo para um reticulado de tamanho 128 x 128 x 128. A escolha desse tamanho se deve ao alto custo computacional. Podemos perceber o ganho de desempenho com o aumento no número de processos. Comparando a nuvem OpenStack com ambiente nativo podemos notar uma diferença de desempenho gradativa entre os ambientes. Com dois processos o OpenStack foi 2% mais lento que o ambiente nativo. Com quatro processos a diferença aumenta para 4% e com oito essa diferença de tempo chega a ser 8% maior na nuvem OpenStack.

Tabela 8 – Speed-Up e Eficiência do MLB paralelo com reticulado 128 x 128 x 128

Processos	**Eficiência**		**Speed-Up**	
	OpenStack	Nativo	OpenStack	Nativo
2	94,9%	96,7 %	1,8	1,9
4	60,4%	62,8 %	2,4	2,5
8	44,2 %	48,1 %	3,5	3,8

Tabela 9 – Variação da Resolução da Malha do OLAM.

Caso	**Resolução da Malha**	**Triângulos**	**Faces**	**Vértices**
A	202,0 Km	12502	18752	6252
B	144,2 Km	24502	36752	12252
C	101,0 Km	50002	75002	25002
D	77,69 Km	84502	126752	42252

A Tabela 8 reproduz o *speed-up* e a eficiência do MLB (128 x 128 x 128). Como podemos ver na tabela com o aumento do número de processos, consequentemente ocorreu um aumento nos *speed-up* obtidos. A questão da baixa eficiência com mais processos esta relacionada ao tempo de comunicação necessário para a troca de mensagens e sincronização entre os mesmos.

Com esses resultados podemos perceber que o aumento do tempo na nuvem está diretamente ligado ao *overhead* na comunicação causado pela virtualização. E como vimos na subseção 4.1.2, o MLB é uma típica aplicação que exige constante comunicação, onde a cada iteração é necessário que haja trocas de informações entre as regiões vizinhas, devido a propagação das informações do fluído.

5.1.2 Avaliação do Ocean-Land-Atmosphere Model (OLAM)

Para os casos de testes no modelo OLAM foram simuladas 60 horas da atmosfera terrestre, considerando 28 níveis da camada vertical, com resoluções horizontais variadas de: (A) 202,0 km, (B) 144,2 Km, (C) 101,0 Km, (D) 77,69 Km e (E) 67,33 km. O cálculo da resolução da malha é feito dividindo 5050 pelo parâmetro NxP. A Tabela 9 faz uma análise teórica do número total de pontos. Lembrando que cada ponto possui estruturas relacionadas a Vértices (M), Faces (U) e pontos centrais dos triângulos (W).

Como podemos observar na Tabela 9 quanto menor a resolução horizontal maior será o numero de pontos que deverão ser computados. Para realizar estes experimentos optamos por não utilizar o refinamento de malha em tempo de execução, sendo que cada iteração do modelo representa 60 segundos do tempo real.

