UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

TECNÓLOGO EM REDES DE COMPUTADORES

RÉGIS MATHEUS SILVEIRA MAIA

FERRAMENTA PARA CRIAÇÃO DE CLUSTERS VIRTUAIS PARA O ENSINO DE PROGRAMAÇÃO PARALELA E DISTRÍBUIDA

QUIXADÁ

2016

RÉGIS MATHEUS SILVEIRA MAIA

FERRAMENTA PARA CRIAÇÃO DE CLUSTERS VIRTUAIS PARA O ENSINO DE PROGRAMAÇÃO PARALELA E DISTRÍBUIDA

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Redes de Computadores da Universidade Federal do Ceará como requisito parcial à obtenção do título de Bacharel/Tecnólogo em Redes de Computadores. Área de Concentração: Computação

Orientador: Profº. Antonio Rafael Braga
Co-orientador: Profº. Paulo Antonio Leal Rego

QUIXADÁ
2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M188f Maia, Régis Matheus Silveira.
Ferramenta para criação de clusters virtuais para o ensino de programação paralela e distribuída / Régis Matheus Silveira Maia. – 2016.
45 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2016.
Orientação: Prof. Me. Antonio Rafael Braga.
Coorientação: Prof. Me. Paulo Antonio Leal Rego.

1. Virtualização. 2. Computação de alto desempenho. 3. Cluster (Sistema de computador). I. Título.
CDD 004.6

REGIS MATHEUS SILVEIRA MAIA

FERRAMENTA PARA CRIAÇÃO DE CLUSTERS VIRTUAIS PARA O ENSINO DE PROGRAMAÇÃO PARALELA E DISTRIBUIDA

Monografia apresentada ao curso de Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel/Tecnólogo em Redes de Computadores. Área de concentração: Computação.

Aprovada em: 14 / Julho / 2016.

BANCA EXAMINADORA

_____

Profº. Antonio Rafael Braga (Orientador)
Universidade Federal do Ceará (UFC)

_____

Profº. Paulo Antonio Leal Rego (Co-orientador)
Universidade Federal do Ceará (UFC)

_____

Profº. João Marcelo Uchôa de Alencar
Universidade Federal do Ceará (UFC)

Aos meus pais, Mauro Regis de Oliveira Maia e Regina Silveira Maia por todo o esforço que fizeram para ajudar essa minha trajetória acadêmica e na construção da minha educação.

AGRADECIMENTOS

Agradeço primeiramente a Deus por permitir o dom da vida e proporcionar essa grande conquista na minha vida.

Aos meus pais, que sempre deram todo apoio para que eu pudesse realizar esse sonho de concluir mais uma etapa de estudos.

Aos professores e grandes mestres Antonio Rafael Braga e Paulo Antonio Leal Rego.

A todos os professores do curso de Redes de Computadores da UFC, Campus Quixadá, que foram muito importantes para minha formação acadêmica.

A todos da minha família, tios, primos, que mim apoiaram, para que eu pudesse chegar até essa conquista

Aos meus amigos do AP e curso, Joel Sousa, Renato Cavalcante e Wellington Rebouças, que sempre estiverem comigo durante esse trajeto.

“Para nós os grandes homens não são aqueles que resolveram os problemas, mas aqueles que os descobriram”.

(Albert Schweitzer)

RESUMO

Em faculdades e universidades, o ensino dos conceitos de computação paralela e distribuída a estudantes de tecnologia da informação (TI) tem-se tornado um assunto cada vez mais importante. Para prática desses conceitos é necessário um ambiente que propicie essas a realização da experimentação em ambiente real de execução, que no caso da computação paralela e distribuída nem sempre e de facial acesso. Nesse sentido, este trabalho visa criar uma ferramenta para o ensino de programação paralela e distribuída em faculdades e universidades, para que os alunos possam praticar os conceitos aprendidos dentro da sala de aula, em um ambiente virtual e semelhante ao real, onde o aluno poderá criar seu próprio *cluster* virtual, para a prática dos conceitos. Para validar a ferramenta desenvolvida foi criado um estudo de caso, onde foi feita uma análise de desempenho, onde analisamos o tempo de criação de *clusters* virtuais na ferramenta desenvolvida.

Palavras Chaves: Virtualização. Computação de Alto Desempenho. *Cluster* Virtual.

ABSTRACT

In colleges and universities, teaching the concepts of parallel and distributed computing information technology students (IT) has become an increasingly important issue. To practice these concepts need an environment where those conducting the trial in real execution environment, which in the case of parallel and distributed computing is not always facial access. In this sense, this work aims to create a tool for parallel programming teaching and distributed in colleges and universities, so that students can practice the concepts learned in the classroom in a virtual and similar to the real environment where the student can create your own virtual cluster, to practice the concepts. To validate the developed tool was created a case study where a performance analysis was done, where we analyzed the time of creation of virtual clusters in developed tool.

Keywords: Virtualization. High Performance Computing. Virtual Cluster.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE GRÁFICOS

SUMÁRIO

1 INTRODUÇÃO

Na atualidade, pode-se observar que o paralelismo está crescendo no mundo da computação. Podemos observar que muitas empresas já estão prestando serviços como venda do tempo de execução em *cluster* de computadores. Entre essas empresas, podemos citar: *Amazon Web Services*, *Microsoft Azure* e *Rackspace* (SHOOP et. al. 2012). Nas faculdades e universidades, o ensino dos conceitos de computação paralela e distribuída a estudantes de TI tem-se tornado um assunto importante e que está ganhando impulso entre os docentes (SHOOP et. al. 2012). Um problema que muitas instituições enfrentam é a falta de recursos computacionais para a criação de *clusters* físicos do tipo *Beowulf*, onde esse problema se torna um motivo para a construção de *clusters* virtuais do tipo *Beowulf*. Muitas das instituições não possuem pouco ou nenhum acesso a um ambiente de computação paralela para o ensino de conceitos, sendo que o ensino desses conceitos em um ambiente real ou semelhante ao real se torna mais eficiente o aprendizado (LUDIN et al. 2013).

O *cluster* do tipo *Beowulf* é formado por computadores pessoais. Foi criado por dois pesquisadores da NASA, Thomas Sterling e Donald J. Becker, em 1994, com a finalidade de processar informações espaciais recolhidas pela entidade (BACCELAR, 2010). O motivo da criação desse tipo de *cluster* foi a necessidade de processar muitas informações, pois para processar uma grande quantidade de dados, precisava-se de um supercomputador, sendo que naquela época, um supercomputador custava um preço muito alto (BACCELAR, 2010). Uma das características principais desse tipo de *cluster* é a utilização de sistemas operacionais de código aberto.

Os computadores pessoais (PCs – *Personal Computers*) atuais possuem grande poder de processamento em estado ocioso. Isso é facilmente observado em laboratórios de informática de faculdades e universidades. Geralmente, esses computadores são utilizados apenas em horários de aulas e durante atividades práticas por uma pequena quantidade de alunos, e desligados no restante do dia, finais de semana e feriados (BESERRA et al. 2013). Uma boa alternativa para o bom aproveitamento do potencial de processamento ocioso desses computadores seria a criação de *clusters* virtuais do tipo *Beowulf*. Tais *clusters* permitem criar um ambiente idêntico ao de um *cluster* físico propriamente dito.

No trabalho de Alexandre (2011), é proposta a criação de uma arquitetura para disponibilizar serviços para aplicações de gerência de rede de computadores, que permitem a construção automatizada de *clusters* de Máquinas Virtuais (MV) – também chamado de *cluster* virtual, permite controlar o ciclo de vida das MV, monitorar o ambiente e gerenciar as

aplicações executadas sobre o *cluster* virtualizado. O trabalho de Butler, Lowry e Pettey (2005) apresenta um sistema de código aberto para a criação de *clusters* virtuais, com o objetivo de testar *software* e fazer a depuração dos mesmos para identificar problemas com o *software.* Já no trabalho de Shoop et al. (2012), são apresentadas duas arquiteturas de *clusters* virtuais usadas em ambientes acadêmicos para o aprendizado de conceitos fundamentais em computação paralela para alunos que possuem, em sua grade curricular, disciplinas de computação paralela e distribuída.

Contribuir para a construção de uma ferramenta que possibilite a construção de *clusters* virtuais, certamente é uma alternativa para o bom aproveitamento de parte do *hardware* ocioso nos laboratórios de informática. Partindo dessa motivação o trabalho tem o objetivo desenvolver uma ferramenta para criação de *clusters* virtuais do tipo *Beowulf*. O objetivo é permitir aos alunos de programação paralela e distribuída praticarem os conceitos aprendidos em sala de aula e o desenvolvimento de *software* em um ambiente semelhante ao de um *cluster* físico propriamente dito. Assim, os estudantes poderão testar o *software* desenvolvido em um ambiente real, permitindo identificarem erros e *bugs* e melhorarem a qualidade do software desenvolvido.

Para atingir esse objetivo, decidiu-se por utilizar ferramentas de código aberto para construir a arquitetura básica do *cluster virtual,* portanto, não necessita de gastos com *software* e *hardware*, já que serão utilizados computadores existentes nos laboratórios de informática.

O trabalho busca ajudar os alunos que possuem, em seu currículo, a disciplina de programação paralela e distribuída, melhorando a didática de ensino do professor que ministra a disciplina e fazendo com que os alunos realmente fixem os conceitos de programação paralela e distribuída e possam praticar em um cenário próximo da realidade.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver e avaliar uma ferramenta para auxiliar a criação de clusters virtuais com o intuito de permitir aos estudantes de programação paralela e distribuída praticarem o desenvolvimento de *software* em um ambiente semelhante ao de um cluster físico propriamente dito.

1.1.2 Objetivos Específicos

- Identificar qual técnica de virtualização e hipervisores utilizar.

- Identificar as principais técnicas e linguagens para programação paralela e distribuída.
- Criar um modelo da ferramenta para facilitar a criação de ambiente virtual para execução de aplicações paralelas e distribuídas.
- Validar a ferramenta desenvolvida.

2 TRABALHOS RELACIONADOS

Alexandre (2011) propõe criar uma arquitetura que visa fornecer serviços para o gerenciamento de um *cluster* de máquinas virtuais, permitindo a criação automatizada de *clusters* virtuais, controlar o ciclo de vida das MV, monitorar esses ambientes e a gerência das aplicações executadas no *cluster*. A semelhança entre esse trabalho e a arquitetura proposta por Alexandre (2011) é a criação automatizada de *clusters* virtuais. A diferença é que Alexandre (2011) propõe outros tipos de serviços focados apenas no gerenciamento do ciclo de vida de um *cluster* virtual, já neste trabalho o foco é criar *clusters* virtuais para a execução de aplicações feitas por estudantes de disciplinas de programação paralela e distribuídas.

