UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

MICHAEL BIFFI NASCIMENTO

CLUSTER DE ALTO DESEMPENHO PARA USO NA DISCIPLINA DE COMPUTAÇÃO PARALELA E DISTRIBUÍDA UTILIZANDO CONTÊINERES

CRICIÚMA

2019

MICHAEL BIFFI NASCIMENTO

CLUSTER DE ALTO DESEMPENHO PARA USO NA DISCIPLINA DE COMPUTAÇÃO PARALELA E DISTRIBUÍDA UTILIZANDO CONTÊINERES

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. Me. Paulo João Martins

CRICIÚMA

2019

MICHAEL BIFFI NASCIMENTO

CLUSTER DE ALTO DESEMPENHO PARA USO NA DISCIPLINA DE COMPUTAÇÃO PARALELA E DISTRIBUÍDA UTILIZANDO CONTÊINERES

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Computação de Alto desempenho

Criciúma, 25 de junho de 2019.

BANCA EXAMINADORA

Prof. Me. Paulo João Martins - UNESC - Orientador

Prof. Esp. Sérgio Coral - UNESC

Prof. Me. Valter Brauth Júnior - UNESC

Dedico este trabalho a DEUS, aos meus pais e minha irmã que com carinho e paciência souberam entender minha ausência.

AGRADECIMENTOS

Agradeço primeiramente a DEUS por ter iluminado meu caminho e por ter me dado forças para alcançar minhas metas.

Aos meus pais que acreditaram em mim e em meu potencial me incentivando nessa caminhada.

Em especial, à minha irmã que me incentivou a não desistir deste curso e com seu empenho auxiliou em todos os momentos.

Aos meus amigos e colegas que sempre estiveram presentes e contribuindo nesta caminhada para que eu chegasse até aqui, em especial a Bernardo Schmitz, Pedro Bonetti e Wesley Ricardo.

Agradeço a todos os professores que fizeram parte da minha formação contribuindo para que construísse uma aprendizagem que fosse significativa para a profissão auxiliando-me no meu crescimento profissional.

Agradeço especialmente ao meu orientador professor Paulo João Martins que com sabedoria em suas orientações para que eu pudesse concluir esta excelente pesquisa.

“A ciência de hoje é a tecnologia de amanhã”.

(EDWARD TELLER)

RESUMO

Nas mais diversas áreas, cientificas e profissionais, é crescente a necessidade por de se obter cada vez mais poder computacional, seja para manter uma enorme quantidade de serviços ativos, ou resolver problemas científicos e matemáticos complexos, porém o alto custo de um supercomputador faz com que se busque alternativas de se obter esse desempenho, sendo uma delas a utilização de *clusters* como por exemplo o tipo *Beowulf* que tem como proposta, unir recursos computacionais local ou geograficamente dispersos trazendo vantagem inclusive em relação aos supercomputadores. Porém, mesmo sendo estes mais viáveis financeiramente, ainda representam gastos elevados, levando as empresas e universidades optarem por utilizar ambientes compartilhados na intenção de reduzir os custos, surgindo com isso, problemas oriundos do compartilhamento de recursos, conflitos de dependências entre outras. Como solução para esses problemas, surgem técnicas de virtualização, sendo uma delas os chamados contêineres, que são mais leves e capazes de isolar as aplicações, passando a ser uma boa alternativa para resolver muitos destes problemas. Utilizando essa ideia, o presente trabalho busca alternativas de implementar um *cluster* de alto desempenho baseado em contêineres, para ser utilizado na disciplina de computação paralela e distribuída, e com isso evitar problemas de compatibilidade, falta de infraestrutura necessária para a implementação de uma *cluster* físico e evitar conflitos de configuração devido a uma instalação malsucedida, além de acelerar o processo de instalação do *cluster*, e com isso dar aos alunos a experiência de desenvolverem programas paralelos, em um ambiente próximo ao real.

Palavras-chave: Contêineres. *Docker*. Computação Paralela. Virtualização. *Cluster.*

ABSTRACT

In the most diverse scientific and professional areas there is a growing need for ever more computational power, either to maintain an enormous amount of active services or to solve complex scientific and mathematical problems. However, the high cost of a supercomputer is a reason to seek alternatives to reach this level of performance, one of them being the use of clusters such as the Beowulf type. It's proposal is to unite local or geographically dispersed computing resources, bringing advantage even in relation to supercomputers. Even though these are more financially viable, they still represent high costs, leading companies and universities to choose to use shared environments in order to reduce them, resulting in problems arising from resource sharing, dependency conflicts and so on. As a solution to these problems, there are virtualization techniques, one of which is called containers, which are lighter and able to isolate the applications, becoming a good alternative to solve many of these problems. Using this idea, the present work seeks alternatives to implement a high performance cluster, based on containers, to be used for parallel and distributed computing and, with this, avoid problems of compatibility, compensate the lack of the necessary infrastructure for the implementation of a physical cluster and avoid configuration conflicts due to an unsuccessful installation. Also, it is possible to accelerate the cluster installation process, and thereby give students the experience of developing parallel programs in an environment close to the real one.

Keywords: *Containers. Docker. Parallel Computing. Virtualization. Cluster*.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

APIs	*Application Programming Interface*
HD	*Hard Disk*
HPC	*High-Performance Computing*
IOMMU	*Input-Output Memory Management Unit*
KVM	*Kernel-Based Virtual Machine*
MB	*Megabytes*
MPI	*Message Passing Interface*
MPICH	*Message Passing Interface over Chameleon*
NFS	*Network File System*
NTP	*Network Time Protocol*
PVFS	*Parallel Virtual File System*
PVM	*Parallel Virtual Machine*
RAM	*Random Access Memory*
RAMS	Regional *Atmospheric Modeling System*
TI	Tecnologia da Informação
TCP	*Transmission Control Protocol*
VM	*Virtual Machine*

SUMÁRIO

1 INTRODUÇÃO

Nos mais variados campos científicos e profissionais, tem-se notado um aumento na necessidade de recursos computacionais de alto desempenho, tendo em vista que estes são de extrema importância na solução de problemas cada vez maiores (CORRÊA, 2016). Segundo Pitanga (2008 apud, CARNEIRO; DUARTE 2018), por conta dos custos elevados da compra de supercomputadores, tem-se a necessidade de uma busca por alternativas mais viáveis. Uma solução popular de baixo custo é o *cluster Beowulf*, que pode vir a ser utilizados por universidades e empresas menores na obtenção de poder computacional de alto desempenho, a um custo relativamente menor que um supercomputador.

Porém mesmo com custos menores, a aquisição de um *cluster Beowulf* não é barato, pois são necessários investimentos para se adquirir os recursos necessários a montagem do mesmo, além de gastos com manutenção. Por conta disso, tende-se a utilizar ambientes compartilhados por grupos de pesquisa e empresas, as quais precisam de um alto poder computacional para suas aplicações, no intuito de diminuir as despesas (CORRÊA, 2016).

Grupos menores enfrentam problemas como: conflito de dependências entre aplicações, recursos indisponíveis, entre outros. Com o avanço das tecnologias surgem os contêineres de software, possibilitando uma abordagem mais flexível, quanto a disponibilização de aplicações (CORRÊA, 2016). Existem vários softwares gerenciadores de contêineres*,* tais como: *Docker*, *Kubernetes*, *Azure Container Service* e *OpenShift Container*, dentre outros.

Segundo Abranches (2016), contêiner é uma tecnologia que permite a criação de instancias isoladas de processamento, proporcionado a virtualização a nível de sistema operacional, de tal forma que um contêiner não desconhece a existência do outro, pois cada um possui seu próprio nível de abstração quanto a camada de rede, memoria e processos.

A disciplina de computação paralela e distribuída costuma despertar o interesse dos alunos para o tema, mas geralmente encontram dificuldades, por não haver computadores ou um laboratório exclusivo para que os alunos possam realizar experimentos, aplicando assim o que é aprendido na sala de aula, isso acontece porque os laboratórios de informática servem para dar suporte para as disciplinas

teóricas e, portanto, não é possível fazer alterações na sua configuração, pois inviabilizaria a sua atividade principal (AMARAL, 2006).

Pensando nisso, este trabalho tem como objetivo a implementação de um *cluster* de alto desempenho, utilizando como base o sistema operacional *Linux* e se propondo a estudar métodos simplificados para que os alunos possam criar o *cluster* no próprio laboratório utilizado na disciplina, para isso, pretende-se utilizar softwares gerenciadores de contêineres buscando uma melhor opção para o caso.

A ideia do projeto é dar suporte a disciplina de computação paralela e distribuída, pois a criação dos *clusters* convencionas costuma tomar um bom tempo de instalação e configuração, tornando o processo custoso.

1.1 OBJETIVO GERAL

Disponibilizar um *cluster* de alto desempenho para ser utilizado na disciplina de computação paralela e distribuída.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa são:

a) aplicar os conceitos de computação distribuída em um *cluster* computacional;
b) implementar o ambiente para um *cluster* de alto desempenho;
c) enunciar as etapas de implementação do contêiner;
d) descrever o contêiner a ser utilizado para a implementação do *cluster* na disciplina de computação paralela e distribuída.

1.3 JUSTIFICATIVA

Com o desenvolvimento acelerado da tecnologia, surgem cada vez mais aplicações que precisam um alto grande computacional, por esse motivo é que são desenvolvidos os supercomputadores, *clusters* e grades computacionais (PRADO; SILVA, 2010). Sendo as Instituições de Ensino um ambiente adequado à pesquisas e estudos, é indispensável que elas busquem sempre se manterão atualizadas, oferecendo aos alunos e pesquisadores meios para produzirem seus projetos que

requerem um alto nível de computação e que são alcançados apenas com recursos computacionais com um alto poder de processamento (PRADO; SILVA, 2010). Porém um problema encontrado na maioria das instituições de ensino, é a falta de recursos para se implementar um cluster físico, uma vez que se faz necessário computadores exclusivos para esse fim, fazendo com que elas optem pelo uso de *clusters* virtuais (MAIA, 2016). Segundo Ludin et al. (2013), a aprendizagem desses conceitos em um ambiente real se torna mais eficaz, uma vez que facilita ao estudante assimilar o conteúdo.

Com a possibilidade de se utilizar máquinas virtuais para simular um *cluster* real, os alunos passam a ter a oportunidade de aplicar os conceitos aprendidos em um ambiente próximo ao *cluster* do tipo físico. Uma máquina virtual pode ser definida como um componente isolado de software com sistema operaciona, desse modo uma não tem conhecimento da existência de outras máquinas virtuais. Para tornar possível esse isolamento, é utilizada uma camada de software chamada *hypervisor*, que tem a responsabilidade de executar e gerenciar o uso dos recursos físicos (ABRANCHES, 2016).

Um tipo de virtualização já citado neste trabalho são os contêineres que apesar de terem semelhanças com a convencional possuem algumas diferenças, pois segundo Alles (2018), nesse tipo de virtualização as aplicações compartilham do mesmo *kernel* de sistema operacional trazendo benefícios e malefícios, porque, se por um lado não se pode executar *kernels* diferentes ao mesmo tempo, por outro a existência de apenas um único sistema operacional executando, não se faz mais necessário a utilização do *hypervisor*, diminuindo assim a perda de desempenho na execução de uma aplicação.

2 ENSINO DE COMPUTAÇÃO PARALELA E DISTRIBUÍDA

O surgimento do hardware *multicore* levou a computação paralela ao centro das atenções. Também trouxe desafios aos programadores de software, pois a única opção para aumentar o desempenho de aplicações sequenciais existentes é utilizar alguma forma de paralelismo. Isso obviamente implica que a comunidade de desenvolvedores de software (incluindo os futuros), busquem aprender modelos de programação paralela e ferramentas para escrever código otimizado para processadores *multicore* e hardware de computação de alto desempenho do inglês *High-Performance Computing* (HPC) (SHAFI et al., 2014, tradução nossa).

De acordo com Kim et al. (2016, tradução nossa), os exercícios práticos realizados no laboratório desempenham um papel importante na engenharia e educações científicas. Muitos institutos acadêmicos desenvolveram seus cursos com aulas práticas para ajudar a acelerar o aprendizado dos alunos em diferentes tipos de laboratórios, como o físico, simulação ou *online*. Especialmente nos cursos em Tecnologia da Informação (TI), exercícios práticos se tornaram um componente essencial do curso porque fornece aos alunos uma oportunidade de aprender e observar como aplicar os conceitos. Segundo Maia (2016) é importante existir um ambiente que permita a realização de experimentos, possibilitando aos alunos a oportunidade de pôr em prática os conceitos aprendidos em sala, se tratando de computação paralela e distribuída, não costuma ser de fácil acesso.