Figura 19 – Tempo de execução do OLAM variando a resolução horizontal da malha

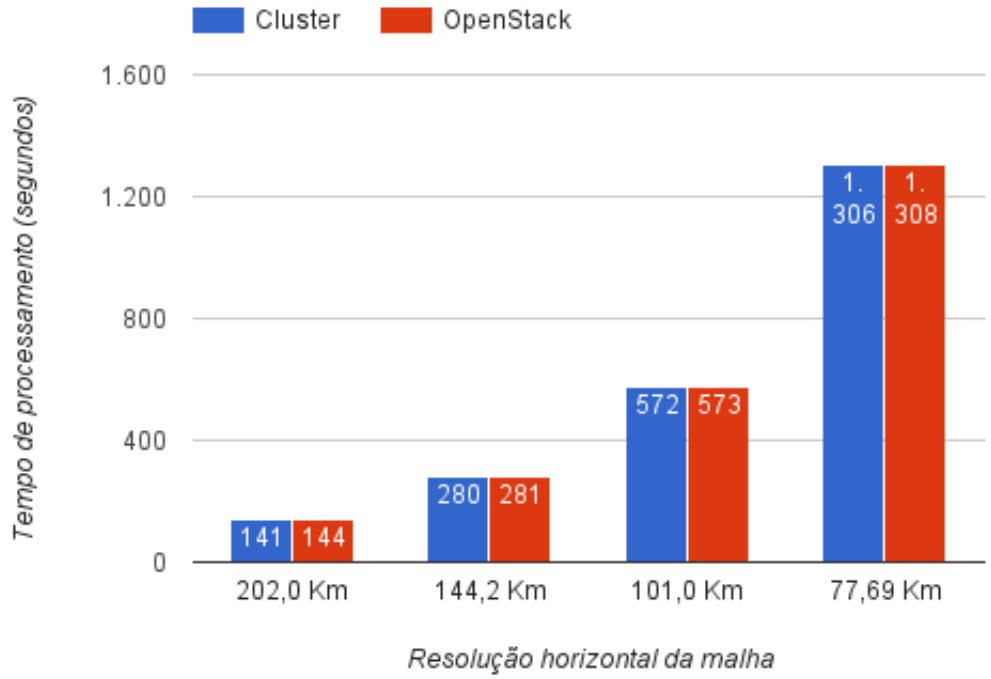


Tabela 10 – Consumo de Memória aproximado do OLAM

Resolução	202,0 km	144,2 km	101,0 km	67,3 km
Memória	168 MB	324 MB	660 MB	1440 MB

5.1.2.1 Avaliação Sequencial do OLAM

A avaliação sequencial para o OLAM foi feita considerando diferentes resoluções da malha horizontal. De acordo com a Figura 19, podemos perceber que a diferença de desempenho entre a nuvem gerenciada pelo OpenStack e o ambiente nativo foi muito pequena, sendo menor que 2% para malha com resolução de 202,0 Km. A Tabela 10 descreve ainda o consumo de memória aproximado para cada variação da malha horizontal.

5.1.2.2 Avaliação Paralela do OLAM

A avaliação paralela do OLAM foi feita utilizando uma resolução de 101,0 Km, variando o número de processos entre 2, 4 e 8. A escolha de 101,0 Km de resolução foi feita por ser um tamanho apropriado a dimensões globais. Após a divisão, cada processo MPI ficou responsável por operar uma parte da malha terrestre. A Figura 20 ilustra uma redução do tempo com o aumento no número de processos e uma diferença gradativa entre o ambiente nativo e a nuvem OpenStack. Com dois processos a nuvem OpenStack possui