No trabalho de Butler, Lowry e Pettey (2005), é proposto um sistema que tem como objetivo a construção de maneira simplificada de *clusters* virtuais por parte do usuário, facilitando assim a construção de ambientes de *cluster* virtualizado personalizado. A semelhança entre o trabalho que apresentamos e o trabalho de Butler, Lowry e Pettey (2005) é a construção de *clusters* virtuais com uma interface amigável para o usuário criar um ambiente personalizado e com um nível de conhecimento básico, o trabalho também utiliza *software* de código aberto e sem a necessidade de comprar *hardware* novos para a construção do *cluster* virtual. A diferença que mais se destaca é que, em Butler, Lowry e Pettey (2005), utiliza-se o QEMU para emulação do processador, no nosso trabalho utilizamos a virtualização baseada em *containers* e o *Docker* para a criação dos *containers*. Dessa forma, esperamos que o tempo para criação dos clusters seja menor. Uma vez que, conforme Xavier et al. (2014), a virtualização baseada em *containers* tem apresentado desempenho melhor que outros tipos de virtualização tradicional.

O trabalho de Shoop et al. (2012) fala sobre a importância da computação paralela não só no meio empresarial, mas como no meio acadêmico. Nesse trabalho ele cita também como pode ser criado *clusters* virtuais a partir de computadores ociosos em laboratórios de informática nas universidades. No trabalho, os autores mostram dois *clusters* virtuais. O primeiro é o *cluster MistRider*, em *Saint Olaf College*, e o outro é o *cluster Selkie* em

Macalaster College. A semelhança do trabalho com o trabalho desenvolvido é que os dois *clusters* citados no trabalho tem a finalidade de ajudar no ensino de computação paralela no meio acadêmico, utilizando-se de recursos de *hardware* ociosos dos laboratórios de informática e suporta o MPI (*Message Passing Interface*). A maior diferença é o tipo de virtualização e o *hipervisor* utilizado para construção das VM, onde o tipo de virtualização utilizada pelos autores é a virtualização assistida por *hardware* e o *hipervisor* é o KVM. Já este projeto utiliza virtualização em nível de sistema operacional e utiliza o *Docker* como *hipervisor* para a criação das VM.

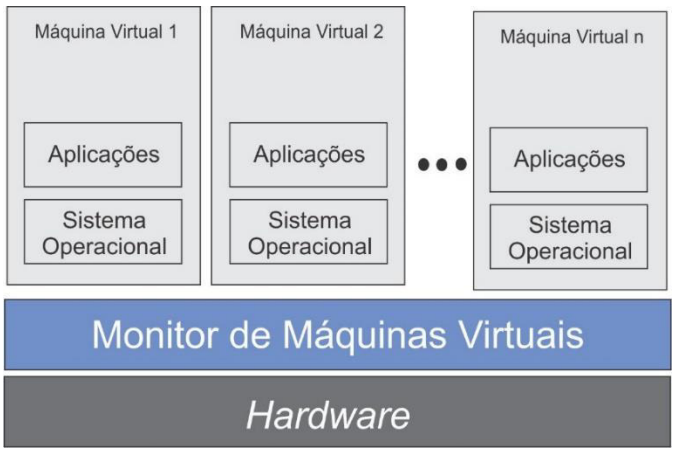
3 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão definidos alguns conceitos básicos para um bom entendimento desse projeto. Nas seções 3.1, 3.2 e 3.3 são apresentados conceitos de Virtualização, Computação de Alto Desempenho e *Clusters* Virtuais, respectivamente.

3.1 Virtualização

O conceito de virtualização surgiu na década 1960 pela IBM *Corporation*. Pode-se definir virtualização como uma camada de abstração de *software* que é introduzida entre o *hardware* e o sistema operacional. Pode ser definida também como uma abstração dos recursos computacionais ou como uma técnica para esconder as características físicas dos recursos computacionais (REGO, 2012). Essa camada, que é introduzida, é mais conhecida como monitor de máquina virtual, do inglês: *Virtual Machine Monitor* (VMM) ou hipervisor (*hypervisor*) (SAHOO; MOHAPATRA; LATH, 2010). Este *software* cria ambientes de simulação de computação, mais conhecidos como VM (*Virtual Machine*). Uma VM também pode ser definida como um ambiente operacional que se comporta como se fosse uma máquina física independente. A Figura 1 é um exemplo de um ambiente que utiliza virtualização. No exemplo, o ambiente virtual contém a camada do *hardware*. Logo acima, vemos o monitor de máquinas virtuais e, mais acima, as VM rodando sistemas operacionais distintos com aplicações distintas.

Figura 1 - Exemplo de um ambiente que utiliza virtualização



Fonte: (ALEXANDRE, 2011).

O trabalho de Ferreira (2013) mostra que, atualmente, a virtualização vem sendo muito utilizada por trazer inúmeras vantagens, entre elas estão:

- Redução de equipamentos – implicando em economia de energia e espaço físico.
- Adaptação a diferentes cargas de trabalho – atribuindo e retirando recursos computacionais de uma VM de acordo com sua necessidade.
- Balanceamento de carga – movimentando recursos de uma VM para outra e assim balanceando e homogeneizando a carga do sistema.
- Compartilhamento de recursos – diferentemente de um ambiente não virtualizado em que todos os recursos de *hardware* são dedicados aos programas em execução, em ambientes virtualizados, as máquinas virtuais compartilham o mesmo *hardware* (SEO, 2009).
- Isolamento – programas executados em uma VM não enxergam programas executando em outra VM (SEO, 2009).

3.1.1 Tipos de Virtualização

No trabalho de Ferreira (2013) são citados três tipos de virtualização que podem ser utilizados para prover uma infraestrutura de TI, são elas:

- Virtualização de servidores – é definida como um processo no qual os sistemas operacionais, programas e aplicações utilizadas em servidores físicos, passam a ser

virtualizados utilizando máquinas virtuais e assim fazer um uso mais eficiente dos recursos, como *hardware* (FERREIRA, 2013).

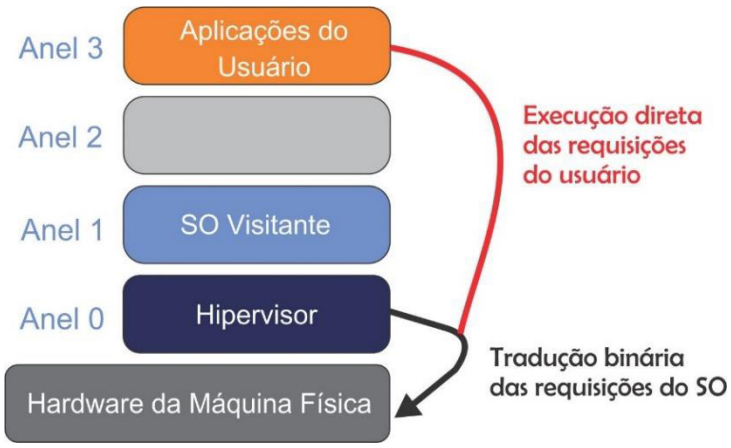
- Virtualização de aplicativos – os aplicativos são executados na máquina local ou virtual, mas não tem permissão para nenhum tipo de alteração. Este ambiente virtual age como uma camada entre a aplicação e o sistema operacional.

- Virtualização de *desktops* – esse tipo de virtualização possui capacidade de executar diversos desktops em um ou mais servidores físicos. Esse tipo de virtualização é diferente do modelo tradicional, onde os usuários executam os sistemas localmente. Na virtualização de desktops, é adotado o modelo cliente-servidor, onde todos os programas, aplicações, processos e dados são mantidos e executados de forma centralizada.

Assim como existem vários tipos de virtualização, existem diferentes formas de fazer virtualização (REGO, 2012). A grande diferença entre essas formas de fazer virtualização é como são tratadas as instruções privilegiadas ou *hipercalls* para executarem no *hardware*. Para definir como sistemas operacionais podem gerenciar as instruções serão executadas no hardware, existem quatro níveis de privilégio na arquitetura x86 mais conhecidos como anéis, numerados de 0 a 4 (REGO, 2012). A seguir, de acordo com Rego (2012), listamos as diferentes maneiras de fazer virtualização.

- Virtualização total (*Full Virtualization*) – simula todo o *hardware* para permitir que o sistema operacional convidado seja executado de maneira isolada sem precisar alterar o *kernel* do sistema operacional visitante. Como é visto na Figura 2, é utilizada uma técnica de tradução binária e de execução direta para executar as chamadas de sistemas, onde essas chamadas passam pelo hipervisor. Por sua vez o hipervisor, que está no Anel 0, manda as instruções diretamente para o *hardware* (VMWARE, 2007). Ela pode ser definida também como uma forma de prover uma réplica virtual do *hardware* subjacente (CARISSIMI, 2008). Como exemplo de hipervisores de virtualização total, podemos citar o Virtualbox.

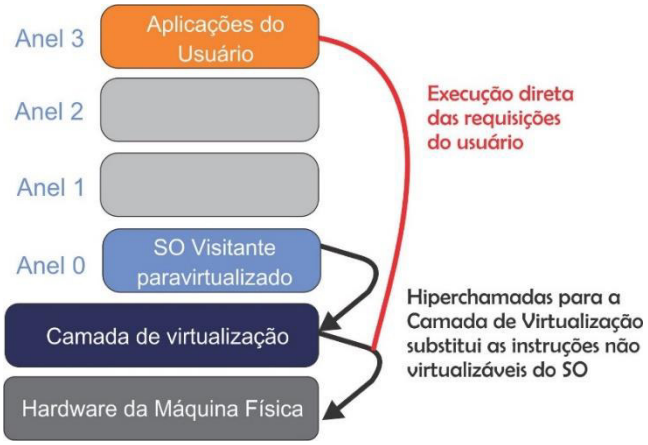
Figura 2 - A tradução binária, abordagem de virtualização x86



Fonte: (VMWARE, 2007).

- Paravirtualização – ao contrário da virtualização total, neste tipo de virtualização, o *kernel* do SO visitante é modificado especificamente para executar no hipervisor (REGO, 2012). Neste tipo de virtualização, ao contrário da virtualização total, o sistema operacional visitante pode executar chamadas de sistema diretamente no *hardware* da máquina física como visto na Figura 3, exceto as hiperchamadas ou *hypercalls* (VMWARE, 2007). As hiperchamadas são interceptadas pelo hipervisor, que por sua vez, executam as chamadas no *hardware* da máquina física. No Anel 3 ficam as aplicações do usuário, no Anel 0 está o SO visitante paravirtualizado e abaixo do SO visitante está a camada de virtualização, onde fica o hipervisor. Como exemplo de hipervisores que trabalham com paravirtualização, podemos citar o Xen (ANISH et al. 2014).