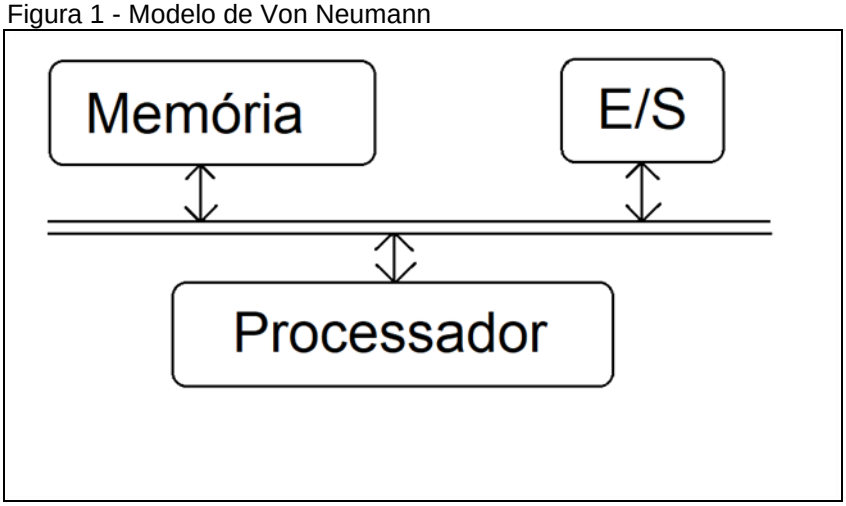
Geralmente, os laboratórios de educação para os cursos de tecnologia requerem uma variedade de equipamentos, como computadores pessoais, servidores, *switches* e assim por diante. A variedade de equipamentos geralmente incorre no custo, que se torna a principal restrição para fornecer ao laboratório um ambiente apropriado (KIM et al. 2016, tradução nossa).

Segundo Kim et al. (2016, tradução nossa), devido à restrição de custos, as tecnologias de virtualização foram amplamente adotadas em muitas áreas. Como a tecnologia virtual fornece um ambiente isolado da máquina hospedeira ou outras máquinas virtuais, tornou-se muito popular como solução para o ensino. A tecnologia de virtualização se torna um componente essencial na educação em TI porque vários sistemas virtualmente isolados podem ser criados, modificados, testados e excluídos facilmente com pouco ou nenhum custo adicional.

2.1 SISTEMAS PARALALOS

Na década de 50 surge o que tornaria o conceito básico de arquitetura dos computadores, passando a ser a base para o desenvolvimento da computação no mundo, o modelo de *Von Neumann* representado pela figura 1. Modelo esse composto apenas por processador, memorias e dispositivos de entrada e saída, sendo que o processador executa as instruções de forma sequencial, de acordo com as ordens de uma unidade central (JARDIM; SÁ, 2010).

Figura 1 - Modelo de Von Neumann



Fonte: Jardim e Sá (2010).

Existem alguns problemas, sendo um deles, segundo Jardim e Sá (2010) o chamado gargalo de *Von Neumann*, que é uma falha na comunicação entre a memória e as outras unidades. Uma das soluções encontradas foi aumentar a velocidade dos dispositivos e executar as tarefas em paralelo, permitindo mudanças nas arquiteturas e originando características nas máquinas, como a de processadores próprios para dispositivos de entrada, o surgimento de técnicas como *cache* e *interleave* e *pipelines,* que tinha a função de paralelizar as instruções.

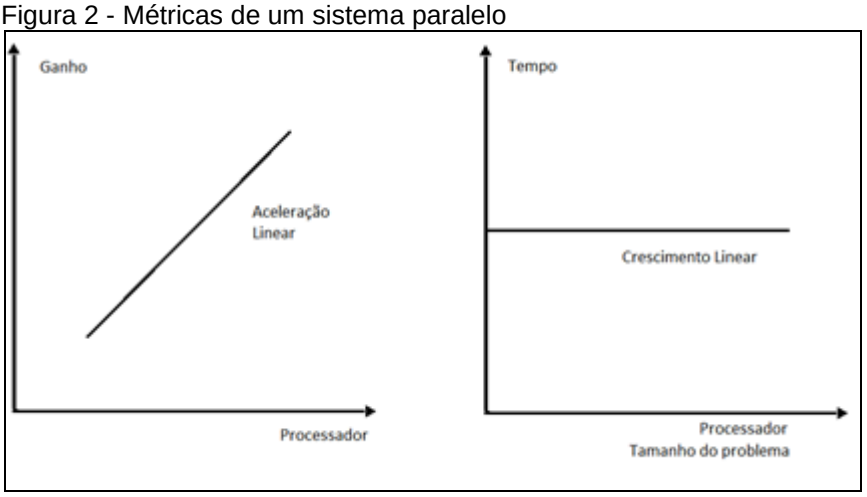
A Programação Paralela sendo ela uma parte de HPC, se tornou muito importante para a tecnologia, especialmente se considerar os vários tipos de programação oriundos das diferentes arquiteturas paralelas (BACHIEGA et al., 2017). Ela se dá a partir da quebra de um problema em partes independentes, para cada elemento de processamento possa executar sua parte do algoritmo simultaneamente com os outros. Os elementos podem ser heterogêneos e incluem recursos como: uma ou várias máquinas com um ou vários processadores, hardware

especializado ou uma combinação dos modelos citados (SHENG et al., 2012, tradução nossa).

Carvalho (2007) classifica os tipos de programação em três, sendo a primeira a programação sequencial, que tem como característica a execução de inúmeras tarefas uma em seguida à outra, para que a próxima instrução do programa possa ser executada, a anterior precisa ser finalizada, já a segunda é a chamada programação concorrente, que ao contrário da primeira, executa várias tarefas sem que as antecessoras tenham sido finalizadas, e a terceira, tem como característica a capacidade de executar várias tarefas simultâneas.

Segundo Meyer (2013), é possível medir a eficiência de um sistema paralelo de duas formas, sendo uma delas a aceleração linear, que não leva em conta o tamanho do sistema, ou seja, caso o hardware dobrar, o esperado é que o tempo de execução caia pela metade, e a outra forma é crescimento linear, que tem como métrica o ganho, que se dá por meio da razão do tempo de execução de uma mesma tarefa de um a n processadores, conforme a figura 2 demonstra.

Figura 2 - Métricas de um sistema paralelo



Fonte: Meyer (2013).

O desenvolvimento de programas paralelos costuma ser mais difícil que sequenciais, pois a concorrência entre processos introduz várias novas classes de possíveis erros de software. A comunicação e sincronização entre diferentes tarefas são tipicamente um dos maiores obstáculos para se conseguir uma boa performance (SHENG et al., 2012, tradução nossa).

2.2 SISTEMAS DISTRIBUÍDOS

Substituindo a arquitetura paralela, surgem os sistemas distribuídos, por conta do seu custo menor, maior flexibilidade e escalabilidade (JARDIM; SÁ, 2010). A computação distribuída tem sido muito usada para execução de tarefas com uso intensivo de dados, extensas infraestruturas de *hardware* e serviços distribuídas e poderosas, capazes de processar milhões dessas tarefas estão disponíveis em todo o mundo e estão sendo usadas pela indústria para agilizar seu processamento de dados pesado (MESTRE, 2018).

Um sistema distribuído é uma coleção de entidades independentes que cooperam para resolver um problema que não pode ser resolvido individualmente (KSHEMKALYANI; SINGHAL, 2008, tradução nossa). De acordo com Piovesan (2017), esse sistema deve ficar sempre ativo, mesmo que algumas máquinas fiquem fora de funcionamento, sem que os usuários notem a perda desses nós, ou o retorno deles.

Computação distribuída, também é conhecida como metacomputação, consegue combinar vários computadores em diferentes lugares do mundo através da Internet. Quando esse tipo de técnica é utilizado para fins científicos, ela é classificada como grade computacional, ou *Grid Computing*, já quando é utilizada para fins comerciais é chamado de Entropia (JARDIM; SÁ, 2010).

Problemas na rede isolam os computadores conectados a ela, mesmo assim isso não é sinônimo de uma parada completa do sistema. O que acontece na verdade, é que as aplicações em execução talvez nem percebam a falha na rede ou se ela está lenta. A queda de um computador ou erro inesperado em algum programa do sistema, não é imediatamente notada pelo os outros elementos que se comunicam com ele. Um elemento do sistema pode vir a falhar independentemente, sem afetar o funcionamento do sistema (COULOURIS; DOLLIMORE; KINDBERG, 2013).

2.3 CLUSTER

Um *cluster* pode ser do tipo paralelo ou distribuído, e é formado por vários computadores independentes e interconectados trabalhando juntos na realização de tarefas, como se fossem um único recurso de computação integrado (SILVA; BUYYA, 1999, tradução nossa).

Cada máquina que compõem o *cluster* é chamada de nó. A memória de um nó ainda é compartilhada entre seus processadores. Porém, ao conectar vários nós na mesma rede, a ideia de memória “global” deixa de existir. Por isso, o fluxo de informações entre os vários nós, precisa ser realizada através de troca de mensagens ou através de um modelo de memória distribuída (ALLES, 2018).

2.3.1 Cluster de Alta Disponibilidade

O cluster de alta disponibilidade (HA) é formado por um grupo de máquinas organizadas de maneira que, mesmo que um ou mais serviços falham em uma das máquinas, esses ainda serão acessíveis por outro computador (LI. et al, 2013 tradução nossa). Isso é alcançado, com a utilização de equipamentos, que em conjunto, trabalham para manter os serviços sempre ativos, além de também replicar os dados (THOMÉ; HENTGES; GRIEBLER, 2013).

Segundo Morimoto (2003 apud Eduardo, 2013), a principal função deste tipo de *cluster* é ser tolerante a falhas, e o mecanismo que permite funciona da seguinte forma: dois computadores ficam ligados em rede, onde um deles executa a tarefa enquanto o outro fica apenas monitorando e se atualizando com os dados do primeiro, como se fosse uma cópia exata, para que se o computador que está executando a tarefa parar de funcionar o segundo computador assume a execução da tarefa no lugar do primeiro, esse tipo de *cluster* é usado em servidores *web* ou banco de dados em *intranets*.

2.3.2 Cluster de Balanceamento de Carga

O objetivo principal do balanceamento de carga é para otimizar a utilização de recursos, manter o custo do data center e máquina virtual, maximizar o rendimento, reduzir o tempo de resposta e evitar sobrecarga (NARALE; BUTEY, 2018).

Em um *cluster* de balanceamento de carga, os processos são distribuídos igualmente entre as máquinas que compõem o *cluster*, o objetivo é fazer com que cada computador receba e processe uma requisição, e não, obrigatoriamente, que dívida com outras máquinas. Pode-se imaginar, como exemplo, um site que recebe em média mil visitas por segundo, e que para atender essa demanda criou-se um

cluster com 20 máquinas, como o foco é o balanceamento de carga as mil requisições serão distribuídas de maneira uniforme entre as 20 máquinas, fazendo com que cada uma receba e atenda, em média, 50 requisições por segundo (EDUARDO, 2013).

Esse tipo de *cluster* compartilha os dados dos processos entre os vários nós, fazendo com que todos possam rodar os mesmos processos, desta forma se um dos nós tiver algum tipo de problema, a carga dele será distribuída aos demais nós (BIAZOTTO, 2011). O balanceamento de carga pode ser usado em vários tipos de aplicações, porem o seu uso mais comum é na internet, já que esse tipo de solução tem uma maior tolerância ao aumento de requisições (EDUARDO, 2013).

2.3.3 Cluster de Alto Desempenho

Computação de alto desempenho, remete ao uso de um agrupamento computacional, com grande poder de processamento, para lidar com grandes volumes de dados, e que necessitam de muito esforço computacional (DOS SANTOS, 2018).

O *cluster* de alto desempenho é um tipo de sistema utilizando no processamento paralelo ou distribuído, e é construído com um conjunto de máquinas interconectadas, trabalhando como se fossem uma única máquina (PITANGA, 2008). Compreender as características e requisitos das aplicações que são executados em um HPC é um componente importante no design, configuração, e otimização de máquinas atuais e futuras. Os arquitetos de hardware e administradores de sistemas, por exemplo, podem usar esta informação para ajustar os parâmetros da máquina e bibliotecas para atender a um orçamento de energia específico, melhorar o desempenho da aplicação, ou maximizar o rendimento. Ao mesmo tempo, os responsáveis por futuras aquisições de hardware podem fazer decisões informadas sobre uma gama de opções disponíveis com respeito ao desempenho e confiabilidade do processador, memória e componentes de rede. O equilíbrio certo depende do conjunto de aplicativos que irá executar em tal sistema. Quanto mais se entender as características e exigências das aplicações, melhores serão as decisões que se pode tomar para fornecer o melhor desempenho dentro um determinado orçamento de capital ou poder (LEÓN, et al., 2016).

2.3.4 Bibliotecas de comunicação

Para que seja possível a comunicação entre os nós do *cluster*, são usadas bibliotecas, sendo elas responsáveis pela interação e transmissão das informações entre os nós. Existem algumas disponíveis para o uso, e cada uma com sua própria forma de comunicação, sendo que duas se destacam dela a *Parallel Virtual Machine* (PVM) e a *Massage Passing Interface* (MPI) (PRADO; SILVA, 2010).

2.3.4.1 PVM e MPI

PVM cria emulações de computação paralela heterogênea em computadores ligados em rede, podendo-se trabalhar com várias arquiteturas. Essa biblioteca monta uma maquina virtual com n processadores para usa-la no envio e recebimento de tarefas e utilizando rotinas simples (BACELLAR, 2010). O PVM oferece um modelo simples de interfaces e que pode ser implementado sem grandes problemas, para se realizar a execução e finalização de tarefas na rede, além de sincroniza-las. Processos em execução podem inicializar e encerrar outros processos, o PVM também possui características que fazem dele, tolerante a falhas (FERNÁNDEZ, 2009).