Figura 20 – Tempo de execução do OLAM com resolução de 101,0 Km

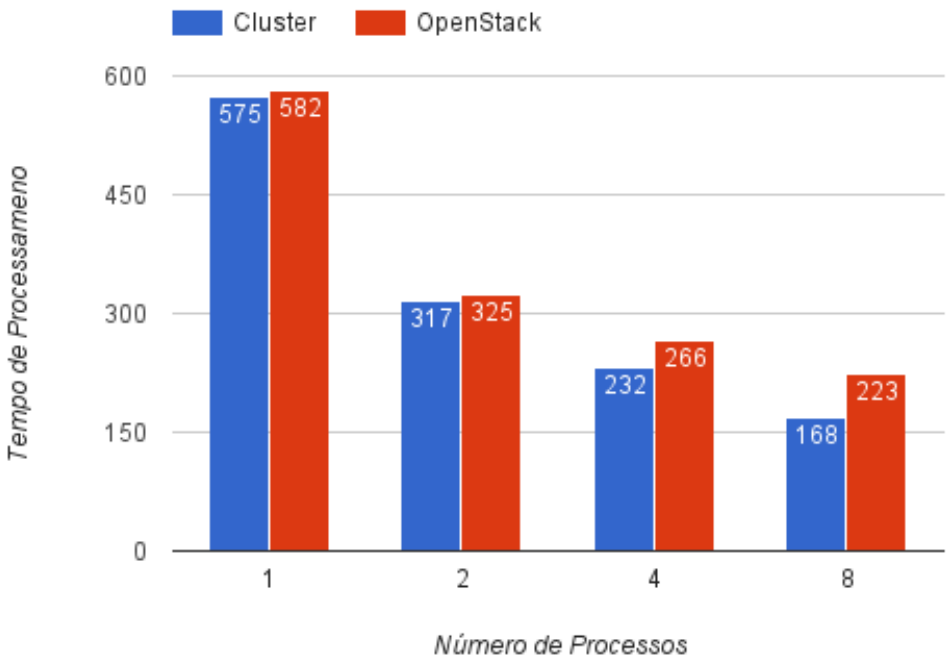


Tabela 11 – Speed-Up e Eficiência do OLAM paralalelo com resolução de 101,0 Km

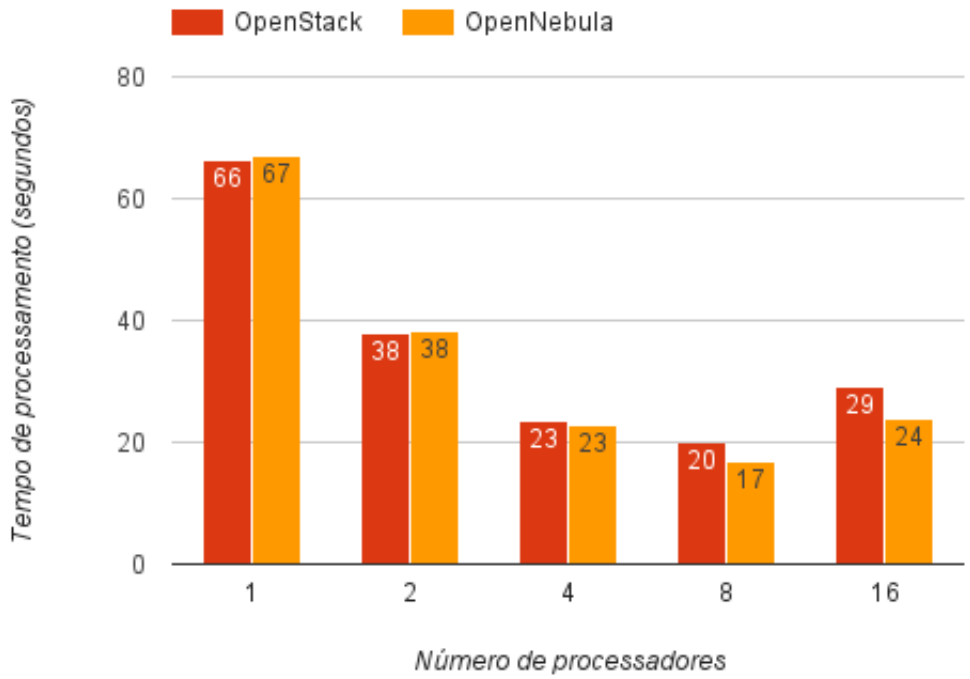
Processos	**Eficiência**		**Speed-Up**	
	OpenStack	Nativo	OpenStack	Nativo
2	88,5%	91,8 %	1,7	1,8
4	53,9 %	62,7 %	2,1	2,5
8	32,1 %	43,3 %	2,5	3,4

uma pequena de 2%, com 4 processos essa diferença aumenta para 12%, chegando a 24% com 8 processos.

A Tabela 11 apresenta a eficiência e o *speed-up* do OLAM com uma resolução de 101,0 Km. De modo semelhante ao analisado na subseção 5.1.2 com o MLB, podemos notar que com o aumento do número de processos, há um aumento nos *speed-ups* obtidos e uma redução na eficiência devido a comunicação e sincronização dos processos. Podemos perceber ainda a diferença entre os ambientes, com o ambiente nativo chegando a ser 11% mais eficiente que a nuvem com OpenStack.

Do mesmo modo que no MLB, os resultados do OLAM para o ambiente de nuvem tiveram um decréscimo de desempenho com o aumento no número de processos. Se comparado ambas aplicações, podemos perceber ainda, que com o OLAM a diferença de desempenho maior que com MLB.

Figura 21 – Comparação de desempenho OpenNebula x OpenStack com o MLB



5.2 Comparação entre OpenStack e OpenNebula

Esta segunda fase de testes teve por objetivo avaliar e comparar o impacto que as ferramentas de gerenciamento de nuvem OpenStack e OpenNebula causam no desempenho de recursos virtuais. Para realizar o estudo de caso, foi utilizado duas nuvens isoladas, implantadas sobre o mesmo virtualizador (KVM) e *hardware* físico. Cada nuvem é formada por 3 nós, cada um composto por 2 núcleos e 4 *threads*, totalizando 12 unidades de processamento por nuvem.

Para essa análise, foi escolhido o MLB com um reticulado de tamanho 64 x 64 x 64. A execução da versão paralela em MPI, foi feita variando o número de processos: 2, 4, 8 e 16. Como podemos observar na Figura 21 o desempenho de ambas ferramentas foram semelhantes. Podemos perceber que com 1, 2 e 4 a diferença é mínima sendo inferior a 1%. Com o aumento no número de processos, consequentemente um maior número de comunicação, a nuvem OpenNebula passou a ter vantagem sobre o OpenStack. Com 8 processos, o OpenStack foi 14%, chegando a uma diferença de 18% com 16 processos.