Figura 3 - Abordagem com paravirtualização na arquitetura x86

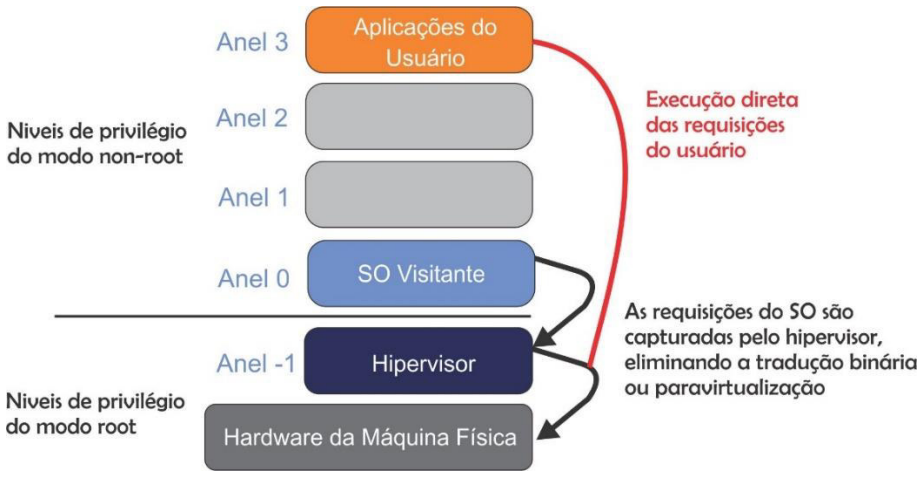


Fonte: (VMWARE, 2007).

- Virtualização no nível do sistema operacional – este tipo de virtualização utiliza partições isoladas em uma única máquina física. Neste tipo, instala-se a camada de *software* de virtualização em cima do sistema operacional, logo, todas as VM executam o mesmo sistema operacional do hospedeiro, sendo que os recursos são independentes para cada uma das VM e as mesmas possui o próprio sistema de arquivos (REGO, 2012).

- Virtualização assistida por *hardware* – esse tipo de virtualização utiliza recursos de virtualização integrados nas últimas gerações de CPU da Intel e AMD (REGO, 2012). Nesse tipo, o próprio *hardware* possui, em sua arquitetura, suporte para a execução de máquinas virtuais de modo isolado. Neste tipo de CPU, é acrescentado um novo nível de privilégio: é incluído então o Anel -1, em que o hipervisor pode operar e o sistema operacional visitante pode operar no Anel 0 da CPU, como pode ser visto na Figura 4 (REGO, 2012).

Figura 4 - Abordagem com virtualização assistida por hardware com arquitetura x86



Fonte: (VMWARE, 2007).

No trabalho desenvolvido, será utilizada a virtualização a nível de sistema operacional, pois será utilizado o *Docker,* como *hipervisor* para criação das máquinas virtuais do *cluster*.

3.2 Computação de Alto Desempenho

Tanto no passado, como, nos dias atuais, pode-se notar uma grande gama de aplicações que necessitam de um alto poder de processamento de dados, tais como:

mapeamento genético, previsões meteorológicas, renderização de gráficos e etc (BACCELAR, 2010).

Atualmente apenas uma pequena porcentagem de universidades tem a cultura de computação de alto desempenho, ou seja, poucas universidades tem o costume de utilizar supercomputadores ou *clusters* dedicados para processar aplicações que necessitam de alto poder de processamento (AHMED et al. 2013).

Existem duas formas de se conseguir alto poder de processamento. A primeira é a criação de sistemas fortemente acoplados ou supercomputadores. A grande vantagem desse tipo de sistema é o alto poder computacional. Por outro lado, o grande problema é que os mesmos têm um custo muito elevado para que possam ser implementados (BACCELAR, 2010). A segunda forma, ou uma alternativa, seria a criação de sistemas fracamente acoplados, que no inglês pode se chamar *cluster*. Um *cluster* é um agrupamento de computadores comuns, conectados por uma rede de conexão. O mesmo processa tarefas de forma paralela, concorrente e transparente, aparentando ser apenas um único sistema para o usuário (BACCELAR, 2010).

Essas duas formas de conseguir alto poder de processamento podem ser entendidas como sistemas que proveem computação de alto desempenho. A primeira com custo bastante elevado para a implementação e a outra podendo ser implementada com um custo bem menor do que a outra.

A partir desse contexto, podemos definir computação de alto desempenho como uso dos dois métodos de obter alto poder de processamento citados anteriormente, sendo o primeiro o supercomputador e o segundo o *cluster*, para processar aplicações que necessitam de um alto poder computacional e que não podem ser encontrados em computadores comuns.

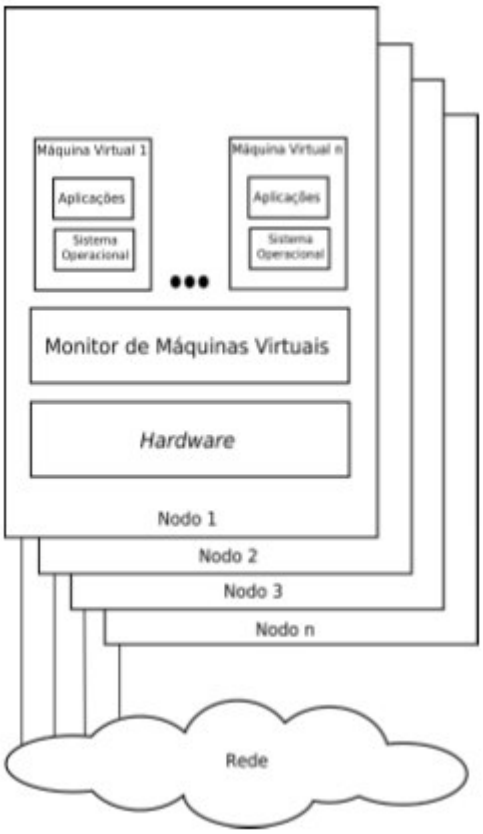
3.3 *Cluster* Virtual

É notável nos dias de hoje que o alto desempenho na computação é obtido com processamento paralelo e distribuído (SILVA et al. 2009). Neste tipo de processamento, são utilizados mais de um processador, em que os mesmos trabalham simultaneamente para obter um resultado mais rapidamente. Por exemplo, cálculos matemáticos, pesquisas espaciais, entre outros (SINGH et al. 2015).

Antigamente, para se obter um alto desempenho de processamento requerido pelos exemplos citados anteriormente, eram necessários *hardwares* muito caros, tais como os chamados supercomputadores. Mas, com os avanços tecnológicos, foram surgindo novas alternativas para tentar resolver esse problema (SILVA et al. 2009). Diante do problema de conseguir alto desempenho de processamento, surgiu o *cluster Beowulf*. Este tipo de *cluster* é constituído por computadores comuns e que são agrupados e conectados por uma rede *Ethernet* (SILVA et al. 2009). Outra característica muito importante nesse tipo de *cluster* é o uso de *software* de código aberto.

Diante desse contexto e conceitos preliminares, apresentamos a seguir o conceito de *cluster* virtual, que é o tipo de *cluster* que será utilizado diretamente nesse trabalho. Um *cluster* virtual pode ser definido como um conjunto de máquinas virtuais executando sobre nós físicos conectados por uma rede de conexão (ALEXANDRE, 2011). A Figura 5 mostra um exemplo de *cluster* de máquinas virtuais. Na Figura 5, é possível perceber que cada nó, como mostrado na seção anterior, está divido em três camadas: na primeira, ficam as aplicações e o sistema operacional convidado, na segunda, o monitor de máquinas virtuais e na terceira e última, o *hardware.*

Figura 5 - Exemplo de um cluster de máquinas virtuais



Fonte: (ALEXANDRE, 2011).

Clusters de máquinas virtuais estão sendo utilizados com bastante frequência, tanto no meio acadêmico, quanto em corporações. O objetivo deste projeto é utilizá-los no meio acadêmico (ALEXANDRE, 2011).

4 PROPOSTA

O trabalho tem como objetivo desenvolver uma ferramenta que possibilite a criação de *clusters* virtuais para que os alunos das disciplinas de computação paralela e distribuída possam programar e executar seus códigos fontes em um ambiente semelhante ao real. Tanto os códigos de criação da ferramenta e todos os *scripts* desenvolvidos nesse trabalho foram publicados do site GitHub[1].

Para implementar a proposta, utilizamos 6 máquinas, que pertencem ao laboratório de Redes de Computadores da Universidade Federal do Ceará (UFC) - *Campus* Quixadá, onde essas máquinas serão utilizadas para a construção da infraestrutura da nuvem privada que será utilizada para a criação dos *clusters* virtuais.

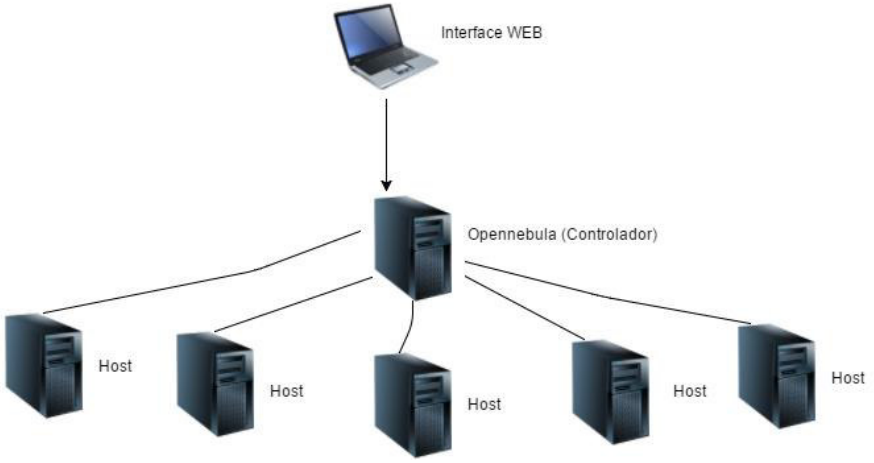
Com a criação de uma infraestrutura de nuvem privada, logo, recursos como, CPU e memória RAM serão agregados e assim aumentando a capacidade de criação de um número maior de máquinas virtuais, que por consequência a possibilidade da criação de um número maior de *clusters* virtuais.

Uma visão geral da infraestrutura utilizada na proposta está representada na Figura 6, onde podemos observar que existe uma nuvem privada, que consiste em 6 máquinas, onde uma será utilizada para que o aluno possa acessar a interface da ferramenta, onde essa interface foi desenvolvida em PHP es outras seis máquinas pertencentes a nuvem. Na visão *top-down,* podemos observar o computador onde está instalado o controlador de gerenciamento da nuvem e o servidor que irá hospedar a aplicação web, e por fim 5 computadores, que integram a nuvem privada, que serviram como *hosts*, por consequência aumentará a quantidade de recursos computacionais da nuvem privada.

No Apêndice 1, pode ser visto o tutorial de instalação e configuração da ferramenta, nele contém o passo a passo.

[1] https://github.com/matheusmaia43/virtualcluster

Figura 6 - Visão geral da proposta



Fonte: Elaborada pelo autor.