O MPI foi desenvolvido com o propósito de resolver o problema de portabilidade em relação as diferentes plataformas paralelas e suas características peculiares (FERNÁNDEZ, 2009). Segundo Eller Junior (2013), essa portabilidade mencionada se dá por conta da capacidade de execução de códigos de envio de mensagem em diferentes plataformas e sistemas heterogêneos. A biblioteca trabalha com vários tipos de arquiteturas computacionais disponibilizando interfaces de programação em C, C++ e Fortran (BACELLAR, 2010). O MPI tem como base o uso de tarefas, sendo que cada tarefa pode ser *threads* ou processo, onde cada núcleo do processador pode receber uma tarefa. Cada processador executa uma cópia do código fonte, dessa forma cada variável declarada pertence apenas ao processador em questão (MORENO; LIMA; DIAS, 2016). Nos *clusters,* os dados são distribuídos e executados entre os vários nós por meio da rede. Existem algumas implementações de código aberto do MPI como o OpenMPI e *Message Passing Interface over Chameleon* (MPICH) (DE OLIVEIRA, 2019).

2.3.4.1.1 Protocolo SSH

O *Secure Shell* (SSH), é utilizado pelas bibliotecas MPI e PVM na para realizarem a comunicação entre máquinas (AMARAL, 2006). Esse protocolo é utilizado para que não haja a necessidade de senha entre os nós quando e conexão entre eles for estabelecida (ROSTIROLLA, 2014).

O SSH é um pacote de software que permite a administração segura do sistema e a transferência de arquivos em redes inseguras. Ele é usado em quase todos os data centers, em todas as grandes empresas. Ele usa criptografia para proteger a conexão entre um cliente e um servidor. Todas as autenticações de usuário, comandos, saída e transferências de arquivos são criptografadas para evitar ataques na rede (SSH COMMUNICATIONS SECURITY, 2019).

O protocolo funciona no modelo cliente-servidor, o que significa que a conexão é feita pelo cliente SSH conectado ao servidor. O cliente conduz o processo de configuração da conexão e usa a criptografia de chave pública para verificar a identidade do servidor. Após a fase de configuração, o protocolo usa criptografia simétrica e algoritmos de *hashing* para garantir a privacidade e a integridade dos dados trocados entre o cliente e o servidor (SSH COMMUNICATIONS SECURITY, 2019).

3 VIRTUALIZAÇÃO

Segundo Bachiega (et al. 2018, tradução nossa), virtualização é uma tecnologia em expansão em todas as áreas de pesquisa e desenvolvimento. Tem sido amplamente aceita por causa de características como elasticidade e flexibilidade na entrega de recursos sob demanda. Archer et al. (2010, tradução nossa) cita que virtualização pode reduzir os custos de várias maneiras, como custos de hardware, software e gerenciamento. Várias áreas nas quais a virtualização pode ser usada incluem, consolidação de servidor e contenção de servidor, ambientes flexíveis de desenvolvimento e teste, por conta da facilidade e a flexibilidade de criar e reconfigurar sistemas operacionais convidados, conhecidos como máquinas virtuais do inglês *Virtual Machines* (VM), fazendo com que os ambientes de desenvolvimento e teste obtenham benefícios significativos da virtualização.

A virtualização é um método que torna possível dividir um sistema operacional em vários outros, as chamadas máquinas virtuais, cada uma destas máquinas possui um ambiente completo bastante similar ao de uma máquina física, podendo ter além de um sistema operacional próprio, possuir serviços de rede e aplicativos, também é possível conectar elas por meio de interfaces de redes. Uma das vantagens obtidas com a virtualização, é a possibilidade de se hospedar várias máquinas virtuais diferentes em uma única máquina física, cada uma com o seu próprio sistema operacional (CARISSIMI, 2008).

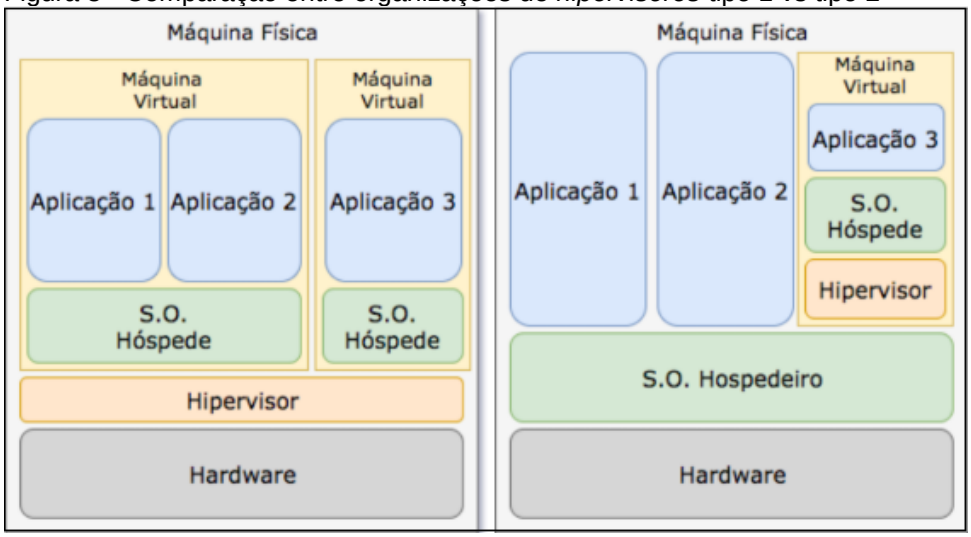
Pode ser utilizada como solução para o problema de se ter vários servidores dedicados, cada um a um único serviço, com o propósito de que se um servidor parar, apenas um dos serviços ficará indisponível, sem que afete os demais. A solução consiste em se usar várias máquinas virtuais potencialmente com sistemas operacionais diferentes em um único servidor, onde cada máquina seria responsável por um serviço, mantendo assim a mesma ideia citada anteriormente (TANENBAUM; BOS, 2015, tradução nossa).

3.1 VIRTUALIZAÇÃO EM NÍVEL DE HARDWARE

Esse tipo de virtualização proporciona uma abstração completa do sistema físico permitindo a execução de sistemas operacionais. Para tanto, se faz

necessário a adição de uma camada de software ao sistema operacional, sendo esta responsável por realizar a detecção e gerenciamento dos recursos disponíveis. A esta camada se dá o nome de Monitor de Máquina Virtual, do inglês *virtual machine monitor* (VMM), ou *hipervisor*. Por meio dele, os sistemas virtualizados tem acesso ao hardware físico, mantendo o mesmo isolado (ALLES, 2018).

Figura 3 - Comparação entre organizações de *hipervisores* tipo 1 vs tipo 2



Fonte: Alles (2018).

Existem basicamente duas formas de implementação de VMMs ilustrados pela figura 3, os de tipo 1, que ficam próximo do hardware da máquina física e as maquinas virtuais rodam sobre ele, *Xen* e *VMware ESX Server* são dois exemplos deste tipo, e os de tipo 2, que executam sobre um sistema operacional nativo se comportando como um processo do sistema, alguns exemplos deste tipo são *Vmware Player*, *VirtualBox* e *MS VirtualPC* 2007 (CARISSIMI, 2009).

3.2 VIRTUALIZAÇÃO COMPLETA E PARAVIRTUALIZAÇÃO

Uma das dificuldades envolvendo máquinas virtuais, é realizar a implementação de tal forma, que mesmo existindo o compartilhamento de recursos físicos entre os ambientes virtuais, um não interfira no outro. O processador precisa definir de que forma as instruções privilegiadas dos sistemas operacionais convidados serão executados, já que só o sistema hospedeiro tem esses privilégios. (ALLES, 2018). Os autores Popek e Goldberg (1974, apud CARISSIMI, 2009), concluíram que instruções desse tipo precisam ser tratadas pelo *hypervisor.* Existem duas maneiras de tratar essas instruções, sendo elas a virtualização completa, que

traduz as instruções recebidas pelo sistema convidado para binário, e são interpretadas pelo *hypervisor*, sendo ele responsável por decidir se uma instrução pode ou não ser executada diretamente no processador, ou se deve receber algum tipo de tratamento antes da execução.

Esse método não necessita de qualquer alteração no sistema operacional hóspede, também não precisa ter o conhecimento do *hypervisor*. Porém, por ser necessários que toda instrução passe pelo *hypervisor* para ser testado, o método em questão possui um alto custo de execução. Para tentar reduzir o problema, existe a paravirtualização, que ao contrario da virtualização completa, o sistema hóspede possui o conhecimento da existência de um *hypervisor*, realizando a comunicação diretamente com ele, dessa forma, as instruções privilegiadas são substituídas por chamadas ao *hypervisor*, que intercepta e faze os devidos tratamentos. Com isso tem-se um custo menor do que a virtualização completa, pois apenas serão interceptadas pelo *hypervisor* as chamadas que possuem privilégios (POPEK; GOLDBERG, 1974 apud CARISSIMI, 2009).

Para que seja possível utilizar a paravirtualização são necessárias modificações no *kernel* do sistema operacional hóspede, e por conta disso, não é possível realizar essa técnica em qualquer sistema operacional. Outra forma de se ter o desempenho do sistema aumentado, se dá por meio de extensões de hardware que possuem suporte a virtualização. (LEE, 2014).

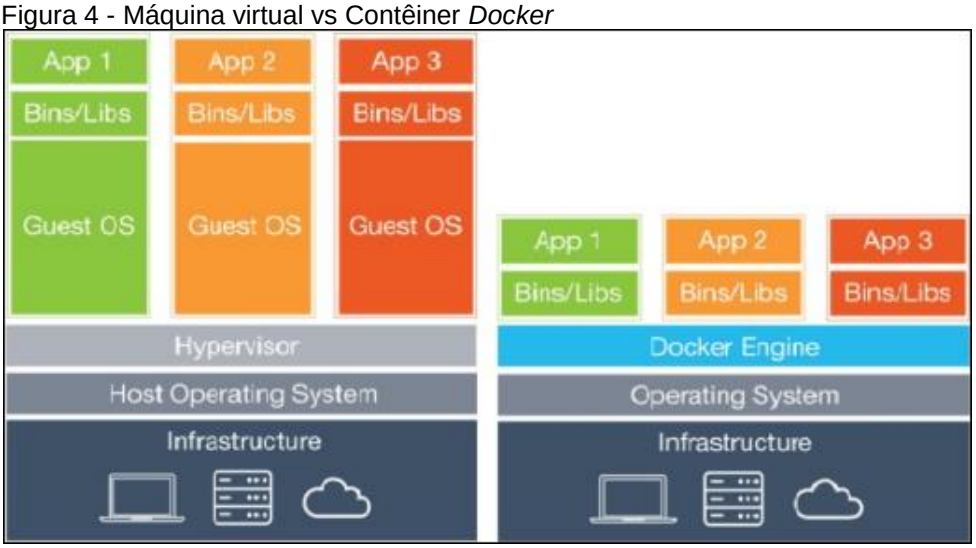
3.3 VIRTUALIZAÇÃO A NÍVEL DE SISTEMA OPERACIONAL

Esse tipo de virtualização tem como principal objetivo o isolamento dos recursos da aplicação. Os recursos virtualizados são definidos pela implantação do *kernel* do sistema operacional, porém, costumam ser árvores de processo, usuários, *interfaces* de rede e sistemas de arquivos. Com isso, uma aplicação executando dentro de um ambiente como este, tem a impressão de estar em uma máquina dedicada e não saberá que os recursos estão sendo compartilhados. Uma característica dessa virtualização é que as aplicações utilizam o mesmo *kernel* (ALLES, 2018).

3.4 CONTÊINER

Os contêineres são uma nova forma de virtualização que vem se destacando à medida que as grandes empresas de tecnologia estão dando mais suporte a elas (BACHIEGA et al., 2018, tradução nossa). De acordo com Oliveira (2017), contêineres concedem uma organização de funcionamento tão leve, porque não precisam de um *hipervisor,* ou *Kernel* próprio, pois são executados como um processo do sistema e compartilham com o *Kernel* da máquina hospedeira todos as bibliotecas, binários e arquivos necessários para o seu funcionamento, isso permite a execução de vários contêineres em uma única estação de trabalho. Na figura 4 pode-se observar as diferenças entre um contêiner e uma máquina virtual, que na imagem em questão utiliza o gerenciador de contêiner Docker.

Figura 4 - Máquina virtual vs Contêiner *Docker*



Fonte: Oliveira (2017).

Máquinas virtuais e contêineres tem funções bem parecidas no que diz respeito ao isolamento e alocação de recursos, porém funciona de maneira um pouco diferente, permitindo aos contêineres uma maior portabilidade e eficiência (CORRÊA, 2016). Também é possível usar um tipo de contêiner que possua em suas características, a opção de agregação ou *clustering,* permitindo a inserção de um conjunto de contêineres, por meio do compartilhamento de recursos (OLIVEIRA, 2017). Segundo Alles (2018), se tratando de implementação, existem três ferramentas necessárias em um *kernel Linux* que possibilitam a criação e uso de contêineres, sendo elas o *chroot, cgroups e namespaces.*

Chroot é um comando que modifica o diretório principal do processo em execução junto com seus filhos. Se um processo for executado dentro desse ambiente, o mesmo não possui acesso a arquivos e diretórios externos (CHROOT, 2018).

Cgroups são usados para realizar a alocação de recursos, como memória do sistema, tempo de CPU e largura banda de rede. Por meio dele é possível realizar o monitoramento, impedir o acesso de certos Cgoups a recursos e reconfigurar automaticamente. Administradores de sistema podem utilizar esse recurso pois ganham controle sobre a alocação, prioridades, negação gerenciamento e recursos de monitoramento (PRPIC; LANDMANN; SILAS, 2018).