Como esses resultados podemos perceber que existem diferenças de desempenho entre as ferramentas. Neste caso, avaliando a ferramenta OpenNebula e OpenStack, podemos perceber que a ferramenta OpenStack apresentou um decréscimo de desempenho

de até 18% com o aumento da comunicação entre os processos.

5.3 Conclusão do Capítulo

Este capítulo teve por objetivo apresentar os resultados propostos neste trabalho. Em um primeiro momento foi comparado o desempenho do ambiente nativo com a nuvem gerenciada pela ferramenta OpenStack. A segunda fase de testes, avaliou o desempenho das ferramentas OpenNebula e OpenStack. Como podemos perceber, ambientes virtuais impactam no desempenho de aplicações científicas, principalmente em aplicações com grande comunicação entre processos. Os resultados também mostraram que a ferramentas podem influenciar no desempenho.

6 Conclusão e Trabalhos Futuros

A computação em nuvem vem sendo alvo de diversas linhas de pesquisas, sua flexibilidade e elasticidade na alocação de recursos vem atraindo cada vez mais o interesse de pesquisadores, com inúmeras organizações investindo nesse novo paradigma computacional. Este trabalho apresentou uma análise de desempenho de aplicações científicas reais em ambientes de computação em nuvem privada gerenciadas pelas ferramentas de administração de nuvem OpenNebula e OpenStack.

A análise foi dividida em duas etapas. Na primeira parte dos testes foi avaliado o desempenho do OLAM e do MLB em uma nuvem privada gerida pela ferramenta OpenStack. Os resultados mostraram que em ambas aplicações as maiores quedas de desempenho foram nas execuções que necessitaram de uma maior comunicação entre os processos. Esse comportamento é devido ao uso da virtualização de recursos, que acaba causando um perda de desempenho em relação ao ambiente nativo.

O segundo caso de teste teve como objetivo avaliar e comparar o impacto que as ferramentas de gerenciamento de nuvem OpenStack e OpenNebula causam no desempenho de recursos virtuais. Para realizar o estudo foi utilizado o MLB. Os resultados mostraram que as ferramentas de administração de nuvem podem impactar no desempenho, com uma vantagem para a ferramenta OpenNebula. Considerando que na avaliação foi sobre uma mesma infraestrutura física e mesmo virtualizador.

Como vimos, a computação em nuvem pode apresentar perdas de desempenho. No entanto, deve se levar em conta as inúmeras vantagens que este ambiente pode prover. Com a computação em nuvem podemos construir ambientes virtuais diminuindo a ociosidade de processamento, otimizando a utilização dos recursos, gerando mais economia. Além de poder proporcionar a cientistas resursos instantaneamente e simultaneamente de acordo com a sua demanda.

Assim podemos concluir que os resultados desta pesquisa contribuíram com o tema. Os próximos objetivos dessa pesquisa são: Avaliar outras classes de aplicações científicas; avaliar um número maior de nós e instâncias virtuais; avaliar outros *hipervisores* e outras ferramentas de administração de nuvem.

Referências

AL-MUKHTAR, M. M. A.; MARDAN, A. A. A. Performance evaluation of the cloudstack private cloud. *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, v. 2, n. 6, p. 403–416, 2014. Citado 2 vezes nas páginas 38 e 39.

ALVES, M. M.; DRUMMOND, L. M. de A. Análise de desempenho de um simulador de reservatórios de petróleo em um ambiente de computação em nuvem. *WSCAD 2014 - XV Simposio em Sistemas Computacionais de Alto Desempenho*, 2014. Citado 2 vezes nas páginas 38 e 39.

AMAZON: Site. 2015. Disponível em: <http://aws.amazon.com/>. Acesso em: 04 jan. 2015. Citado na página 21.

APPENGINE: Site. 2015. Disponível em: <https://appengine.google.com/>. Acesso em: 07 jan. 2015. Citado na página 22.

AZURE: Site. 2015. Disponível em: <http://www.windowsazure.com>. Acesso em: 07 jan. 2015. Citado na página 22.

BESERRA, D. et al. Performance evaluation of hypervisors for hpc applications. *IEEE International Conference on Systems, Man, and Cybernetics*, 2015. Citado na página 38.

BUYYA, R. High performance cluster computing. *New Jersey: Frentice*, 1999. Citado 2 vezes nas páginas 33 e 34.

BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, Elsevier, v. 25, n. 6, p. 599–616, 2009. Citado na página 17.

CARISSIMI, A. Virtualização: da teoria a soluções. *Minicursos do Simpósio Brasileiro de Redes de Computadores–SBRC*, v. 2008, p. 173–207, 2008. Citado 2 vezes nas páginas 25 e 26.

CARISSIMI, A. Desmistificando a computação em nuvem. 2015. Citado na página 25.

CHAKTHRANONT, N. et al. Exploring the performance impact of virtualization on an hpc cloud. In: IEEE. *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*. [S.l.], 2014. p. 426–432. Citado 2 vezes nas páginas 38 e 39.

CHAPMAN, B.; JOST, G.; PAS, R. v. d. *Using OpenMP: Portable Shared Memory Parallel Programming*. [S.l.]: The MIT Press, 2007. ISBN 0262533022, 9780262533027. Citado na página 36.

CHEN, S.; DOOLEN, G. D. Lattice boltzmann method for fluid flows. *Annual review of fluid mechanics*, Annual Reviews 4139 El Camino Way, PO Box 10139, Palo Alto, CA 94303-0139, USA, v. 30, n. 1, p. 329–364, 1998. Citado na página 45.

CLOUDSTACK: Site. 2015. Disponível em: <http://cloudstack.apache.org/>. Acesso em: 12 Setembro. 2015. Citado na página 22.

EUCALYPTUS: Site. 2015. Disponível em: <https://www.eucalyptus.com/>. Acesso em: 07 jan. 2015. Citado na página 22.

FOSTER, I. *Designing and building parallel programs*. [S.l.]: Addison Wesley Publishing Company, 1995. Citado na página 35.

FOSTER, I.; KESSELMAN, C. *The Grid 2: Blueprint for a new computing infrastructure*. [S.l.]: Elsevier, 2003. Citado na página 34.

GLOBUS: Site. 2015. Disponível em: <https://www.globus.org/>. Acesso em: 08 nov. 2015. Citado na página 34.

GOGRID: Site. 2015. Disponível em: <http://www.gogrid.com/>. Acesso em: 07 jan. 2015. Citado na página 21.

GOOGLE: Site. 2015. Disponível em: <http://www.google.com/>. Acesso em: 07 jan. 2015. Citado na página 22.

GROPP, W. et al. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, Elsevier, v. 22, n. 6, p. 789–828, 1996. Citado na página 37.

KVM: Site. 2015. Disponível em: <http://www.linux-kvm.org/page/Main_Page>. Acesso em: 05 jan. 2015. Citado na página 27.

LAUREANO, M. *Máquinas virtuais e emuladores: conceitos, técnicas e aplicações*. [S.l.]: Novatec Editora, 2006. Citado na página 26.

MARINOS, A.; BRISCOE, G. Community cloud computing. In: *Cloud Computing*. [S.l.]: Springer, 2009. p. 472–484. Citado 2 vezes nas páginas 22 e 23.

MARON, C. A. F. et al. Em Direcao a Comparacao do Desempenho das Aplicacoes Paralelas nas Ferramentas OpenStack e OpenNebula. In: *15th Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul (ERAD/RS)*. Gramado, RS, Brazil: Sociedade Brasileira de Computacao, 2015. Citado 3 vezes nas páginas 17, 38 e 39.

MELL, P. M.; GRANCE, T. *SP 800-145. The NIST Definition of Cloud Computing*. Gaithersburg, MD, United States, 2011. Citado 3 vezes nas páginas 17, 19 e 23.

MPI-FORUM: Site. 2015. Disponível em: <http://www.mpi-forum.org/>. Acesso em: 14 jun. 2015. Citado na página 37.

MPICH: Site. 2015. Disponível em: <https://www.mpich.org/>. Acesso em: 08 nov. 2015. Citado na página 37.

OPEN-MPI: Site. 2015. Disponível em: <http://www.open-mpi.org/>. Acesso em: 08 nov. 2015. Citado na página 37.

OPENNEBULA: Site. 2015. Disponível em: <http://opennebula.org/>. Acesso em: 07 jan. 2015. Citado 3 vezes nas páginas 22, 29 e 30.

OPENSTACK: Site. 2015. Disponível em: <http://www.openstack.org/>. Acesso em: 07 jan. 2015. Citado 3 vezes nas páginas 22, 28 e 29.