4.2 Infraestrutura da nuvem privada

Conforme Jadeja e Modi (2012), o termo nuvem surgiu do mundo das telecomunicações, quando os provedores de acesso à internet começaram a utilizar VPN (*Virtual Private Network*) para comunicação de dados. A computação em nuvem é um conceito importante para o entendimento da infraestrutura que foi criada para a utilização da ferramenta desenvolvida. O Instituto Nacional de Padrões e Tecnologia (*National Institute of Standards and Technology* - NIST) afirma que: "A computação em nuvem é um modelo para permitir, sob demanda de acesso à rede conveniente para um *pool* compartilhado de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento de aplicações e serviços) que podem ser rapidamente fornecidos e liberados com esforço de gerenciamento mínimo ou interação do provedor de serviços".

A infraestrutura básica para criação da nuvem foi criada no laboratório de Redes Computadores supracitado. A mesma é composta por 6 máquinas e em uma delas foi instalado o *software* responsável pela criação do ambiente de nuvem, utilizamos o OpenNebula[2], o restante das máquinas é utilizado como *hosts* da nuvem. No Quadro 1, podemos observar as configurações de cada uma das máquinas dentro do ambiente de nuvem privada.

Quadro 1 - Configuração das máquinas utilizadas no ambiente de nuvem privada

Função	Sistema Operacional	CPU	Memória RAM	HD

[2] http://opennebula.org/

Controlador	Ubuntu 15.04 LTS	Intel Core i3	4 GB	500 GB
Host	Ubuntu 14.04 LTS	Intel Core i7	8 GB	1 TB
Host	Ubuntu 14.04 LTS	Intel Core i7	8 GB	1 TB
Host	Ubuntu 14.04 LTS	Intel Core i7	8 GB	1 TB
Host	Ubuntu 14.04 LTS	Intel Core i7	8 GB	1 TB
Host	Ubuntu 14.04 LTS	Intel Core i7	8 GB	1 TB

Fonte: Elaborada pelo Autor.

Com essa quantidade de máquinas na infraestrutura da nuvem, temos uma boa quantidade de recursos para utilizarmos na criação dos *clusters* virtuais. Observando no Quadro 2, podemos ver a quantidade de recursos totais que as máquinas utilizadas disponibilizam.

Quadro 2 - Total de recursos disponíveis na nuvem

Máquinas	**CPU – (Núcleos)**	**Memória RAM - GB**
Controlador	4 (Núcleos lógicos)	4
Host	4	8
Host	4	8
Host	4	8
Host	4	8
Host	4	8
Total	24 Núcleos	44 GB

Fonte: Elaborada pelo autor.

4.3 Instalação e configuração do OpenNebula

O OpenNebula pode ser comparado com um hipervisor de VMs, sendo que ele, gere recursos de um sistema de computação em nuvem, por exemplo, ele gere máquinas virtuais, redes virtuais, contas de usuário, imagens de armazenamento de dados, dentre outros (RAHIM, 2014).

A instalação do OpenNebula foi feita utilizando o gerenciador de pacotes do Ubuntu, sendo instalada a versão 4.14.0, vale ressaltar que é a versão mais recente.

Inicialmente modificamos o *hostname* de cada máquina, definindo assim, um padrão de nome para cada máquina pertencente a nuvem. Foi necessário também configurar em todas as máquinas o arquivo */etc/hosts*, onde adicionamos os IP e *hostnames* das máquinas. Essa configuração é feita para facilitar a comunicação entre as máquinas, precisando apenas do *hostname* da máquina para referenciá-la. Depois foi preciso a criação de um usuário que será usado para a comunicação entre as máquinas da nuvem, o mesmo foi criado em todas as máquinas.

Para que o controlador possa se comunicar com os *hosts*, se fez necessária a instalação do serviço de acesso remoto, SSH (*Secure Shell*) em todas as máquinas. Configurou-se o SSH para que não necessite de senha, mas, para que esse acesso seja seguro, criamos um par de chaves assimétricas, mais conhecidas como chave privada e a chave pública. Então, enviamos a chave pública para cada um dos *hosts*.

Em sequência, foi preciso instalar e configurar um sistema de arquivos compartilhado via rede, onde foi utilizado o NFS (*Network File System*). No controlador configuramos o arquivo *exports*, adicionando uma linha, apontando para o diretório do OpenNebula, onde ficam as VM. Nos *hosts,* configuramos o arquivo *fstab,* adicionando uma linha no final do arquivo apontando para o diretório compartilhado pelo controlador.

A comunicação do controlador com os *hosts* é feita através de rede, para isso foi preciso fazer uma configuração de rede. Configuramos o arquivo *interfaces*, onde criamos uma interface virtual em modo *bridge,* chamamos a mesma de br0, que recebeu o endereçamento IP via DHCP (*Dynamic Host Configuration Prototcol*), ou seja, o endereço IP é obtido de forma automática.

Para facilitar na administração da nuvem privada, foi instalado o OpenNebula *sunstone*, que é uma interface gráfica, onde o objetivo é o administrador usar funcionalidades encontradas na CLI (*Command Line Interface*) do OpenNebula na forma gráfica.

4.4 Instalação e configuração do Onedock

O Onedock é um conjunto de extensões para o OpenNebula para ser usado o *Docker* como hipervisor, dando suporte para a criação de VMs leves, onde chamamos de *containers*, que se comporta como o KVM e outros hipervisores, dentro do contexto do OpenNebula (OPENNEBULA, 2015).

O *Docker* surgiu em 2013 e logo se tornou popular devido sua facilidade de uso, onde seu propósito é a gestão de *containers* (RAHO et al. 2015). Basicamente o *Docker* é uma plataforma aberta para desenvolvedores administradores de sistemas, onde é possível construir e executar aplicações distribuídas. (LIU; ZHAO, 2014).

Por padrão o OpenNebula não dá suporte à criação de VM utilizando *containers*, dessa forma, pesquisamos um plugin do OpenNebula que adicionasse tal suporte, então o Onedock surgiu como melhor opção. O Onedock é uma extensão para OpenNebula que trabalha em harmonia com *Docker*, além de dá o suporte necessário para a criação de VM utilizando *containers*.

A virtualização baseada em *containers,* tais como, Linux VServer, OpenVZ e Linux Continaers (LXC) surgiram como uma alternativa as máquinas virtuais tradicionais (XAVIER et al. 2014). Nos mesmos é possível observar um aproveitamento melhor do *hardware* e uma diminuição nos custos operacionais (SOPOIALA et al. 2016). Por esses benefícios citados anteriormente sobre o uso de virtualização baseada em *containers*, utilizamos o *Docker,* pois o mesmo utiliza esse mesmo tipo de virtualização citada e por ser uma alternativa leve de virtualização (DUA et al. 2014).

Para instalar o Onedock foi necessária à instalação de alguns pré-requisitos, tais como os pacotes, *docker, xmlstarlet*, *qemu-utils* e *bridge-utils*. As instalações de todos esses pacotes foram feitas através do gerenciador de pacotes do Ubuntu.

Feita a instalação dos pré-requisitos, realizamos a instalação do OneDock a partir da versão disponibilizada no Github[3]. Foi feito o *download* do mesmo, que contém todos os arquivos necessários para o funcionamento do Onedock juntamente com o OpenNebula. Em seguida, configuramos o arquivo de configuração oned.conf para que ele referencie o *Docker* como *hipervisor* do OpenNebula e assim foi possível a criação dos *containers*. Feito isso, foi dada permissão ao usuário oneadmin para que ele possa executar o *Docker*, esse usuário, como dito na seção de instalação do OpenNebula, é usado para a comunicação entre as máquinas da nuvem privada.

[3] https://github.com/indigo-dc/onedock

4.5 Instalação e configuração do servidor WEB

Como o trabalho utiliza uma interface desenvolvida em PHP, foi preciso um servidor que hospede a ferramenta, para isso, instalamos o servidor Apache na mesma máquina em que foi instalado o OpenNebula, onde utilizamos o gerenciador de pacotes do Ubuntu para obter o Apache.

Depois de instalado, copiamos o código fonte da ferramenta (que apresentaremos mais adiante), para o diretório do Apache, assim a ferramenta pode se tornar disponível para o acesso dos usuários finais, os alunos.

4.6 Criando uma imagem contextualizada com o *Docker*

Na criação das máquinas virtuais para a construção do *cluster* virtual foi necessária uma imagem base com todos os requisitos de *software* que é utilizado para que o *cluster* possa ser configurado e funcione corretamente. Para isso criamos um arquivo que é chamado de *Dockerfile*. Nesse arquivo, descrevemos instruções, ou melhor, comandos que utilizamos na linha de comando do Linux, mas, com uma sintaxe apropriada para que o *Docker* possa interpretar. Depois de criado, inserimos no arquivo todos os comandos necessários para a instalação do conjunto de *software,* necessários para a configuração do *cluster* virtual.

No total foram criados dois arquivos *Dockerfile*, um com o conjunto de *software* necessários para o nó mestre do *cluster*. O nó mestre é responsável por distribuir as tarefas entre os nós escravos, o nó escravo, recebe a tarefa do nó mestre, faz o processamento e retorna o resultado para o nó mestre. E por fim o arquivo com o conjunto de *software* necessários para o funcionamento do nó escravo. A partir desses arquivos, conseguimos gerar duas imagens contextualizadas com o conjunto de *software* necessários para a construção do *cluster* virtual. Dentro da imagem do nó mestre foi instalado os seguintes *softwares*: *nfs-server, open-ssh, gcc* e *mpich2* e na imagem do nó escravo instalamos os seguintes: *nfs-client, open-ssh, gcc* e *mpich2*. Criada essas duas imagens bases, a construção do *cluster* é simplificada.

4.7 Automatizando a criação e configuração do *cluster* virtual

Sabe-se que para a criação de qualquer tipo de *cluster* é necessária toda uma configuração para que o mesmo funcione corretamente. Como foi desenvolvida uma ferramenta com uma interface gráfica simples e transparente para o usuário, toda a

configuração necessária é feita a partir de *scripts* desenvolvidos com *Shell Script*. Para criar um *cluster* o aluno irá passar somente alguns parâmetros, para que todo o *cluster* virtual seja criado automaticamente, sem nenhuma necessidade de interação. O aluno informa para a interface WEB, apenas, a quantidade de nós, a quantidade de CPUs e a quantidade de memória RAM.