Namespace é uma abstração que tem como função dar a impressão aos processos dentro do *namespace*, de que eles possuem sua instancia isolada. Alterações nesse recurso são visíveis para os outros processos que são membros do *namespace*, porem os demais não tem acesso a essa informação (NAMESPACES, 2019).

3.4.1 Docker

O Docker aproveitou conceitos de computação existentes em torno de contêineres e, especificamente do Linux, conhecidos como cgroups e namespaces. A tecnologia do Docker é única porque se concentra nos requisitos de desenvolvedores e operadores de sistemas para separar dependências de aplicativos da infraestrutura. O sucesso no mundo do Linux gerou uma parceria com a Microsoft que trouxe contêineres do Docker e suas funcionalidades para o Windows Server (às vezes chamados de contêineres Windows Docker). Tecnologia disponível no Docker e em seu projeto de código aberto, o Moby[1] (Uma estrutura para montar contêineres especializados), foi aproveitado por todos os principais fornecedores de data center e provedores de nuvem. Muitos desses provedores estão aproveitando o Docker para suas ofertas IaaS nativas de contêiner. Além disso, as principais estruturas sem servidor de código aberto utilizam a tecnologia de contêineres Docker (DOCKER, 2019, tradução nossa).

[1] https://github.com/docker/engine

Os componentes e ferramentas do Projeto Moby são inicialmente os componentes de código aberto que o Docker e a comunidade criaram para o Projeto Docker. Ele é destinado a ser um local para que os colaboradores trabalhem com código-fonte aberto, corrijam bugs e tornem o código mais útil.

O Docker permitiu que os desenvolvedores utilizassem contêineres ao trabalhar em qualquer aplicativo, seja um novo serviço ou um aplicativo existente. Os contêineres encapsulam o código, as configurações e as dependências em um pacote isolado, potencialmente tornando o aplicativo mais seguro e portátil (DOCKER, 2019, tradução nossa).

A plataforma Docker é baseada em tecnologias de código aberto usado por milhões de desenvolvedores e profissionais de TI em todo o mundo, o Docker inclui a biblioteca e o ecossistema de contêineres mais importante do mundo, com mais de 100.000 imagens de grandes fornecedores de software, projetos de código aberto e da comunidade (DOCKER, 2019, tradução nossa).

Algo importante de se observar na instalação, é de que há duas versões, sendo elas a *Docker Community Edition* (Docker CE) e a *Docker Enterprise Edition* (Dokcer EE), ambos possuem versões para Windows, Linux e MacOS. O Docker CE é recomendado para desenvolvedores e equipes de menor porte que desejem utilizar o Docker e testar softwares baseados em contêiner, já a versão Docker EE é recomentado para sistemas de produção e equipes maiores, que constroem, enviam e executam aplicações críticas (MONTEIRO, ALMEIDA, 2017).

3.4.1.1 Docker CE

É uma plataforma gratuita para infraestrutura aberta que atende às necessidades das comunidades DEV e OPS. Ele é ótimo para desenvolvedores e equipes começarem a criar aplicativos contêineres (DOCKER, 2017, tradução nossa).

3.4.1.2 Docker EE

Com o ele é possível gerenciar cargas de trabalho de contêiner de maneira flexível no Windows, Linux, site ou nuvem. Ele possui gerenciamento de imagem privado, políticas de assinatura de imagem integrada e gerenciamento de

cluster com suporte para os orquestradores Kubernetes e Swarm. Ele permite que você implemente políticas *Role Based Access Control* (RBAC), baseadas em nó, políticas de promoção de imagens, espelhamento de imagens e análise de imagens em busca de vulnerabilidades. Ele também tem suporte a Service *Level Agreement* (SLA) definidos e ciclos de manutenção estendidos para patches por até 24 meses (DOCKER, 2019).

O Docker mantém repositórios visíveis publicamente para o código CE, bem como repositórios privados para o código EE. A automação é usada para manter as ramificações entre CE e EE em sincronia, assim como os recursos e correções são mesclados nas várias ramificações nos repositórios do CE (upstream), os repositórios e ramificações EE correspondentes são mantidos em sincronia (Docker, 2019).

3.4.1.2.1 Docker Swarm

É um orquestrador de contêineres, podendo administrar recursos de redes de computadores, de maneira que cada um deles é uma plataforma de execução Docker com recursos independentes. Quando o modo Swarm é ativado, esse conjunto de máquinas se tornam um *cluster* nativo Docker, tornando possível a inicialização, monitoramento e orquestração de contêineres (ALVES, 2017).

3.4.1.2.2 Kubernates

É uma infraestrutura de código aberto e nível de produção para a implantação, dimensionamento, gerenciamento e composição de contêineres de aplicativos em clusters, inspirados em trabalhos anteriores do Google. Ele é mais do que apenas um “orquestrador de contêineres”. O objetivo é eliminar a carga de gerenciar a infraestrutura computacional física, virtual e de rede, e permitir que os operadores e desenvolvedores de aplicativos se concentrem totalmente em primitivos centralizados para operação de autoatendimento. Também fornece uma base estável e portátil (uma plataforma) para a criação de fluxos de trabalho personalizados e automação de alto nível. O Kubernetes é voltado principalmente para aplicativos compostos de vários contêineres. Por isso, agrupa contêineres

usando *pods* e rótulos em formações fortemente acopladas e fracamente acopladas para fácil gerenciamento e descoberta (KUBERNETES... 2019, tradução nossa).

4 TRABALHOS CORRELATOS

Para a realização deste trabalho, foram feitas pesquisas em trabalhos relacionados ao tema, no meio nacional e internacional, para se ter uma ideia dos projetos semelhantes, usando-os como base para deste projeto.

4.1 CLUSTER DE ALTO DESEMPENHO: AGREGANDO VALOR À INFRA-ESTRUTURA COMPUTACIONAL DAS UNIVERSIDADES

O trabalho desenvolvido por Fábio Vieira do Amaral e apresentado no ano de 2006 ao Instituto de Pesquisas Tecnológicas - IPT para a obtenção do título de mestre em Engenharia da Computação – Ênfase em Engenharia de Software, tem por objetivo o desenvolvimento de um *cluster* de alto desempenho utilizando pacotes de software Linux e a infraestrutura computacional já existente dos laboratórios didáticos de uma universidade (AMARAL, 2006).

O segundo objetivo deste projeto é elucidar baseado nos resultados, as aplicabilidades das ferramentas de suporte e relacionar com os tempos de performance das aplicações em suas plataformas específicas, monoprocessadas e no *cluster* de alto desempenho (AMARAL, 2006).

Na presente pesquisa de Amaral (2006), é possível verificar que apontou dois tipos de *clusters* (Tipo I: *Beowulf* e Tipo II: *OpenMoxis*), porém deu mais destaque ao tipo *Beowulf* porque as aplicações usadas nos experimentos foram arrumadas para essa amostra, sendo que se utilizou a distribuição Debian do *Linux*. Os experimentos da pesquisa de Mestrado de Amaral (2006) foram realizados com o *software* meteorológico Regional *Atmospheric Modeling System* - RAMS, este disponibilizado pela organização Somar Meteorologia (AMARAL, 2006).

O RAMS ainda como experimento foi inserido em um *notebook*, com a finalidade de se integrar ao projeto flexibilidade, podendo assim, executar demonstrações em campus diferentes da instituição em estudo.

No início do projeto, este tinha recursos escassos até alcançar sua forma superior, no momento em que foi exposto em outubro de 2004 na VI Exposição e Congresso Internacional de Telemedicina, Telesaúde, Educação e Treinamento a Distância, Tecnologias e Sistemas de Informação, realizado nos dias 27 a 29 do corrente ano, no qual foi exibida a oitava versão do *cluster* com o suporte de um

notebook, onde foi capaz de implantar a arquitetura de *cluster* de processamento paralelo em laboratório, sem nenhuma configuração preliminar, somente impulsionando as máquinas da seção de rede na condição de *boot* remoto, sem a necessidade de inserir algum fragmento de código no computador participante, ou seja, sem afetar a instalação que já existia (AMARAL, 2006).

Como resultado do projeto foi possível aumentar as opções de utilização dos laboratórios ociosos da universidade para além dos fins didáticos. Foi possível alcançar níveis superiores de processamento de uma empresa, como a Somar Meteorologia, que se utiliza de processamento em *cluster* para prestação de serviço. Agregou a universidade uma ferramenta computacional poderosa, possuindo um atributo importante por conta da sua mobilidade permitida pelo desenvolvimento (AMARAL, 2006).

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

O Trabalho de Conclusão de Curso (TCC) desenvolvido por Régis Matheus Silveira Maia e apresentado no ano de 2016 aos professores de uma Banca Examinadora do Curso de Tecnologia em Redes de Computadores da Universidade Federal do Ceará, tem como objetivo o desenvolvimento e avaliação de uma ferramenta para o auxílio da criação de *clusters* virtuais com o intuito de proporcionar aos estudantes de programação paralela e distribuída o desenvolvimento de *software* em um ambiente próximo ao de um *cluster* físico (MAIA, 2016).

A presente pesquisa trata-se de um estudo de caso, no qual foi realizada uma análise de desempenho do tempo de criação *clusters* virtuais na ferramenta que foi desenvolvida por Maia (2016). Para tal, foram usados 6 computadores, referentes ao laboratório de informática da universidade em estudo, no qual foram criados uma infraestrutura de nuvem privada que foi usado para o desenvolvimento dos *clusters* virtuais. Dessa forma, recursos como CPU e *Random Access Memory* (RAM) foram incorporados ampliando a capacidade de desenvolvimento do número de *clusters* virtuais. (MAIA, 2016).

Em uma das seis máquinas já citadas, foi instalado o software responsável pela criação do ambiente de nuvem, *OpenNebula*, e as demais ficaram

como *hosts* da nuvem. Foi optado por se utilizar contêineres por ser um modo de virtualização mais leve que o tradicional, porém segundo Maia (2016) o software *OpenNebula* não possui suporte nativo para esse tipo de virtualização, por essa razão utilizou-se um *plugin* chamado *OneDock* para adicionar tal suporte, e que trabalha em harmonia com o Docker, além de permitir a criação de VM utilizando contêineres. A interface *web* foi desenvolvida em PHP, por essa razão, foi vital a instalação do servidor *Apache* no mesmo computador em que se encontrava o *OpenNebula*, para que este pudesse hospedar a ferramenta (MAIA, 2016).

A partir de uma ferramenta com uma interface gráfica, após efetuar o *login*, o aluno precisa informar apenas a quantidade de nós, de CPU's e memória RAM desejada para que o *cluster* seja criado automaticamente. Por meio da solicitação do usuário, vários *scripts* são invocados um chamando o outro e verificando se os requisitos informados estão disponíveis caso estejam a criação do *cluster* virtual é iniciada, caso não estejam o usuário será informado para que diminua a quantidade de recursos solicitada (MAIA, 2016).

O estudo de caso desenvolvido no TCC de Maia (2016) serviu para validar a criação *clusters* virtuais, no qual para a análise foi usado a ferramenta *Apache JMeter* que possibilita desenvolver *threads*, onde cada uma delas representa um usuário, desse modo, buscando localizar qual o tempo despendido para o criação do *cluster*, tal como, avaliar a quantidade de usuários que a ferramenta tem a capacidade de atender. Cada *cluster* testado era constituído por dois computadores virtuais sendo um *slave* e outro *nó master*, sendo que cada máquina tinha uma CPU e 512 *Megabytes* (MB) de memória RAM. Conforme os resultados que foram adquiridos na presente pesquisa, estes apresentam um grau de confiança de 95%.

Como conclusão do trabalho foi observado na infraestrutura uma limitação com relação ao *Docker,* que em alguns momentos travou durante testes com muitos usuários, sendo necessário a reinicialização do controlador do *OpenNebula*, porém analisando os resultados obtidos com 12 usuários simultâneos acessando a ferramenta, pode-se considerar um tempo muito bom, apontando que tanto a ferramenta quanto a infraestrutura, responderam de forma eficaz, podendo ser utilizada nas aulas práticas das disciplinas de programação paralela e distribuída (MAIA, 2016).

4.3 APLICAÇÃO DE CLUSTER BEOWULF EM INSTITUIÇÕES DE ENSINO

Monografia desenvolvida por Claudio Lima do Prado e João Messias Alves da Silva, apresentado no ano de 2010 na Faculdade de Tecnologia de Guaratinguetá, para a graduação no Curso Superior de Tecnologia em informática, ênfase em Redes de computadores tem como objetivo explorar o tema proposto considerando dois atores, sendo que um deles é o *Cluster Beowulf* como uma opção de *cluster* de alto desempenho e baixo custo e as instituições de ensino que são sementes de desenvolvimento cientifico (PRADO; SILVA, 2010).

A metodologia do trabalho consiste em pesquisas a livros, periódicos técnicos, monografias e artigos relacionados, realizou-se também uma pesquisa acadêmica em instituições de ensino da região aplicando-se um questionário, com o intuito de se conhecer o pensamento acerca da pesquisa científica e os atuais laboratórios de computação e pesquisa dessas instituições. Também é abordada uma proposta real de um laboratório de pesquisa com *cluster* de computadores, realizando um levantamento de requisitos de software e hardware, custo estimado e um roteiro de instalação (PRADO; SILVA, 2010).