OSTERMANN, S. et al. A performance analysis of ec2 cloud computing services for scientific computing. In: *Cloud computing*. [S.l.]: Springer, 2010. p. 115–131. Citado na página 37.

RACKSPACE: Site. 2015. Disponível em: <http://www.rackspace.com/pt/>. Acesso em: 07 jan. 2015. Citado na página 21.

RIGHI, R. d. R. Elasticidade em cloud computing: conceito, estado da arte e novos desafios. *Revista Brasileira de Computação Aplicada*, v. 5, n. 2, p. 2–17, 2013. Citado na página 21.

ROLOFF, E. et al. High performance computing in the cloud: Deployment, performance and cost efficiency. In: IEEE. *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. [S.l.], 2012. p. 371–378. Citado na página 46.

ROSE, C. D.; NAVAUX, P. Fundamentos de processamento de alto desempenho. p. 3–29, 2002. Citado na página 34.

SALESFORCE: Site. 2015. Disponível em: <http://www.salesforce.com/>. Acesso em: 31 ago. 2015. Citado na página 22.

SCHEPKE, C. Distribuiçao de dados para implementaçoes paralelas do método de lattice boltzmann. *Instituto de Informática, UFRGS, Porto Alegre*, 2007. Citado 3 vezes nas páginas 45, 46 e 47.

SCHEPKE, C. Exploiting multiple levels of parallelism and online refinement of unstructured meshes in atmospheric model application. 2012. Citado na página 43.

SMITH, J. E.; NAIR, R. The architecture of virtual machines. *Computer*, IEEE, v. 38, n. 5, p. 32–38, 2005. Citado na página 26.

SOUSA, E. de et al. Evaluating eucalyptus virtual machine instance types: A study considering distinct workload demand. In: *Third International Conference on Cloud Computing, GRIDs, and Virtualization*. [S.l.: s.n.], 2012. p. 130–135. Citado 2 vezes nas páginas 38 e 39.

SUBRAMANIAN, V. et al. Rapid 3d seismic source inversion using windows azure and amazon ec2. In: IEEE. *Services (SERVICES), 2011 IEEE World Congress on*. [S.l.], 2011. p. 602–606. Citado na página 17.

SUCCI, S. *The lattice Boltzmann equation: for fluid dynamics and beyond*. [S.l.]: Oxford university press, 2001. Citado 2 vezes nas páginas 45 e 46.

TAURION, C. *Cloud Computing-Computação em Nuvem*. [S.l.]: Brasport, 2009. Citado 2 vezes nas páginas 18 e 19.

TOP500: Site. 2015. Disponível em: <http://www.top500.org/>. Acesso em: 08 nov. 2015. Citado na página 34.

TUDORAN, R. et al. A performance evaluation of azure and nimbus clouds for scientific applications. In: ACM. *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*. [S.l.], 2012. p. 4. Citado na página 38.

VAQUERO, L. M. et al. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 39, n. 1, p. 50–55, dez. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1496091.1496100>. Citado na página 19.

VASQUEZ, T. *Weather forecasting red book*. Garland, TX: Weather Graphics Technologies, 2006. Citado na página 41.

VECCHIOLA, C.; PANDEY, S.; BUYYA, R. High-performance cloud computing: A view of scientific applications. In: IEEE. *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*. [S.l.], 2009. p. 4–16. Citado 2 vezes nas páginas 24 e 37.

VERAS, M.; TOZER, R. *Cloud Computing: nova arquitetura da TI*. [S.l.]: Brasport, 2012. Citado 2 vezes nas páginas 20 e 25.

VMWARE: Site. 2015. Disponível em: <http://www.vmware.com/>. Acesso em: 05 jan. 2015. Citado na página 27.

WALKO, R. L.; AVISSAR, R. The ocean-land-atmosphere model (olam). part i: Shallow-water tests. *Monthly Weather Review*, v. 136, n. 11, p. 4033–4044, 2008. Citado 3 vezes nas páginas 41, 43 e 44.

WILKINSON, B.; ALLEN, C. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Pearson/Prentice Hall, 2005. (An Alan R. Apt book). ISBN 9780131405639. Disponível em: <https://books.google.co.in/books?id=F8C2QgAACAAJ>. Citado 2 vezes nas páginas 33 e 35.

XEN: Site. 2015. Disponível em: <http://www.xenproject.org/>. Acesso em: 05 jan. 2015. Citado na página 27.

Índice