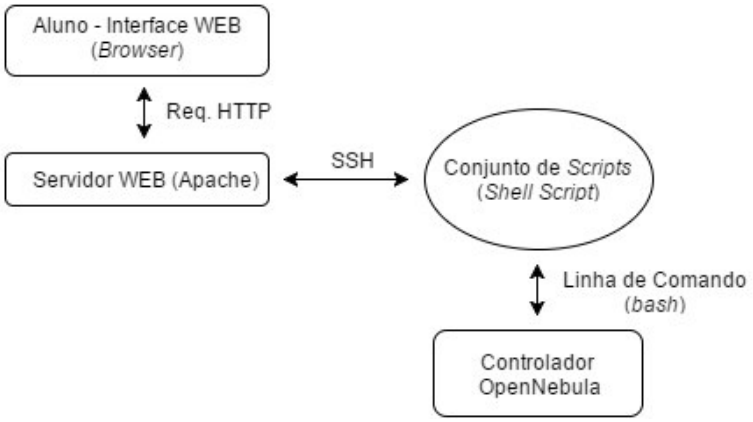
A partir da solicitação do usuário, existe um *script* principal que invoca outros *scripts*. Primeiramente o *script* principal invoca o *script* de criação das VMs, por sua vez, o *script* de criação das VMs, verifica se os recursos solicitados estão disponíveis, se estiver, as VMs são criadas, caso contrário, será informado para o usuário que diminua a quantidade de recursos solicitados. Além disso, o controlador do OpenNebula é responsável por alocar as VMs nos *hosts,* ou seja, existe um algoritmo de alocação, onde o controlador verifica a quantidade de recursos disponíveis em cada *host* da nuvem e de acordo com os recursos disponíveis, é feita a alocação das VM nos *hosts* com mais recursos disponíveis.

Caso dê tudo certo e as VM entrem em execução, um novo *script* é invocado para começar a configurar os nós do *cluster* virtual, o *script* de configuração do *cluster*, envia para os nós do *cluster* um diretório que contém arquivos com os endereços IP das VM e outro com um arquivo utilizado pelo *cluster* para a execução dos códigos, onde nesse arquivo contém o *hostname* de cada nó e três *scripts* usados na configuração do *cluster*. Em suma, esses *scripts* configuram o arquivo *hosts* com os IP e *hostname* de cada nó, configura o *NFS,* onde no nó mestre do *cluster* é compartilhado o diretório em que ficam salvos os códigos fontes enviados pelo o usuário*,* cria um par de chaves assimétricas, para que os nós possam se comunicar entre si sem a necessidade de senhas e por fim troca esse par de chaves entre os nós.

Para entendermos melhor o funcionamento da ferramenta, como é feita toda a comunicação entre o servidor WEB com os *scripts* e o OpenNebula. Na Figura 7 conseguimos entender com mais clareza como funciona toda a comunicação. Como mostra na Figura 7, primeiro o aluno acessa via *browser* a ferramenta, onde é gerada uma requisição HTTP para o servidor que hospeda a ferramenta, em seguida, o servidor WEB retorna a página de *login*, necessária para o uso da ferramenta, depois que o usuário fizer o *login*, o servidor WEB requisita ao conjunto de *scripts* o *status* das VMs do *cluster*, caso exista algum *cluster* criado, por sua vez, o *script* responsável por verificar o *status* das máquinas requisita o controlador OpenNebula e por fim, o controlador retorna uma requisição com a informação das VMs para o *script,* onde o mesmo, retorna para o servidor WEB e finalmente é exibida para o aluno o *status* na interface WEB.

Para concluirmos, vale salientar que toda e qualquer comunicação com o controlador OpenNebula, como solicitação de *status* das VMs, criação de VMs é feita através do conjunto de *scripts*, como mostra a Figura 7.

Figura 7 - Arquitetura geral da ferramenta



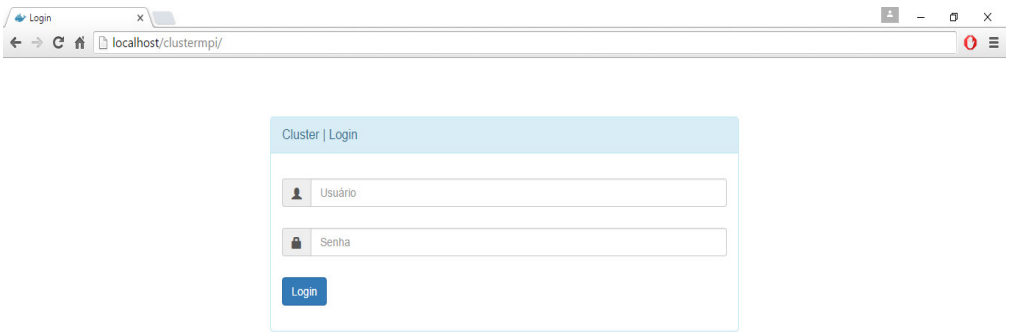
Fonte: Elaborada pelo autor.

4.8 Desenvolvimento e funcionalidades da ferramenta WEB

Depois de toda a infraestrutura montada e funcionando, criamos a ferramenta, que foi desenvolvida em HTML, PHP e Java Script.

Com uma interface simples e intuitiva, primeiramente o aluno deve fazer o *login* na ferramenta para poder usufruir das suas funcionalidades, nas quais podemos citar, uma das principais, que é a criação de *clusters* virtuais, nela possui um simples editor de código, onde o aluno pode editar o seu código, salvá-lo. É possível também fazer o download do arquivo com o codigo-fonte para que o usuário possa executá-lo posteriormente no *cluster*. Como já dito, a ferramenta possui também a função de executar o código desenvolvido pelo aluno. Na tela de autenticação, o aluno informa o nome de usuário e a senha. Na figura 8 podemos visualizar a tela de autenticação da ferramenta.

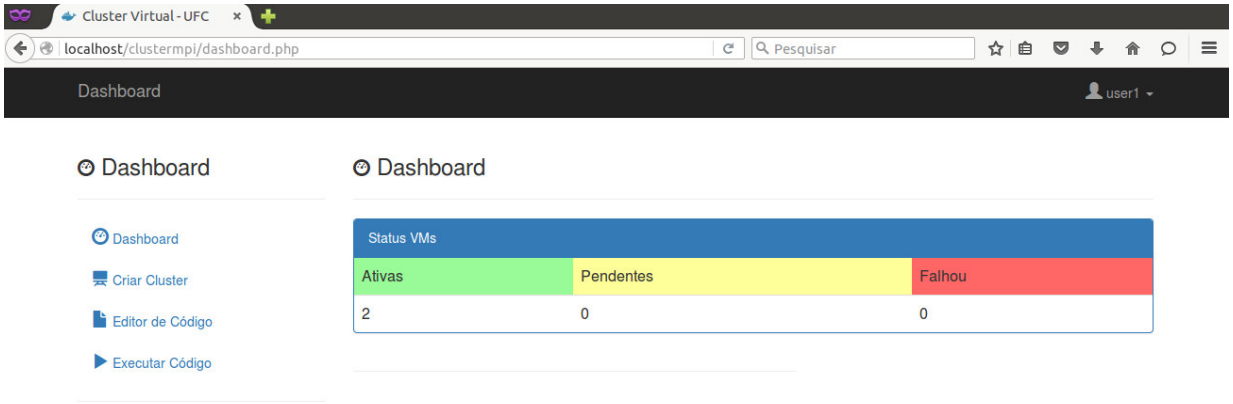
Figura 8 - Tela de autenticação da ferramenta



Fonte: Elaborada pelo autor

Depois que o aluno se autenticar, ele será redirecionado para uma tela que irá conter informações das VM, como exemplo, quantidade de VM ativas, pendentes e que falhou, mais conhecido como *Dashboard*. Podemos observar com melhor facilidade na Figura 9.

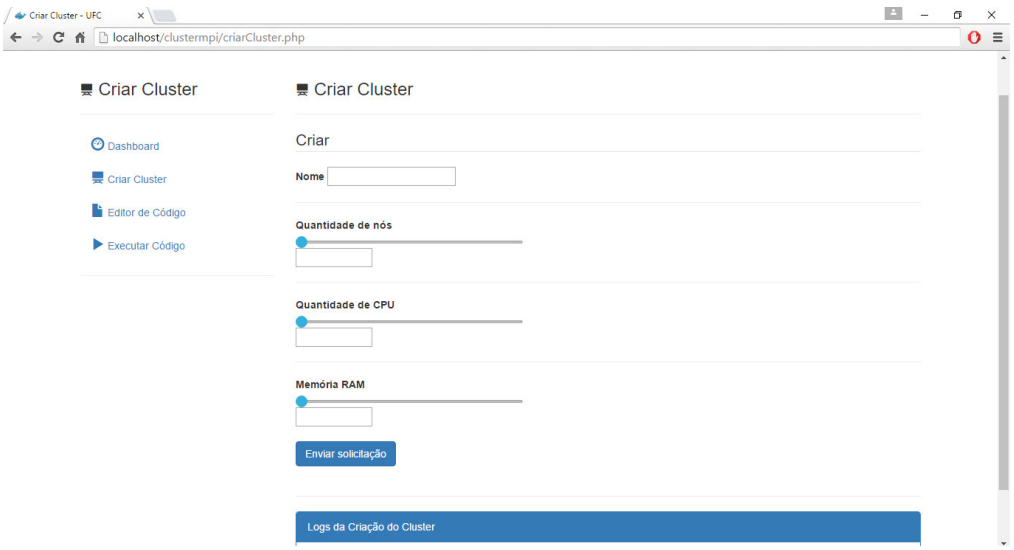
Figura 9 - Tela inicial da ferramenta



Fonte: Elaborada pelo autor.

Como podemos observar na Figura 8, que as funcionalidades da ferramenta são de fácil acesso para o aluno, onde podemos visualiza-las no lado esquerdo superior da tela e que os nomes das funcionalidades são intuitivos. Na tela de criação do *cluster,* ilustrada na Figura 10, podemos observar, que o aluno deve informar somente o nome que dará ao *cluster*, a quantidade de nós do *cluster,* a quantidade de CPU e memória RAM e abaixo do botão enviar solicitação, contém os *logs* da criação do *cluster*.

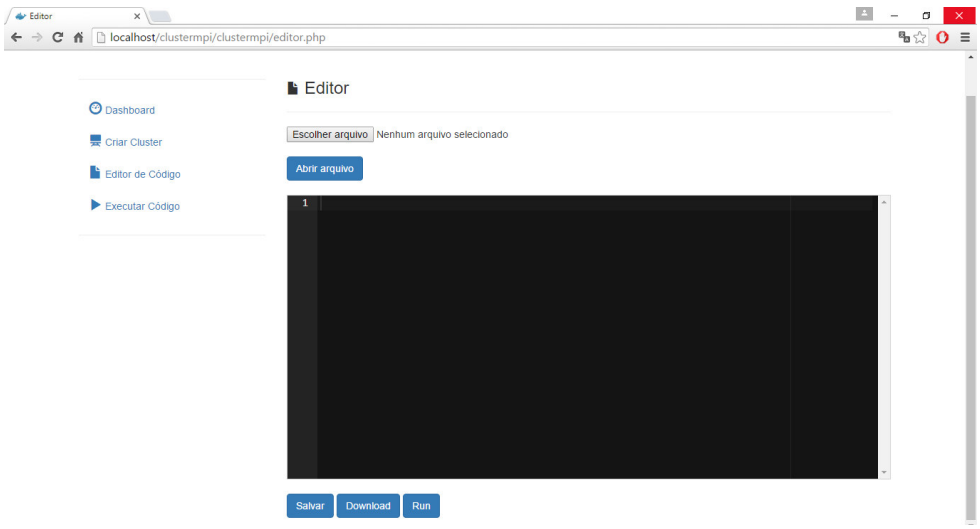
Figura 10 - Tela de criação do cluster



Fonte: Elaborada pelo autor

A ferramenta também dispõe de um simples editor de código, que poderá ser utilizado pelo aluno para edição do seu código fonte, caso necessite. Podemos ver uma ilustração da tela do editor na Figura 11.