Foi dado, para fins de nomenclatura o nome de *Cluster Skywulf* para o *cluster* sugerido, sendo ele do tipo *Beowulf.* A área total do laboratório proposto é de 80m^2 sendo divididas em duas salas por uma “parede falsa” ou vidrada separada por uma porta de controle básico. A sala do *cluster* é onde se abrigará o *Cluster Skywulf* e o administrador/suporte do *cluster,* já o outro espaço será destinado aos alunos, tendo oito computadores, duas mesas de leitura, um quadro branco e uma mesa para a montagem de uma *Datashow* ou suporte ao professor (PRADO; SILVA, 2010).

Os componentes que compõem o *Cluster Skywulf* são, um *Switch Cluster* padrão *Gigabit-Ethernet* de alta confiabilidade e performance, um *Switch* Pesquisa padrão *Fast-Ethernet* usado para realizar a interligação dos computadores da sala de pesquisa, um servidor de pequeno porte *Linux* para ser o *firewall,* 8 computadores *Dual Core*, compatível ou superior *Pentium* 4 com 1GB de memória RAM e *Hard Disk* (HD) 160GB, para o administrador/suporte 1 computador *Dual Core*, compatível ou superior *Pentium* 4 com 2 GB de memória RAM e HD 320GB, Computador comum, desejável 512 MB de RAM ou superior para ser o nó mestre do *cluster*, duas placas rede *Gigabit off board,* monitor, mouse e teclado, para servir de

nós escravos 5 computadores comuns, desejável 512 MB ou superior, placa rede *Gigabit off board*, cabos de rede, conectores RJ-45, *No-break* e Ar-Condicionado para garantir a temperatura da sala de acordo com a norma do cabeamento estruturado (PRADO; SILVA, 2010).

Para a realização de testes o *Cluster Skywulf* foi feito a montagem real dele, e pode-se constatar é a tendência pela estabilidade, pois os menores indicadores de *megaflops* estavam chegando perto dos maiores. O *cluster* obteve 1,2 *Gigaflops* e média geral de 1 *Gigaflops* de processamento viabilizando a sua implantação (PRADO; SILVA, 2010).

4.4 PERFORMANCE OF A VIRTUAL CLUSTER IN A GENERAL-PURPOSE TEACHING LABORATORY

O artigo elaborado em 2011 por *Eric Johnson, Patrick Garrity, Timothy Yates* e *Richard A. Brown* para a IEEE *Internacional Conference on Cluster Computing* tem como objetivo a criação de um *cluster* virtual do tipo *Beowulf* buscando aproveitar o poder computacional das máquinas em estado ocioso e aproveitar os investimentos existentes em *hardware* para a faculdade, local do estudo, além de proporcionar aos alunos a oportunidade de ter experiências práticas com sistema operacional, redes e administração do sistema, e, podendo ainda, com a presença de um virtual *cluster* no local para poder ajudar a incentivar os alunos e professores a incorporar mais computação distribuída e *cluster* em cursos e projetos, apoiando assim os alunos a aprenderem os princípios de paralelismo e concorrência que eles precisarão em suas carreiras (JOHNSON et al., 2011, tradução nossa).

O *cluster MistRider* no *Saint Olaf College* é um desses *clusters* virtuais. O *hardware* físico consiste em uma sala de aula e laboratório de computadores com *hardware* e *host* sistema operacional gerenciado e mantido pelo Departamento de TI para apoiar cursos e projetos para o programa CS. Virtualização fornecida pelo *Linux Kernel-Based Virtual Machine* (KVM) que imita a divisão lógica de um laboratório dedicado e *cluster* dedicado.

Os estudantes gerentes do *cluster* possuem privilégios para lançar, destruir máquinas e gerenciar a rede virtual que conecta a eles. Embora esses alunos tenham plenos poderes para instalar e gerenciar imagens do sistema nessas VMs, eles não têm privilégios no laboratório físico, equipamentos ou sistemas

operacionais *host*. Este arranjo simplifica o gerenciamento do *cluster*, permite a atualização ou instalação de *software* e infraestrutura sem o envolvimento do departamento de TI, e fornece um sistema simples, porém é um sistema de segurança flexível. O *software* de *cluster* não precisa ser instalado nas máquinas do laboratório e vice-versa, o *cluster* executa uma variante do sistema operacional *Linux,* podendo executar o *Windows* ou outro sistema operacional. Além disso, caso os nós do *MistRider* serem acessados localmente com os usuários *host* da estação de trabalho, o nó fica inacessível ao *cluster*. O *MistRider* expande o potencial de ensino dos laboratórios sem impedir a plena realização de sua principal utilização (JOHNSON et al., 2011, tradução nossa).

Cada estação de trabalho de laboratório *MistRider* tem um *quad-core* de 2,4 GHz Processador *Intel*, 4 GB de memória e uma *Ethernet* de 1 Gb. Para uso neste estudo, 16 máquinas de laboratório foram removidas da rede do campus e conectado a um local isolado rede em um *switch Ethernet Cisco* WS-C3750 junto com o nó principal existente. O sistema operacional do laboratório utilizado foi o *Fedora* 9 (JOHNSON et al., 2011, tradução nossa).

Foram realizados alguns testes com essa configuração acima citada para a avaliação do *cluster* virtual *MistRider*, e os resultados apontaram que a virtualização com o KVM quase não afeta o desempenho da CPU e memorias dos nós, e o desempenho de rede foi afetado. Porem seguindo as tendências em direção ao suporte de microprocessadores para virtualização de hardware, a futura padronização de *Input-Output Memory Management Unit* (IOMMUs) em placas de rede eliminaria efetivamente a sobrecarga da virtualização completamente (JOHNSON et al., 2011, tradução nossa).

5 COMPUTAÇÃO PARALELA E DISTRIBUÍDA UTILIZANDO CONTÊINERES

O presente trabalho consiste na aplicação dos conceitos de computação paralela e distribuída na criação de um *cluster* de alto desempenho com a tecnologia de contêineres para ser utilizado na disciplina de computação paralela e distribuída. Para tornar isso possível foi necessário a escolha das ferramentas necessárias para realizar a criação e gerenciamento dos contêineres, após a escolha foi necessário a instalação e configuração em todas as máquinas que seriam utilizadas no projeto. Com a instalação do *cluster* é possível oportunizar o contato dos acadêmicos com o ambiente de uma forma mais rápida e prática, trabalhando com os conceitos apresentados em sala de aula.

5.1 METODOLOGIA

Foi feito um levantamento bibliográfico por meio de recursos disponibilizados pela UNESC, bem como adquiridos pelo próprio acadêmico realizando assim a elaboração do referencial teórico. Este levantamento foi realizado com acesso a bases de dados, tais como IEEE XPLORER, Google Scholar, entre outras. A pesquisa se concentrou no acesso no segundo semestre de 2018 e primeiro semestre de 2019. Esta pesquisa tem uma abordagem quantitativa, de natureza aplicada e experimental.

Através dos estudos realizados pelo acadêmico, pode-se se aprofundar mais nos conceitos de computação paralela e distribuída aprendidos em sala de aula, bem como o adquirir mais conhecimento sobre virtualização, em especial a baseada em contêineres. Após isso foi necessário buscar um meio de unir computação paralela e distribuída a tecnologia de virtualização já citada.

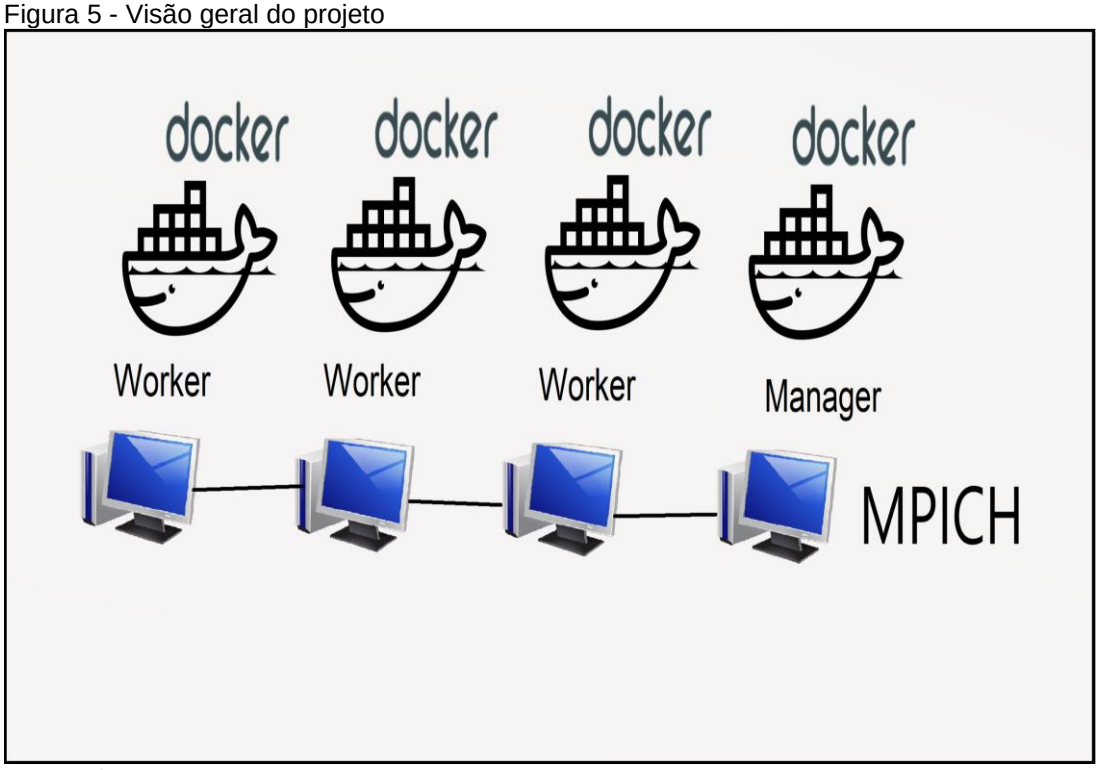
Ao longo da pesquisa foram encontradas algumas alternativas, porém a muitas destas com versões de software já desatualizados, e projetos utilizando outros softwares, então dificultando ou até restringindo o seu uso. Uma das soluções mais promissoras consiste em criar uma nuvem privada utilizando alguns computadores, utilizando máquinas virtuais baseadas em contêiner para o desenvolvimento do projeto e que será melhor elucidado no sub capitulo seguinte.

A solução adotada no projeto foi desenvolvida por Nikyle Nguyen e Doina Bein, por ser uma ferramenta bem completa e com *scripts* que ajudam a realizar a

instalação de forma automatizada, além de permitir a modificação dos *scripts* básicos, dando a possibilidade de customização da implementação. O artigo do projeto foi publicado na IEEE Xplore Digital Library no ano de 2017. E é possível encontra-lo no site GitHub[2] pois está em modo público, e utiliza a ferramenta Docker na criação do *cluster*.

A primeira etapa necessária para a implementação do projeto foi instalação do Docker em todas as máquinas que fariam parte do *cluster*, a figura 5, dá uma ideia do projeto. A implementação desse projeto foi realizada no laboratório de Computação da Universidade, em função da disciplina de Computação Paralela e Distribuída, ser ministrada neste local.

Figura 5 - Visão geral do projeto



Fonte: do Autor

Com a implantação do *cluster* pretende-se proporcionar um contato dos acadêmicos com a aplicação do *Cluster*, demonstrando a confiabilidade e desempenho dos projetos desenvolvidos para executarem neste ambiente, o processamento será distribuído entre os nós. Na máquina que foi escolhida para ser

[2] https://github.com/NLKNguyen/alpine-mpich

o nó principal, foi iniciado o modo Swarm[3] do Docker (que é o orquestrador de contêineres nativo da ferramenta), e adicionado outras três maquinas ao Swarm, após isto, foi realizado o download do projeto presente no GitHub, e através de alguns comandos foi realizado a compilação de arquivos *Dockerfile* para gerar as imagens Docker, e posteriormente inicializado o cluster. A seguir será descrito, como realizar os procedimentos e as ferramentas utilizadas no projeto.

5.1.1 Clusters Estudados

Com o objetivo de aprender na prática os desafios de se implementar um *cluster* físico*,* realizou-se uma busca por modelos disponíveis na Internet, tendo sido encontrado algumas opções. Dentre os vários modelos encontrados, foi escolhido um para ser implementado no laboratório, o mesmo foi escolhido pois, possuía a data mais recente dentre os pesquisados, e por isso, era o que tinha menos chances de possuir algum software desatualizado ou descontinuado. O artigo contava com um bom material, que descrevia a instalação passo-a-passo do *cluster*, além de algumas ferramentas opcionais. O modelo em questão é do tipo *Beowulf,* o manual da montagem encontra-se no site Viva o Linux[4], O artigo menciona que foi utilizado no projeto o sistema operacional Debian Wheezy, mas segundo o autor, ele pode ser adaptado para outras distribuições Linux.

Utilizou-se dois computadores do laboratório da universidade para tentar reproduzir a implementação. Entre as maiores dificuldades encontradas, a principal foi realizar as configurações do servidor SSH e de rede, além de existirem descontinuidade de alguns softwares, obrigando ao acadêmico buscar alternativas para finalizar a implementação. Dos softwares utilizados estão: *Network Time Protocol* (NTP), Servidor SSH, *Parallel Virtual File System* (PVFS), MPICH.