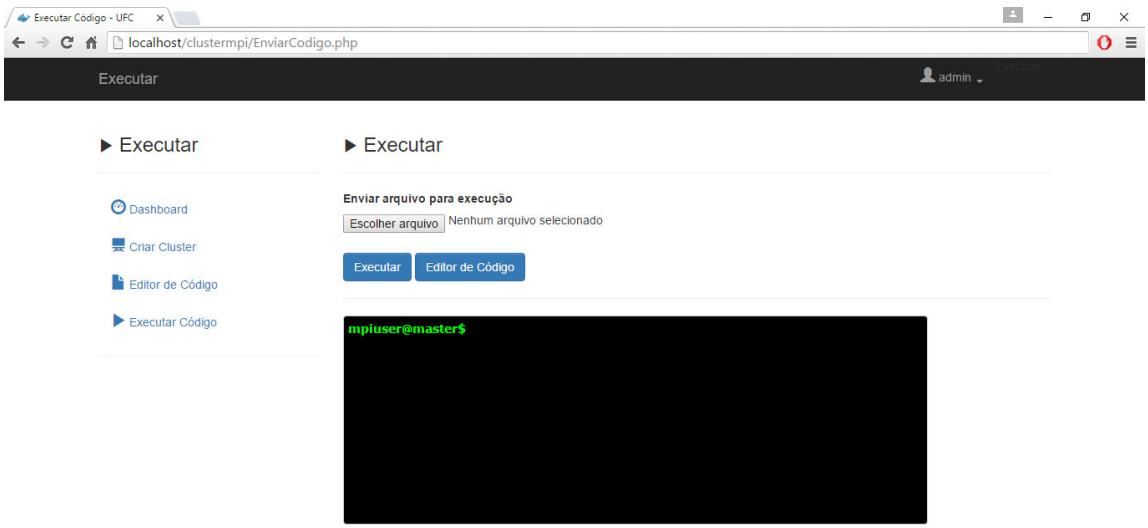
Figura 11 - Tela de edição de código



Fonte: Elaborada pelo autor.

Finalizando, o usuário poderá executar e testar seu código, para isso, o aluno acessa a página de execução de código e faz o *upload* do código fonte e executa, e por fim o aluno recebe o resultado do processamento na tela. A descrição anterior é ilustrada na Figura 12.

Figura 12 - Tela de execução de códigos

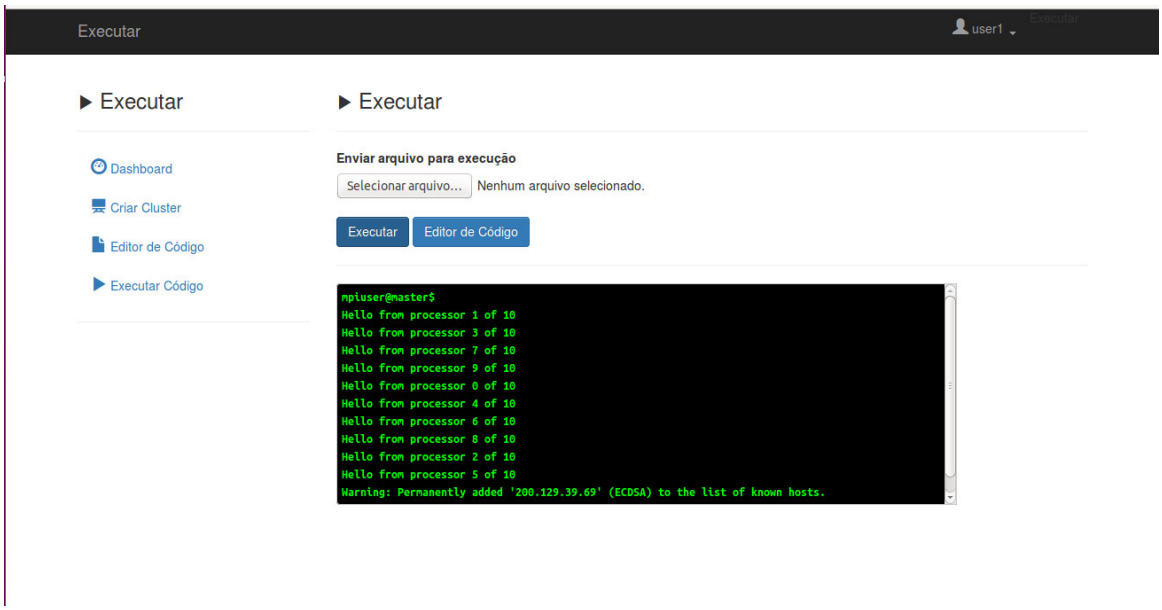


Fonte: Elaborada pelo autor.

4.9 Testes funcionais

Para validarmos a funcionalidade de execução de códigos da ferramenta, utilizamos um pequeno código construído na linguagem C, onde esse código, simplesmente é um *Hello World* para a linguagem MPI (*Message Passing Interface*). Primeiramente, foi feito o *login* na ferramenta, em seguida fizemos o *upload* do arquivo em C e por fim executamos. O resultado desse teste é observado na Figura 13. No Anexo 1, observamos o código utilizado para o teste realizado.

Figura 13 - Execução de código na ferramenta



Fonte: Elaborada pelo autor.

5 ESTUDO DE CASO

Para validação da proposta apresentada nas sessões anteriores, desenvolvemos um estudo de caso na Universidade Federal do Ceará, Campus Quixadá, que visa mostrar a viabilidade da proposta bem como suas limitações em relação a infraestrutura de computação utilizada no estudo de caso.

5.1 Cenário

Para realizar os experimentos foi utilizada a infraestrutura descrita anteriormente na proposta. Para simularmos um ambiente com alunos utilizando a ferramenta desenvolvida, utilizamos uma ferramenta chamada Apache *JMeter*[4], onde ela permite criar *threads*, nas quais, cada *thread* simula um aluno utilizando a ferramenta desenvolvida. Basicamente

[4] http://jmeter.apache.org/

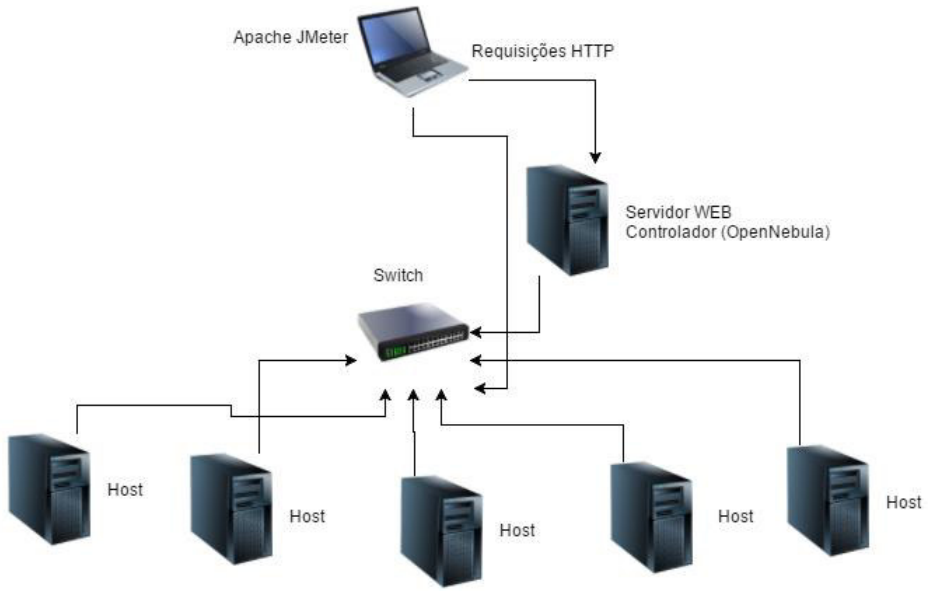
JMeter cria requisições HTTP para a interface WEB da ferramenta, passando parâmetros necessários, como exemplo, na página de *login* se torna necessário a passagem de parâmetros, como o usuário e senha para que o aluno possa se logar. No caso da página de criação de *cluster*, são necessários passar parâmetros, como, nome, quantidade de nós, quantidade de CPU's e memória RAM e assim para as demais funcionalidades da ferramenta.

O *JMeter* é um *software* de código aberto, desenvolvido linguagem Java, originalmente desenvolvido para testar aplicações WEB ().

Para construir o plano de testes com a ferramenta *JMeter*, utilizamos um *plugin* para Google Chrome, chamado *BlazeMeter*[5], onde o mesmo auxilio na construção do plano de testes que foi criado para a realização dos testes na ferramenta desenvolvida. O plano de testes criado pode ser verificado no Apêndice 2.

Na Figura 14 exemplifica o cenário utilizado para realização dos testes da ferramenta desenvolvida. Pode-se observar que o Apache *JMeter* gera e realiza as requisições HTTP para o servidor WEB, onde está hospedada a ferramenta desenvolvida. Vale destacar que a rede foi isolada para a realização dos testes, utilizando uma Switch Dlink de 24 portas e nele ligamos todas as máquinas utilizadas na infraestrutura como mostrado na Figura 12. Um dos motivos do isolamento da rede da nuvem, com a rede do laboratório de Redes de Computadores da UFC Quixadá, foi a falta de endereços IPs, tanto paras as máquinas físicas, quanto paras as máquinas virtuais.

Figura 14 - Cenário de testes



Fonte: Elaborada pelo autor.

[5] https://www.blazemeter.com

Para analisar o desempenho da ferramenta em relação ao tempo de criação dos *clusters* virtuais, como dito anteriormente utilizamos o *JMeter* simulando alguns usuários. Então foi criado alguns cenários, nos quais, podemos observar no Quadro 3.

Quadro 3 - Cenários de testes

Nº	**Métrica**	**Cenários**
1	Tempo de criação do *Cluster*	2 usuários
2	Tempo de criação do *Cluster*	4 usuários
3	Tempo de criação do *Cluster*	6 usuários
4	Tempo de criação do *Cluster*	8 usuários
5	Tempo de criação do *Cluster*	10 usuários
6	Tempo de criação do *Cluster*	12 usuários

Fonte: Elaborada pelo autor.

Como mostrado no Quadro 3, a métrica definida basicamente avalia a principal funcionalidade da ferramenta desenvolvida, que é a criação de *clusters* com ela conseguimos avaliar o tempo que é gasto para a criação de *clusters* e também conseguimos avaliar o quanto de usuários a ferramenta consegue atender e suas limitações. Vale destacar que cada *cluster* é composto por 2 máquinas virtuais, um nó *master* e um nó *slave*, onde cada máquina virtual, possui uma CPU e 512 MB de memória RAM.

5.2 Análise de Desempenho

De acordo com os resultados obtidos pelos testes realizados, foi realizado a análise de desempenho da ferramenta desenvolvida com um nível de confiança de 95%. Vale lembrar que o tamanho da amostra foi no total de 10 amostras para cada cenário. Foi calculado o desvio padrão para cada cenário e por fim, foi calculado a média amostral para cada cenário e a margem de erro dos resultados obtidos. A partir desses cálculos conseguiremos saber o intervalo de confiança com nível de confiança descrito anteriormente. No Quadro 4 é pode-se observar com mais clareza os valores utilizados para o calcular o intervalo de confiança. Os resultados da análise de desempenho estão no Apêndice 4.

Quadro 4 - Valores para o cálculo da análise de desempenho

Nível de Confiança	95%

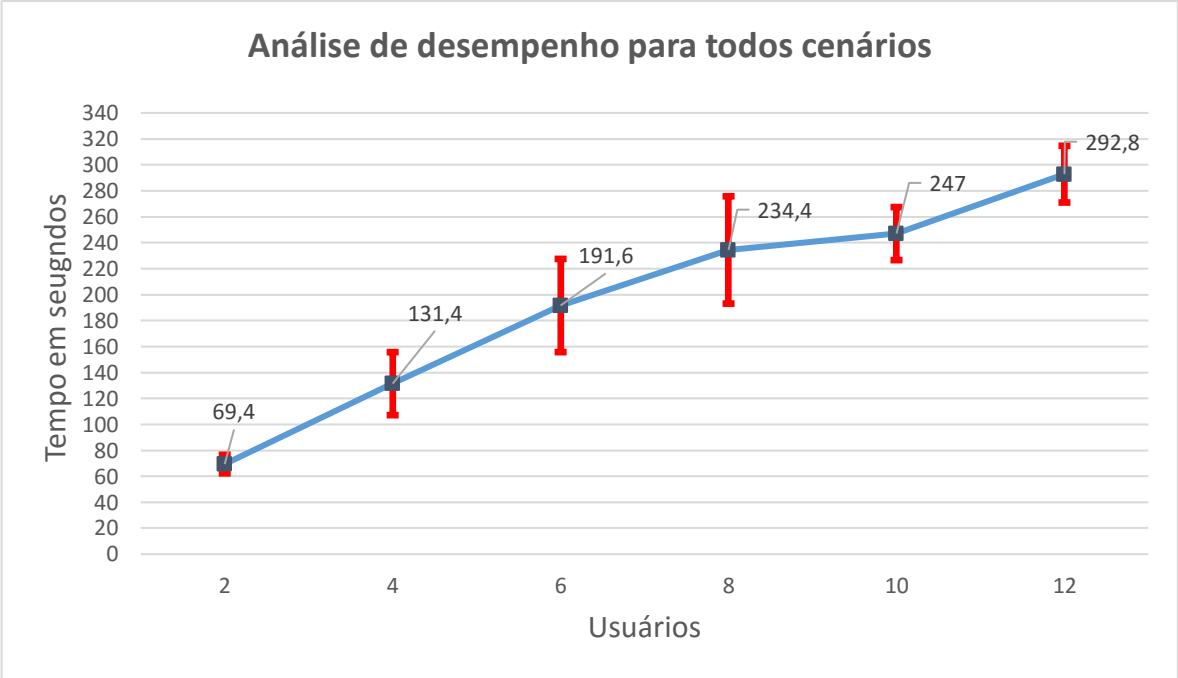
Valor do Alfa	0,05
Número de Amostras	10

Fonte: Elaborada pelo autor.

No Gráfico 1, para 2 usuários, observamos, criando 2 *clusters* virtuais, a média de tempo para a criação desses *clusters* foi de 69,4 segundos, os intervalos de confiança apresentaram pequena diferença de intervalo, ou seja, analisando todas as amostras, os resultados foram aproximados.

Já para o cenário com 4 usuários, observamos no Gráfico 1, simulando 4 usuários, a média de tempo para criação dos *clusters* foi de 131,4 segundos, nesse cenário observamos que os intervalos de confiança variaram bastante em relação a média das amostras.

Gráfico 1 - Análise de desempenho para todos os cenários



Fonte: Elaborada pelo autor.

Ainda no Gráfico 1, com a criação de 6 *clusters* virtuais, simulando 6 usuários acessando a ferramenta desenvolvida, a média de tempo para criação dos 6 *clusters* foi de 191,6 segundos, nesse cenário observamos que os intervalos de confiança variaram bastante em relação a média das amostras, um dos fatores que impactaram no aumento da variação dos intervalos, é o próprio *Docker*, responsável por criar as VMs, no momento dos testes,

percebeu-se que o *Docker* deu algumas travadas e em alguns casos, sendo necessário reiniciar a máquina física.

Para o cenário com 8 usuários, criando 8 *clusters* virtuais, a média das amostras foi de 234,4 segundos, em comparação com o cenário de 6 *clusters*, a média variou 42,8 segundos. Os intervalos de confiança tiveram uma grande variação, como dito anteriormente um fato para tanta variação, é o *Docker*.

Com a criação de 10 *clusters* virtuais, onde simulamos 10 usuários executando a funcionalidade de criação de *clusters* virtuais na ferramenta desenvolvida. A média das amostras foi de 247 segundos. Os intervalos de confiança também tiveram uma grande variação em relação à média amostral.

Finalizando a análise do Gráfico 1, na criação de 12 *clusters* virtuais, simulando 12 usuários criando esses *clusters*, obteve-se uma média amostral de 292,8 segundos na criação desses *clusters*. Os intervalos de confiança, mais uma vez tiveram uma grande variação em relação ao total de amostras.

6 CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma ferramenta que permite aos alunos que estudam computação paralela e distribuída pudessem praticar os conceitos adquiridos em sala de aula em um ambiente que lhes proporcionassem semelhanças a de um ambiente real e que não necessitassem de um investimento com *software* e *hardware* novos, já que é possível utilizar desse ambiente, nos próprios laboratórios da Universidade, onde já possuem uma grande quantidade de máquinas em estado ocioso. Podemos destacar que a ferramenta desenvolvida possui toda uma infraestrutura por baixo, ou seja, é preciso de uma nuvem privada gerenciada com o OpenNebula, vale ressaltar também que a ferramenta funciona independente do hipervisor utilizado pelo OpenNebula, sendo essa uma grande vantagem da ferramenta desenvolvida. A ferramenta em sí, possui uma interface bem intuitiva, simples de ser usada por qualquer aluno.

Como já dito anteriormente na seção de análise de desempenho uma limitação observada na infraestrutura, foi o *Docker,* onde em alguns momentos, o mesmo, travou durante alguns testes com muitos usuários, sendo necessário até o reinicio da máquina controladora do OpenNebula, mas, por outro lado, analisando os resultados obtidos na análise de desempenho, a média de tempo de criação de *clusters* com 12 usuários, acessando

simultaneamente a ferramenta, pode ser considerado como um tempo muito bom, ou seja, tanto a ferramenta, quanto a infraestrutura utilizada pela ferramenta, responderam de forma eficaz a criação de *clusters* virtuais, podendo então ser utilizada para aulas práticas das disciplinas de programação paralela e distribuída.

Como trabalhos futuros, pode-se ser desenvolvida novas funcionalidades para a ferramenta desenvolvida, dentre elas, podemos citar, o suporte para execução de aplicações com o *Hadoop* e RMI.

REFERÊNCIAS

ALEXANDRE, E. (2011). Uma arquitetura baseada em WBEM para o gerenciamento de um cluster de máquinas virtuais. Retrieved from http://repositorio.pucrs.br/dspace/handle/10923/1515.

AHMED, W.; MUTHAHER, M. M.; BASHEER, J. M. Introducing high performance computing (HPC) concepts in institutions with an absence of HPC culture. 2013 Sixth International Conference on Contemporary Computing (IC3), p. 274–277, ago. 2013.

ANISH BABU et al. (2014). System performance evaluation of para virtualization, container virtualization, and full virtualization using Xen, OpenVZ, and XenServer. Proceedings - 2014 4th International Conference on Advances in Computing and Communications,ICACC 2014, 247–250. http://doi.org/10.1109/ICACC.2014.66

APACHE. Apache Jmeter. Disponível em: < https://github.com/apache/jmeter> Acesso em: 19 de Julho 2016.

BACELLAR, H. Cluster: Computação de Alto Desempenho. ic.unicamp.br. 2010.

BESERRA, D.; SOUTO, S. Comparativo de desempenho de um cluster virtualizado em relação a um cluster convencional sob condições equipotentes. IX Workshop em …, p. 3–14, 2011.

BUTLER, R.; LOWRY, Z.; PETTEY, C. C. Virtual Clusters. p. 0–5, 2005.

CARISSIMI, A. Virtualização: da teoria a soluções. 26th Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2008.

DUA, R., RAJA, A. R., & KAKADIA, D. (2014). Virtualization vs containerization to support PaaS. Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014, 610–614. http://doi.org/10.1109/IC2E.2014.41.

LIU, D., & ZHAO, L. (2014). The research and implementation of cloud computing platform based on docker. 2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing (ICCWAMTIP), 475–478. http://doi.org/10.1109/ICCWAMTIP.2014.7073453.

LUDIN et al. (2013). LittleFe The High Performance Computing Education Appliance.

FERREIRA, P. Avaliação de desempenho da escalabilidade em nuvem com virtualização total e a paravirtualização. 2013.

JADEJA, Y.; MODI, K. Cloud computing-concepts, architecture and challenges.2012 International Conference on Computing, Electronics and Electrical Technologies [ICCEET], p. 877–880, 2012.

MAUCH, V.; KUNZE, M.; HILLENBRAND, M. High performance cloud computing. Future Generation Computer Systems, v. 29, n. 6, p. 1408–1416, ago. 2013.

OPENNEBULA. OpenNebula Docker Driver and Datastore. Dec. 2015. Disponível em: < http://opennebula.org/opennebula-docker-driver-and-datastore/> Acesso em: 19 de Julho 2016.

RAHIM, L. A. (2014). Analysis of Design Patterns in OpenNebula.

REGO, P. A. L. FAIRCPU: Uma Arquitetura para Provisionamento de Máquinas Virtuais Utilizando Características de Processamento. 2012. Dissertação (Pós-Graduação em Ciência da Computação) – Departamento de Computação, Universidade Federal do Ceará, Fortaleza, Março/2012.

SAHOO, J.; MOHAPATRA, S.; LATH, R. Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues. 2010 Second International Conference on Computer and Network Technology, p. 222–226, 2010.
SEO, C. Virtualização–Problemas e desafios. IBM Linux Technology Center, v. 1, n. 008278, 2009.