Um outro projeto estudado, já mencionado neste trabalho, utiliza a implementação de uma nuvem privada com a ferramenta OpenNebula, para permitir a criação de *clusters* virtuais Docker, automatizando o processo com uma ferramenta desenvolvida pelo próprio autor, e que tem como função de ser o intermediador entre o usuário e a ferramenta, permitindo a escolha da quantidade de

[3] O modo Swarm é um recurso utilizado pelo container Docker, onde ele fornece funcionalidades de orquestração de contêiner, tendo incluído o clustering nativo de hosts e realiza agendamento de cargas de trabalho

[4] https://www.vivaolinux.com.br/artigo/Montagem-de-Cluster?pagina=1

recursos do *cluster* virtual, isso dentro dos limites físicos disponíveis, dos equipamentos.

O projeto foi considerado uma boa opção a ser utilizado no presente trabalho, já que se tratava de uma implementação de um *cluster* do tipo *Beowulf* e utilizava a tecnologia de virtualização de contêineres para isso, além de contar com uma ferramenta que auxilia a criação dos *clusters*. Porém, foi encontrado uma aparente incompatibilidade entre a versão atual do Docker 18.09.6, e ONEDock[5], que é um projeto construído com o intuito de oferecer suporte ao OpenNebula na criação de contêineres Docker, forçando a utilização de uma versão mais antiga do Docker, dificultando a implementação do mesmo. Além do problema de compatibilidade, notou-se a falta de alguns scripts no projeto em questão, disponível no GitHub[6], trazendo uma dificuldade considerável na implementação.

Além dos softwares já citados, o projeto utiliza um servidor web para hospedar a ferramenta desenvolvida pelo autor, que no caso, realiza o papel de intermediário entre o usuário e o OpenNebula na criação dos *clusters*.

5.2 INSTALAÇÃO DOCKER

Foi utilizado o Docker CE na versão 18.09.6 para Ubuntu de arquitetura x86_64/amd64, por ser de código aberto e de uso gratuito. Os comandos mostrados a seguir, devem ser realizados em todos os nós.

Atualização dos locais de repositórios dos Ubuntu. Onde, é possível atualizar o apt do sistema operacional. Utilizando o seguinte comando:

```
$ sudo apt-get update
```

Instalação de pacotes para permitir que o apt use um repositório via HTTPS:

```
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
```

[5] https://github.com/indigo-dc/onedock
[6] https://github.com/matheusmaia43/virtualcluster

```
gnupg-agent \
software-properties-common
```

Adicionar chave GPG oficial do Docker:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Neste ponto é necessário, verificar a chave que possui a seguinte impressão digital 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, procurando os últimos 8 caracteres se então contidos na mesma.

```
$ sudo apt-key fingerprint 0EBFCD88
```

O próximo ponto a configurar, é definir a procura de repositórios estáveis.

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

Realizado o comando anterior, é necessário verificar se existem atualizações do pacote apt, pois no comando anterior, foi adicionado uma nova base no repositório Linux.

```
$ sudo apt-get update
```

5.2.1 Containerd

O *Containerd* foi projetado para ser usado pelo Docker e Kubernetes, bem como por qualquer outra plataforma de contêiner que queira abstrair o syscalls ou a funcionalidade específica do SO para executar contêineres tanto no Linux, Windows, Solaris ou qualquer outro sistema. Ele é responsável por funcionalidades como push, pull e gerenciamento de imagens. Além de possuir Application Programming Interface (APIs) ciclo de vida de um contêiner, bem como executar, gerenciar (CROSBY, 2017, tradução nossa).

É recomendável trabalhar sempre com a versão mais recente do Docker CE e *containerd*. O comando abaixo realiza a instalação final:

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
$ sudo docker run hello-world
```

Após o término da instalação do Docker CE, é importante verificar a execução das aplicações disponíveis neste ambiente, como por exemplo a imagem hello-world, que aparecem em quase todos os softwares com base de desenvolvimento.

5.2.2 Iniciando o Docker Swarm

Foi inicializado o modo Swarm do Docker na máquina escolhida para vir a ser o nó mestre, utilizando o comando *docker swarm init* tornando-a um nó gerenciador ou *Manager,* do modo Swarm.

```
sudo docker swarm init
```

Foi necessário copiar a saída resultante do comando acima, para executa-la no terminal de todas as máquinas que fariam parte do cluster, onde deve ser adicionado os trabalhadores ou *workers.* A saída citada se assemelha à mostrada logo abaixo.

```
docker swarm join --token SWMTKN-1-4rwdazrx23ta34w06pru9t26u9wjtgui9g9whayhdbsx210ybu-ciunlov23s4pbdafp515zvycm 192.168.7.111:2377
```

Utilizou-se o comando *Docker node ls* na máquina *manager* para verificar se todos os nós adicionados estão ativos. Com isso, uma lista com algumas informações sobre as máquinas que compõem o Swarm aparece, como por exemplo: o nome, se a mesma está ativa ou não, e o tipo *manager* (identificado pela presença da palavra *Leader* na coluna *MANAGER*) ou *worker* como é mostrado abaixo.

```
docker node ls

ID                            HOSTNAME        STATUS    AVAILABILITY    MANAGER
```

```
jth0npmpzeb8leby3ho1czshd     comp9-15          Ready         Active
xc7dyaw84qkyrchazbl6wc6zg *   comp9-16          Ready         Active                  Leader
```

O Docker Swarm, constrói um grupo cooperativo de sistemas que podem fornecer redundância, sendo que se um ou mais nós falharem (Desde que o *manager* ainda esteja ativo). Ele fornece balanceamento de carga de trabalho para os contêineres. Ele os atribui aos nós subjacentes e otimiza recursos ao programar automaticamente o contêiner e cargas de trabalho a serem executadas no host com recursos os mais adequados, mantendo os níveis de desempenho necessários. Um administrador de TI ou desenvolvedor controla o *cluster* usando um nó controlador chamado de *Manager*, que organiza e agenda os contêineres (NAINK, 2016, Tradução nossa).

5.2.3 Download e Configuração

Após ter sido feita a instalação do Docker, seguiu-se ao próximo passo, tendo sido feito o download do projeto disponível no GitHub, executando o seguinte comando:

```
git clone https://github.com/NLKNguyen/alpine-mpich
```

O próximo passo foi acessar a pasta alpine-mpich e executar um comando do Docker responsável por ler e interpretar o *Dockerfile* presente no diretório em questão, e a partir disso gerar uma imagem. Nesse arquivo, é utilizada como base o OS Alpine Linux 3.4, e sobre ele é feita a instalação e configuração do MPICH na versão 3.2 por padrão (sendo possível alterar), também é adicionando um usuário padrão com privilégios *root* de nome mpi e sem senha, e cria-se o diretório chamado *project*, após finalizado, a imagem resultante estará salva na máquina local, sendo inclusive possível inicializar um contêiner Docker a partir dela, semelhante a uma máquina virtual convencional.

5.2.3.1 *Dockerfile*

São arquivos utilizados pelo desenvolvedor para criar imagens Docker personalizadas. Com ele é possível no criar variáveis de ambiente, definir portas *Transmission Control Protocol* (TCP) que ficará visíveis do lado de fora do contêiner

enquanto isso, executa comandos no contêiner durante a compilação. A construção de uma imagem é a execução de todas as instruções dentro de um *Dockerfile* em sua própria camada e, no final, unindo as camadas em uma imagem (OUVERNEY; MARTINS, 2018).

5.2.3.2 Imagem Docker

Uma imagem é um componente de construção Docker a partir do qual os contêineres são lançados. Todas as imagens começam com uma imagem base, e então, outras partes são adicionadas sobre a base, conforme necessário, a fim de gerar uma nova. Por exemplo, uma imagem pode conter um sistema operacional CentOS como base, e em seguida, um *middleware* como o motor Node.js seria adicionado, e finalmente, seu aplicativo da web instalado em cima disso (JARAMILLO; NGUYEN; SMART, 2016, tradução nossa).

Segundo Alles (2018) uma imagem é um clone de um ambiente completo com exceção do *kernel*, e contém toda a informação de um contêiner, sendo a única referência necessária para a inicialização de uma aplicação virtualizada. Um dos motivos da grande popularidade do Docker, é a facilidade com que as imagens podem ser criadas e compartilhadas.

```
cd alpine-mpich
docker build -t nlknguyen/alpine-mpich base/
```

O comando *docker build* é usado para gerar uma imagem a partir do *Dockerfile*, já o argumento -t é usado para nomear a imagem criada que no caso mostrado acima é nlknguyen/alpine-mpich, e *base/* corresponde ao nome do diretório do arquivo. Como já mencionado, é possível escolher a versão do MPICH desejada e mudar o nome da imagem, vale ressaltar que, caso queira enviar a imagem criada ao repositório Docker Hub[7], é necessário colocar o nome de usuário (que no exemplo é comp8), seguido de barra antes do nome escolhido para a imagem, como é demonstrado no exemplo abaixo.

```
docker build MPICH_VERSION="3.2b4" -t comp8/nome-da-imagem base/
```

[7] https://hub.docker.com/

Depois de criada a imagem base, foi necessário ler um segundo *Dockerfile*, desta vez, localizado no diretório *onbuild*. Como base, é utilizada a imagem criada anteriormente. Caso o nome tenha sido alterado na etapa anterior, será necessário alterar a primeira linha o arquivo *Dockerfile* como mostra o exemplo a seguir.

```
#Mudar essa linha
FROM nlknguyen/alpine-mpich:latest
-------------------------------------
#para essa
FROM comp8/nome-da-imagem
```

```
docker build -t nlknguyen/alpine-mpich:onbuild onbuild/
```

O comando acima realizará a instalação do servidor SSH, e cópia de *scripts* que farão a automatização das conexões entre os contêineres, também é gerado as chaves criptografadas utilizando a ferramenta, além de toda a configuração do servidor, retirando a necessidade de senha, proibição de login como root, e ativação de autenticação baseada em chave pública, gera um arquivo de configuração padrão SSH que ignora a pergunta (sim / não) que aparece na primeira conexão realizada.

Ainda nesse arquivo existem algumas instruções *ONBUILD* que segundo Docker (2019, tradução nossa), adiciona à imagem uma instrução de disparo a ser executada posteriormente, quando a imagem é usada como base para outra construção. O acionador será executado no contexto da compilação, como se tivesse sido inserido imediatamente após a instrução *FROM* no *Dockerfile*.

Se por ventura houver alguma falha nos gatilhos, a instrução *FROM* será cancelada, o que, por sua vez, causará falha na construção da imagem. Se todos os gatilhos forem bem-sucedidos, a instrução *FROM* será concluída, e por fim os gatilhos são apagados da imagem final após serem executados (DOCKER, 2019, tradução nossa).

5.2.3.4 Configuração Final e Inicialização do *Cluster*

Por fim, é necessário acessar o diretório de nome *cluster,* contendo este, alguns arquivos incluindo duas pastas, sendo uma delas de nome *project* que contém o código fonte de uma programa MPI que serve para testar o funcionamento de cada contêiner criado, e também como exemplo de código, e a outra de nome ssh cujo o conteúdo são dois arquivos com chaves SSH pública e privada, que servirão para dar permissão a máquina host local de acessar o contêiner *master* do *cluster,* é necessário alterar a permissão do conteúdo da pasta para que seja possível realizar o acesso ao nó mestre posteriormente, para isso basta utilizar os comandos a seguir.

```
# Acessa o diretório cluster
cd cluster
# Muda permissão do conteúdo da pasta ssh, permitindo leitura e gravação ao proprietário
chmod -R 600 ssh/*
```

Neste ponto, é necessário criar toda a infraestrutura para garantir a comunicação entre nós, e para automatizar o processo, existe o *script* chamado swarm.sh, que contém comando Docker, bem como funções e variáveis para a inicialização do *cluster* de contêineres, com ele é possível realizar as configurações finais e definir o número de nós. O comando a seguir serve para salvar as configurações em variáveis de ambiente, que que servirão para a criação de um arquivo de configuração.

```
./swarm.sh config set \
   IMAGE_TAG=comp8/cluster     \
   PROJECT_NAME=my-mpi-project \
   NETWORK_NAME=mpi-network     \
   NETWORK_SUBNET=10.0.9.0/24   \
   SSH_ADDR=192.168.65.131       \
   SSH_PORT=2222
```

É necessário trocar o endereço IP da variável SSH_ADDR, colocando o da máquina local, que é o mesmo que é mostrado quando se inicia o modo Swarm do Docker, a mudança nas outras variáveis é opcional. IMAGE_TAG, corresponde

ao nome da imagem criada. O *script* possui um comando para enviar a imagem gerada para o repositório Docker Hub, para isso é necessário estar “logado”, basta digitar Docker login e informar o nome de usuário e senha, PROJECT_NAME, será o nome dos contêineres, NETWORK_NAME, nome da sub rede, NETWORK_SUBNET definirá a faixa da rede e SSH_PORT é referente a porta que será aberta.

```
docker login

Username: comp8
Password: comp8@cluster
```

Para inicializar é necessário entrar com o comando ./swarm.sh up size=4, esse cria um *cluster* com 4 nós, sendo um deles o *master* e os demais *workers*, o número 4 pode ser mudado, até mesmo para um número maior que os de máquinas físicas associadas ao *cluster*, podendo inclusive funcionar em apenas uma máquina física, os contêineres MPI que representam nós físicos ficam alocados no computador, se der tudo certo aparecerá as informações mostrada abaixo.

```
./swarm.sh up size=4
-------------------------------------------------//----------------------------------------
To run MPI programs in an interative shell:
  1. Login to master node:
     $ ./swarm.sh login

     which is equivalent to:
     $ ssh -o "StrictHostKeyChecking no" -i ssh/id_rsa -p 2222 mpi@192.168.65.131

  2. Execute MPI programs inside master node, for example:
     $ mpirun hostname
      *-----------------------------------------------------*
      | Default hostfile of connected nodes in the cluster |
      | is automatically updated at /etc/opt/hosts         |
      | To obtain hostfile manually: $ get_hosts > hosts   |
      * -----------------------------------------------------*

To run directly a shell command at master node:
     $ ./swarm.sh exec [COMMAND]
```

Example:
$ ./swarm.sh exec mpirun hostname

Existe a possibilidade do acesso de outras máquinas ao *cluster*, e estas não necessariamente precisam fazem parte do *cluster*. É necessário ter na máquina, onde pretende-se realizar o acesso, o diretório SSH que se encontra na pasta *cluster* do projeto e a linha de comando que aparece quando todos os serviços terminam de inicializar, que no caso mostrado acima é:

ssh -o "StrictHostKeyChecking no" -i ssh/id_rsa -p 2222 mpi@192.168.65.131

Para acessar o contêiner *master* basta digitar, *./swarm.sh login*, onde existe um arquivo teste, que foi copiado da máquina local e compilado para dentro de todos os contêineres. Também existem outros comandos do swarm.sh como: *./swarm.sh reload size=3*, que reseta todos os contêineres, *./swarm.sh scale size=2*, esse exclui nós caso o novo tamanho escolhido seja menor que o anterior e acrescenta novos caso seja maior, sem realizar a reinicialização da sub rede criada anteriormente, *./swarm.sh down,* para excluir o *cluster.* Por trás de tudo isso existem comandos Docker, que criam a rede virtual, os nós *master* e *worker* ligando-os a rede recém-criada, além de outras configurações, geram chaves SSH, constroem a imagem final e enviam para o repositório Docker Hub. Com um arquivo chamado Docker-compose.yml, que é utilizado para ajudar, juntamente com o arquivo mpi_bootstrap, a definir qual nó será *master* e qual será *worker.* Utilizando o comando *exit* é possível sair do nó *master*.

Docker Compose é uma ferramenta para definir e executar aplicativos Docker com vários contêineres. Com ele é possível utilizar um arquivo YAML para configurar os serviços da aplicação. Com um único comando, ele inicia todos os serviços e configurações desejadas (Docker, 2019, tradução nossa).

5.3 RESULTADOS E DISCUSSÕES

MPI é uma tecnologia bem estabelecida que é amplamente utilizada em ambientes computacionais de alto desempenho. Embora tenha baixa tolerância (se não zero) para erros de hardware ou de rede, ela oferece uma flexibilidade e mecanismo bem definido para os desenvolvedores construírem programas

distribuídos ou para construir estruturas ou ferramentas customizadas sobre ele. Tradicionalmente, a configuração de um cluster de computadores, por exemplo, um cluster MPI, é uma tarefa desafiadora que exige que os administradores de sistemas gastem um tempo considerável para configurar o sistema e a rede (NGUYEN, BEIN, 2017, tradução nossa).

Com ajuda da tecnologia de virtualização baseada em contêineres essa tarefa acaba ficando mais fácil, já que eles executam de maneira isolada, podendo ser criada uma sub rede para a comunicação entre eles, além de evitar problemas de compatibilidade, com uso em multiplataforma, e sem a necessidade de alterar nenhuma configuração das máquinas físicas. Por conta dessas vantagens trazidas pela tecnologia de contêineres, o tempo de implementação cai drasticamente. Como já mencionado neste trabalho existem algumas dificuldades em se implementar um *cluster* convencional, já muitas das configurações são realizadas manualmente, podendo levar algum tempo para deixar tudo pronto. Já está solução leva em média 30 minutos para ser finalizada.

5.3.1 Utilizando a Ferramenta

Quando o acesso ao nó mestre *do cluster* via SSH acontece, o usuário se encontra em um ambiente Alpine Linux, com sistema de arquivos e tudo mais. É possível digitar comando de atualização ou instalação de outras ferramentas no nó mestre, também como já mencionado, está instalado a ferramenta de desenvolvimento MPI, permitindo compilação e execução de programas. Por padrão, existe um programa MPI na pasta *project* dentro do diretório *cluster* do projeto que é compilado durante a criação de cada nó do *cluster,* também serve para testar cada nó durante a criação, caso haja algum erro na compilação desse arquivo, o contêiner MPI não inicializa. Para realizar a compilação de um código fonte MPI, basta inserir o comando mpicc seguido de -o, o nome desejado para o programa e o nome do código fonte.

```
mpi_hello_world    mpi_hello_world.c
#arquivos existentes na pasta project mostrado acima, antes da compilação
113cb375aa2a:/project$ mpicc -o teste mpi_hello_world.c
113cb375aa2a:/project$
```

```
# arquivos existentes na pasta project mostrado abaixo, depois da compilação
mpi_hello_world    mpi_hello_world.c  teste
```

Para realizar a execução do arquivo recém compilado, insira o código mpirun seguido de ./teste. Durante os testes, notou-se um problema, as vezes é necessário realizar uma cópia do programa para os outros contêineres, isso se dá por não haver instalado por padrão um programa de sistema de arquivos, como exemplo o servidor *Network File System* (NFS), que segundo Nix (2011), tem como função o compartilhamento de arquivos por meio da rede de comunicação, permitindo o acesso de outras maquinas a arquivos, como se estes estivessem em seus próprios arquivos locais.

Por isso é necesario copiar para a pasta *project* de todos os nós. Basta utilizar a conexão SSH para realizar a cópia para os demais nós. Primeiro digita-se o comando para ver o IP dos nós.

```
9c26da5e3a72:/project$ get_hosts
10.0.9.11
10.0.9.12
10.0.9.13

9c26da5e3a72:/project$
```

Depois digita-se scp seguido do nome do arquivo que se deseja copiar e por fim mpi@ o ip e o diretório.

```
9c26da5e3a72:/project$ scp teste mpi@10.0.9.11:/project

teste                         100%   11KB  10.6KB/s  10.6KB/s   00:00
```

Só então é possível executar o programa, também pode ser usada uma outra alternativa para resolver o problema, sendo ela à adição do comando de compilação no arquivo *Dockerfile* localizado no diretório *cluster*, além da cópia do arquivo com o código-fonte para a pasta *project* do projeto, dessa forma, evita-se a perda de tempo, pois caso sejam inicializados muitos contêineres, será necessário um bom tempo para copiar o programa manualmente para todos os nós.

```
RUN mpicc -o mpi_hello_world  mpi_hello_world.c \
```

```
&&mpicc -o teste  teste.c
```

Outro problema encontrado, é que o contêiner mestre acaba recebendo dois IPs, e isso impede que alguns programas MPIs funcionem corretamente. Como solução encontrada, basta realizar a exclusão de um endereço IP, sendo este o único que não segue uma sequência, por exemplo: se houverem quatro IPs com final 8, 15, 16 e 17, o IP de final 8 deverá ser excluído.

Por padrão o Alpine possui o editor de texto *Vi* porem é possível instalar outro, como por exemplo o nano utilizando o comando abaixo:

```
sudo apk update && apk add nano
```

E para realizar a exclusão use o editor para acessar o arquivo contendo todos os IPs, no diretório */etc/opt/hosts.* Após realizada a exclusão tudo funcionará normalmente.