SILVA, SANTOS, ROCHA. Configuração e otimização de um Cluster *Beowulf*. 2009.

SINGH, H. (2015). 2015 Fifth International Conference on Advanced Computing & Communication Technologies A Survey Paper on Task Scheduling Methods in Cluster Computing Environment for High Performance, 241–246. http://doi.org/10.1109/ACCT.2015.64.

SHOOP, E., BROWN, R., BIGGERS, E., KANE, M., Lin, D., & WARNER, M. (2012). Virtual clusters for parallel and distributed education, 517–522. doi:10.1145/2157136.2157287.

SPOIALA, C. C., CALINCIUC, A., TURCU, C. O., & FILOTE, C. (2016). Performance comparison of a WebRTC server on Docker versus Virtual Machine, 295–298.

VMWARE. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. Setembro 2007. White paper, VMware Inc.

XAVIER, M. G., NEVES, M. V., & ROSE, C. A. F. De. (2014). A Performance Comparison of Container-Based Virtualization Systems for MapReduce Clusters. 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 299–306. http://doi.org/10.1109/PDP.2014.78.

APÊNDICES

APÊNDICE 1 – TUTORIAL DE INSTALAÇÃO DA FERRAMENTA

Para se instalar a ferramenta é necessário a instalação e configuração de alguns pré-requisitos, tais como, para a infraestrutura de nuvem privada, é necessária a instalação do software gerenciador de nuvem OpenNebula e o Onedock, que é um *driver* que dá suporte ao OpenNebula trabalhar com o *Docker*. É necessária também a instalação do *docker registry* para que não seja necessária toda vez que for criar o *cluster*, ter que baixar as imagens utilizadas. A instalação de todos esses pré-requistos, podem ser feitas através dos *scripts* disponiveis no Github. Abaixo é mostrado como é feita as instalações.

```
# git clone  https://github.com/indigo-dc/onedock
# cd onedock/install/ubuntu
# ./install-one
# ./install-docker
# ./install-onedock
# ./install-registry
# ./launch-registry
```

Para rodar o *docker-registry* sem a necessidade de certificado para autenticação, é necessário adicionar o comando mostrado abaixo, no arquivo /etc/default/docker.

```
# echo “DOCKER_OPTS="--insecure-registry hostname:5000"” >>
/etc/default/docker
# /etc/init.d/dockerrestart
```

Precisamos criar uma interface virtual para a comunicação entre as máquinas da infraestrutura, chamada de br0, como mostrado abaixo.

```
# apt-get install bridge-utils
--- Agora inserimos no arquivo /etc/network/interfaces as linhas abaixo ---
auto br0
iface br0 inet dhcp
bridge_ports eth0
bridge_maxwait 0
 ----------------------------------------------------------------------------------
# /etc/init.d/networking restart
```

Feito isso, precisamos criar um *datastore* para armazenar as imagens utilizadas para criação do *cluster*, baixar as imagens e criar uma rede virtual. Para facilitar, podem ser usados os *scripts* disponíveis. Abaixo os comandos utilizados.

```
# git clone https://github.com/matheusmaia43/virtualcluster
# cd virtualcluster/scripts_open/
# ./001-test-create-ds
```

```
# ./002-test-create-image-master
# ./002-test-create-image-slave
# ./003-test-create-network
```

Como a ferramenta foi desenvolvida em HTML, PHP e Java Script, se torna necessária a instalação de um servidor WEB e o PHP, onde foi utilizado o Apache2, necessário para a hospedagem da ferramenta, outro pré-requisito para o funcionamento da ferramenta.

```
# apt-get install apache2
# apt-get install php5 libapache2-mod-php5 php5-mcrypt
```

Outro pré-requisito é a instalação do SSH (*Secure Shell*) e o modulo SSH para PHP. Abaixo o comando para a instalação.

```
# apt-get install openssh-server libssh2-1-dev libssh2-php
```

Feita a instalação dos pré-requisitos, copiamos o código fonte da ferramenta, mais precisamente o diretório "clustermpi" para o diretório do Apache /var/www/html, como mostrado logo abaixo. Depois disso, precisamos criar um diretório e copiar o conjunto de scripts para o diretório que é referenciado pela ferramenta. Abaixo os comandos.

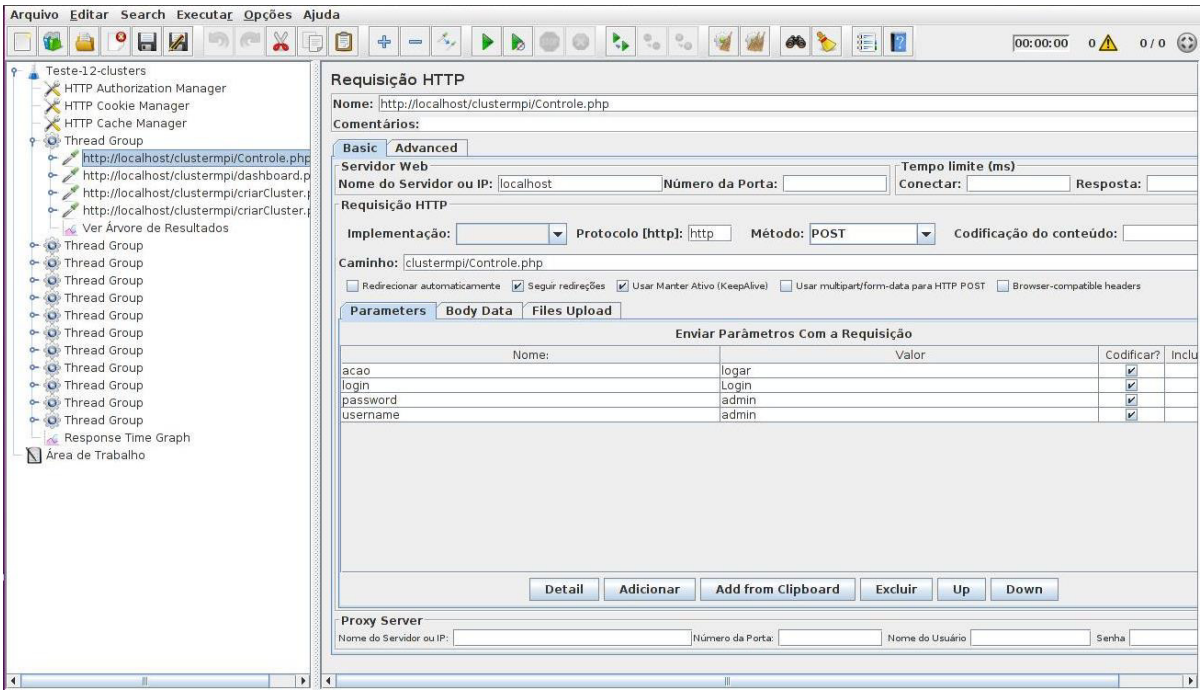
```
# mkdir /var/lib/one/scripts
# cp –r virtualcluster/install/ /var/lib/one/scritps
# cp –r virtualcluster/cria_vms/ /var/lib/one/scritps
# cp –r virtualcluster/configura_cluster/ /var/lib/one/scritps
# cp -f virtualcluster/cria_a.sh /var/lib/one/
```

Por fim, abre o *browser* e acessa a ferramenta http:/localhost/clustermpi. As senhas foram definidas no código fonte da ferramenta, onde existem alguns usuários criados, tais como, admin, user1 até user11, as senhas são iguais os nomes dos usuários.

APÊNDICE 2 – PLANO DE TESTES *JMETER*

Para a realização dos testes com o *JMeter,* é necessário a criação de um plano de teste. Um exemplo de plano de teste utilizado nos testes rodados visualiza-se na Figura 15. Percebe-se que existe vários grupos de *threads*, onde cada grupo representa 1 usuário, este exemplo mostra 12 grupos de *threads*, ou seja, 12 usuários simulados. Para cada grupo, passamos os parâmetros necessários para preencher os campos de usuário e senha, no caso da tela de *login*, onde isso se repete para a tela de criação de *cluster*.

Figura 15 - Plano de testes para 12 usuários



Fonte: Elaborada pelo autor.

APÊNDICE 3 – QUADROS COM OS ARQUIVOS DOCKERFILES

Logo abaixo observamos os arquivos *Dockerfile* utilizado na criação das imagens bases do *cluster*. No Quadro 5 o *Dockerfile* do nó *master* e no Quadro 6 o *Dockerfile* do nó *slave*.

Quadro 5 - *Dockerfile* do nó master

```
FROM rastasheep/ubuntu-sshd

RUN apt-get -y install nfs-server && apt-get -y install build-essential && apt-get -y install
mpich2 && apt-get -y install sshpass
RUN useradd -d /home/mpiuser -m mpiuser && mkdir /home/mpiuser/cloud
```

Fonte: Elaborada pelo autor.

Quadro 6 - *Dockerfile* do nó slave

```
FROM rastasheep/ubuntu-sshd

RUN apt-get -y install build-essential && apt-get -y install mpich2 && apt-get -y install
sshpass
RUN useradd -d /home/mpiuser -m mpiuser && mkdir /home/mpiuser/cloud
```

Fonte: Elaborada pelo autor.

APÊNDICE 4 – QUADROS COM OS RESULTADOS DAS AMOSTRAS E ANÁLISE DE DESEMPENHO

Quadro 7 - Resultados dos Experimentos

	Usuários					
Amostras	2	4	6	8	10	12
1	79	130	334	220	230	257
2	69	135	170	207	232	302
3	64	119	165	363	277	288
4	55	106	177	216	245	310
5	75	106	162	355	330	278
6	74	107	244	216	230	279
7	57	115	180	197	238	300
8	87	137	198	189	236	274
9	54	238	138	188	228	380
10	80	121	148	193	224	260

Fonte: Elaborada pelo autor.

Quadro 8 - Analise de desempenho para todos os cenários

	Usuários					
Análise de desempenho	**2**	**4**	**6**	**8**	**10**	**12**
Tamanho da Amostra	10	10	10	10	10	10
Desvio Padrão	11,52	39,19	57,96	66,7	32,8	35,24
Nível de Conf.	0,05	0,05	0,05	0,05	0,05	0,05
Erro	7,14	24,28	35,92	41,34	20,42	21,84
Limite Inferior	62,27	107,11	155,68	193,06	226,67	270,96
Média	69,4	131,4	191,6	234,4	247	292,8
Limite Superior	76,53	155,69	227,52	275,74	267,33	314,64

Fonte: Elaborada pelo autor.

ANEXOS

ANEXO 1 – TESTE DE EXECUÇÃO DE CÓDIGO NA FERRAMENTA DESENVOLVIDA

O código utilizado para testar a funcionalidade de execução de códigos na ferramenta, no Quadro 9 o código utilizado.

Quadro 9 – Código Hello World

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv) {
   int myrank, nprocs;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

   printf("Hello from processor %d of %d\n", myrank, nprocs);

   MPI_Finalize();
   return 0;

}
```

Fonte: Elaborada pelo autor.