```
9c26da5e3a72:/project$ mpirun -np 4 ./teste

Hello world from processor 75528387492e, rank 2 out of 4 processors
Hello world from processor 9c26da5e3a72, rank 3 out of 4 processors
Hello world from processor 40a08bd0926a, rank 1 out of 4 processors
Hello world from processor 40a08bd0926a, rank 0 out of 4 processors
```

No programa acima foram executados 4 processos distribuídos entre três nós, onde um deles processou 2 enquanto os outros apenas um cada.

5.3.2 Testes de desempenho

Apenas para verificar o funcionamento do *cluster*, foi realizado um pequeno teste envolvendo este mesmo programa citado acima, utilizando uma, duas, três e quatro máquinas, apenas para verificar se haveria algum ganho no tempo de execução. Em todos os testes foram instanciados vinte nós, e 1000 processos, e os resultados obtidos se encontram na tabela 1.

Tabela 1 - Tempos obtidos

Número de maquinas	Tempo
1	7.58 Segundos
2	6.2 Segundos
3	4.9 Segundos
4	3.48 Segundos

Fonte: do Autor

Os resultados obtidos demonstram um ganho significativo no tempo de execução da tarefa, comprovando que o *cluster* está operando normalmente, apesar de suas limitações, por se tratar de uma virtualização.

Foi realizado um outro teste utilizando um programa modificado por Nordlie (2003) e nomeado por ele de *flop.c*, que realiza o cálculo do pi e calcula o desempenho do *cluster* em Megaflops, os resultados não retratam o desempenho real, pois o cálculo do pi não realiza nenhum cálculo complexo, e por tanto não deve ser considerado um guia definitivo para realizar uma medição mais precisa de desempenho.

O teste consistiu em executar o *flop.c,* de um a dezesseis processos, repetindo cada teste três vezes e realizado a média aritmética, foi feito dessa forma apenas para se ter um resultado mais consistente, pois o resultado possuía uma variação considerável.

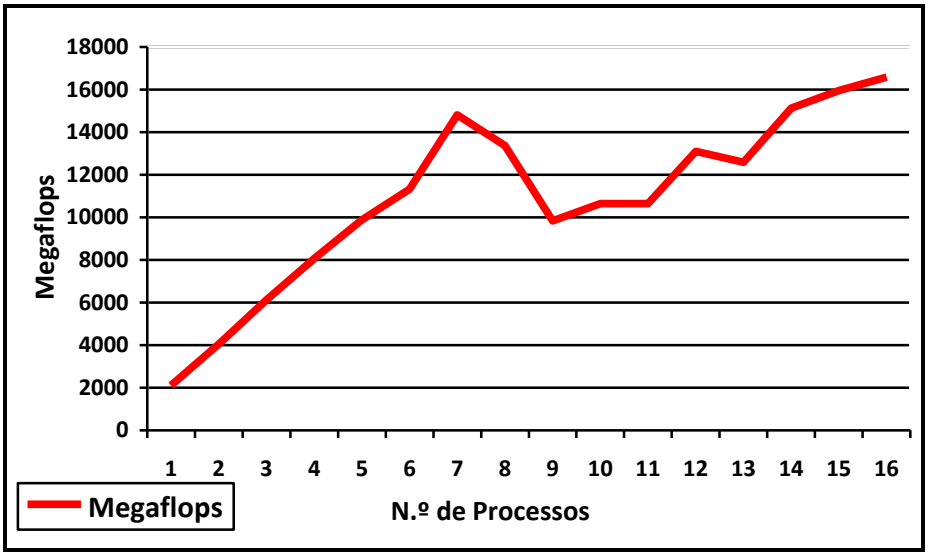
Tabela 2 - Tempos obtidos do teste *flop.c*

Número de processos	Megaflops	Tempo
1	2104.408	2.851
2	4042.297	1.484
3	6122.928	0.979
4	8062.173	0.744
5	9873.825	0.607
6	11329.179	0.529
7	14814.519	0.405
8	13361.230	0.449
9	9830.342	0.610
10	10935.476	0.548
11	10649.311	0.563
12	13104.087	0.457
13	12589.546	0.476
14	15120.620	0.396
15	15958.532	0.375
16	16579.445	0.361

Fonte: do Autor

Os resultados obtidos presentes na Tabela 2 acima, apontam um aumento constante de cerca de 2 Gigaflops em média para cada novo processo adicionado ao teste, até o com sete processos, seguido de uma queda nos dois testes seguintes e uma certa tendência a se estabilizar até o teste com treze processos, e tendo o seu ápice com dezesseis processos, onde alcançou cerca de 16.6 Gigaflops.

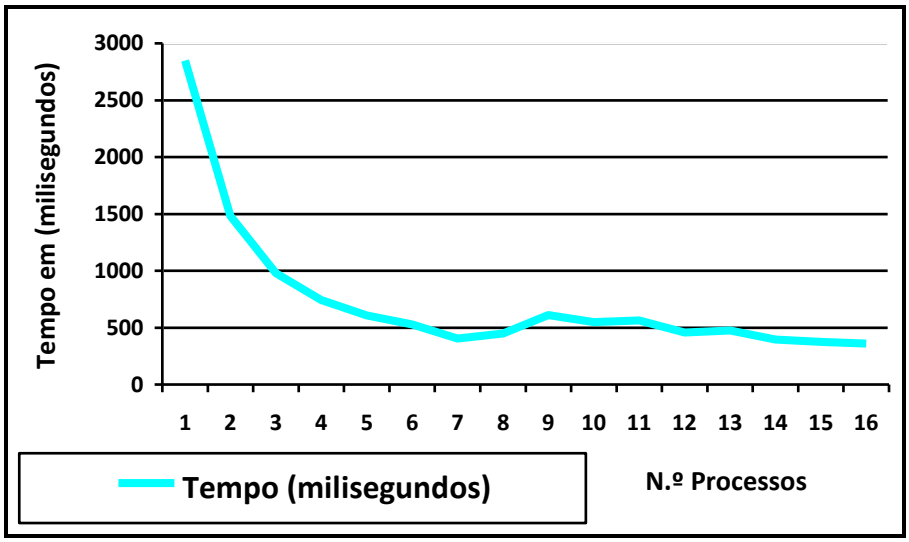
Figura 6 - Gráfico dos resultados obtidos do teste *flop.c* (Processos x *Megaflops*)



Fonte: do Autor

A figura 6 acima mostra o gráfico dos resultados obtido em relacionado o tempo ao número de processos em execução, sendo possível visualizar com mais clareza o escalonamento de desempenho obtido até o sétimo teste, quando é possível ver uma queda seguida de uma parente estabilização dos resultados

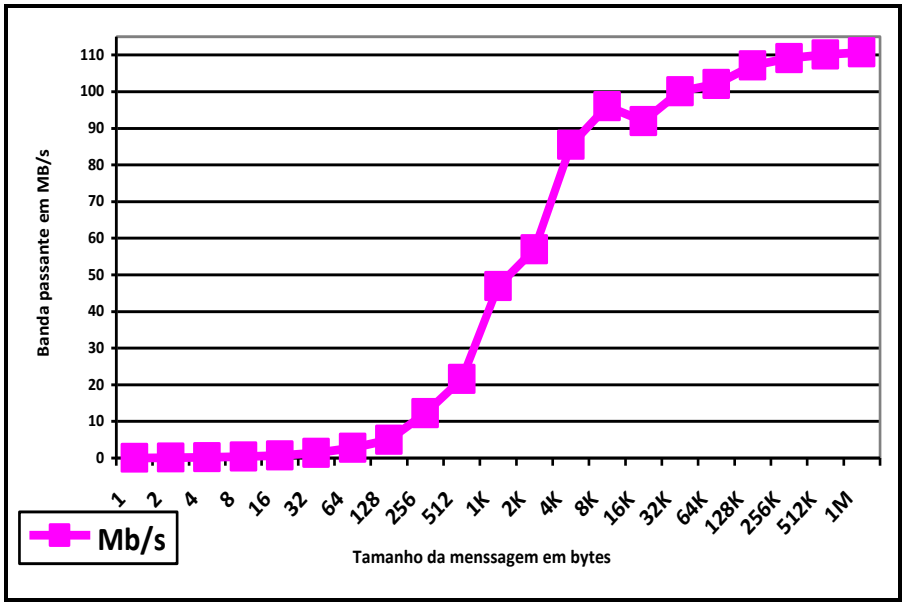
Figura 7 - Gráfico dos resultados obtidos do teste *flop.c* (Proxessos x Tempo)



Fonte: do Autor

Também foi gerado um gráfico relacionando o tempo de execução ao número de processos rodando representado pela figura 7, sendo possível observar mais uma vez uma brusca queda no tempo de execução até o sétimo teste, demonstrando a constância no aumento de desempenho em relação ao número de processos, seguido de uma estabilização nos tempos obtidos, tendo eles pouca diferença um em relação ao outro.

Figura 8 - Gráfico dos resultados obtidos do teste *ping_pong.c*



Fonte: do Autor

Foi realizado um terceiro teste com o programa *ping_pong.c,* e gerado um gráfico, representado pela figura 8, o mesmo serve para analisar o desempenho de rede do *cluster*. A partir dos resultados obtidos, é possível perceber um aumento na taxa de transferência entre o envio de 128 bytes a 16Kb, seguindo por uma estabilização no fluxo de mensagens.

6 CONCLUSÃO

Foi estudado e implementado neste trabalho, um modelo de *cluster* do tipo Beowulf utilizando contêineres Docker desenvolvido por Nguyen e Bein, com o objetivo de possibilitar aos alunos de computação paralela e distribuída, terem um maior contato com programação paralela, e buscando um ambiente o mais próximo possível do real. Com mais exemplos práticos em sala, os acadêmicos podem se interessar mais pela área de conhecimento. Sendo essa uma área de pesquisa importante, pois segundo Corrêa (2016), *clusters* de alto desempenho são utilizados para ajudar a resolver problemas complexos.

Sendo assim, é importante que os alunos tenham um maior contato com essa tecnologia, e aprendam a desenvolver projetos de computação paralela. Segundo Amaral (2006), infelizmente, nem sempre a universidade dispõe de recursos para montar laboratórios exclusivos para que sejam implementados *clusters,* e não podem “desfigurar” os laboratórios existentes, pois na maioria das vezes são para dar suporte as disciplinas.

Apesar das diferenças na implementação de *cluster*, foi possível, a partir dos testes realizados com o programa *flop.c*, pode-se constatar resultados semelhantes aos obtidos por (PRADO; SILVA, 2010), que percebeu em seus testes, que após um certo número de processos, o desempenho passa a se estabilizar, mantendo uma pequena diferença entre cada execução realizada.

A simplicidade, portabilidade e compatibilidade oferecida pela ferramenta Docker, torna possível implementar esse projeto em pouco tempo, se comparado a outros modelos já citados. E apesar de ser tecnicamente um *cluster* virtual, ele oferece um desempenho satisfatório pois segundo Alles (2018), um sistema com Docker executam o Kernel Linux sem a carga adicional de uma máquina virtual.

O presente trabalho conseguiu atingir vários objetivos, dantes estipulados, mesmo com as tentativas e falhas, foi possível aplicar os conhecimentos aprendidos em sala, além de aprender muito durante a realização da pesquisa do mesmo.

Com a implementação foi possível criar um ambiente onde os alunos possam aplicar os conceitos aprendidos na disciplina, podendo ter um contato direto com programação paralela por meio de um *cluster* virtual, com características muito próximas de um modelo real.

O modelo de *cluster* utilizado, trouxe grande aprendizado ao acadêmico,

que estudou o projeto minuciosamente, para que pudesse descrever cada etapa da implementação, buscando explicar de uma maneira clara todos os processos envolvidos.

Como trabalhos futuros, pode-se utilizar as ferramentas Docker disponíveis, para implementar *clusters* de outros tipos, usando o projeto utilizado neste trabalho como base, além de adicionar diretamente no arquivo *Dockerfile*, instalação de ferramentas didáticas de *Banchmark* para avaliação do desempenho, bem algum gerenciador de sistema de arquivos, como por exemplo o NFS, evitando o problema da necessidade de se copiar o programa MPI que se deseja executar para todos os contêineres do *cluster,* facilitando o uso.

REFERÊNCIAS

ABRANCHES, Marcelo Cerqueira de. **Um mecanismo de auto elasticidade com base no tempo de resposta para ambientes de computação em nuvem baseados em *containers***. 121 f. Dissertação (Mestrado em Computação Aplicada) - Universidade de Brasília, Brasília, 2016. Disponível em: http://repositorio.unb.br/bitstream/10482/22884/1/2016_MarceloCerqueiradeAbranches.pdf. Acesso em: 16 out 2018.

ALLES, Guilherme Rezende. **Análise da utilização de tecnologias de contêineres para aplicações de alto desempenho**. 59 f. Monografia (Graduação em Ciência da Computação) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2018. Disponível em: https://www.lume.ufrgs.br/bitstream/handle/10183/175014/001065151.pdf?sequence=1&isAllowed=y. Acesso em: 16 out 2018.

ALVES, Tiago Henrique Costa Rodrigues. **Uma arquitetura baseada em containers para workflows de bioinformática em nuvens federadas**. 2017. Disponível em: http://repositorio.unb.br/bitstream/10482/30994/1/2017_TiagoHenriqueCostaRodriguesAlves.pdf Acesso em: 05 jun. 2019.

AMARAL, Fábio Vieira do. ***Cluster* de alto desempenho**: agregando valor à infraestrutura computacional das universidades. 206 f. Dissertação (Mestrado em Engenharia da Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo, São Paulo, 2006. Disponível em: http://cassiopea.ipt.br/teses/2005_EC_Fabio_Vieira_Amaral.pdf. Acesso em: 22 out 2018.

ARCHER, Ben. *et al*. **System x Virtualization Strategies**. 2010. Disponível em: https://lenovopress.com/redp4480.pdf Acesso em: 20 nov. 2018.

BACELLAR, Hilário Viana. ***Cluster***: computação de alto desempenho. 6 f. Artigo Científico (Graduação em Ciência da Computação) – Universidade Estadual de Campinas, Campinas, 2009. Disponível em: http://www.ic.unicamp.br/~ducatte/mo401/1s2010/T2/107077-t2.pdf. Acesso em: 26 jun. 2019.

BACHIEGA, Naylor G. *et al*. **Container-Based Performance Evaluation: A Survey and Challenges.** In: Cloud Engineering (IC2E), 2018 IEEE International Conference on. IEEE, 2018. p. 398-403. Disponivel em: https://ieeexplore.ieee.org/document/8360360. acesso em: 23 nov. 2018.

BACHIEGA, Naylor G. *et al*. **Mapeamento Sistemático do Ensino Teórico e Prático de Programação Paralela**. In: CONGRESSO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 6., 2017, São Carlos – Brasil. Anais dos Workshops do VI Congresso Brasileiro de Informática na Educação. Recife- Pe: Sociedade Brasileira de Computação – SBC, 2017. p. 1089 - 1098. Disponível em: http://brie.org/pub/index.php/wcbie/article/view/7498. Acesso em: 23 out. 2018.

BIAZOTTO, Anderson Murgi. **Desenvolvimento de um *cluster* de alto desempenho**. 30 f. Projeto de Desenvolvimento (Graduação em Ciência da Computação) – Instituto Municipal de Ensino Superior de Assis, IMESA, Assis, SP, 2011. Disponível em: https://cepein.femanet.com.br/BDigital/arqTccs/0811270075.pdf. Acesso em: 21 abril 2018.

CARISSIMI, Alexandre. **Virtualização**: Princípios básicos e aplicações. Minicurso da 9ª Escola Regional de Alto Desempenho-ERAD, p. 39-69, 2009. Disponível em: http://www.lbd.dcc.ufmg.br/colecoes/erad/2009/004.pdf. Acesso em: 11 dez. 2018.

CARISSIMI, Alexandre. **Virtualização: da teoria a soluções**. In: Simpósio Brasileiro de Redes de Computadores–SBRC, 26.,2008, Rio de Janeiro. Anais SBRC 2008, Rio de Janeiro: Sociedade Brasileira de Computação p.173-207 Disponível em: http://*host*el.ufabc.edu.br/~marcelo.nascimento/BC1518Q3/arquivos/virtualizacao_cap4-v2.pdf. Acesso em: 11 dez. 2018.

CARNEIRO, Lucas Santana; DUARTE, Angelo Amancio. **Differences in the Performance of Parallel Applications in Physical and Virtual Clusters.** Ieee Latin America Transactions, [s.l.], v. 16, n. 2, p.604-612, fev. 2018. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tla.2018.8327419. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8327419&tag=1. Acesso em: 28 jun. 2019.

CARVALHO, Vladimir Fabrício Pereira de. **Otimização de um *cluster* de alto desempenho para o uso do programa PGENESIS em simulações biologicamente plausíveis em larga-escala de sistemas neurais**. 102 f. Dissertação (Mestrado em Ciências) – Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto da USP, Ribeirão Preto, SP, 2007. Disponível em: http://www.teses.usp.br/teses/disponiveis/59/59135/tde-06042008-205710/en.php. Acesso em: 05 out. 2018.

CORRÊA, José Norberto Guiz Fernandes. **LIFTER:** disponibilização de aplicações via *container*s de *software* em um *cluster* de alto desempenho. 93 f. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Universidade Federal de Santa Catarina, Florianópolis, 2016. Disponível em: https://repositorio.ufsc.br/xmlui/bitstream/handle/123456789/171410/TCC_Lifter_JoseNorberto.pdf?sequence=1&isAllowed=y. Acesso em: 03 mar. 2019.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas distribuídos: conceitos e projeto**. 4.ed. Porto Alegre: Bookman Editora, 2013.

CROSBY, Michael. **What is containerd?** 2017. Disponível em: https://blog.docker.com/2017/08/what-is-containerd-runtime/. Acesso em: 07 jun. 2019.

DOCKER. **Docker.** Disponível em: https://www.docker.com. Acesso em: 15 jun. 2019.

CHROOT. Disponível em: https://wiki.archlinux.org/index.php/Chroot. Acesso em: 27 nov. 2018.

DOCKER. **Docker Launches New Enterprise Edition to Accelerate Business-Critical Application Deployments Across Hybrid Clouds.** 2017. Disponível em: https://www.docker.com/docker-news-and-press/docker-launches-new-enterprise-edition-accelerate-business-critical. Acesso em: 05 jun. 2019.

SANTOS, Marcelo Carvalho dos. **MODELO COMPUTACIONAL PARALELO BASEADO EM GPU PARA CÁLCULO EM TEMPO REAL DA DISPERSÃO ATMOSFÉRICA DE RADIONUCLÍDEOS NAS VIZINHANÇAS DE UMA CENTRAL NUCLEAR.** 2018. 80 f. Dissertação (Mestrado) - Curso de Ciência e Tecnologia Nucleares, Instituto de Engenharia Nuclear, Rio de Janeiro, 2018. Disponível em: https://core.ac.uk/download/pdf/159274502.pdf. Acesso em: 29 jun. 2019.

EDUARDO, Fernando Kanda Kassobo. **Implementação de *cluster* de alto desempenho usando GNU/linux**. 111 f. Monografia (Graduação em Ciência da Computação) - Instituto Superior Politécnico Metropolitano de Angola, Luanda, 2013. Disponível em: https://pt.slideshare.net/kandakassobo/implementao*cluster* altodesempenhofernandoeduardo20090726imetro2013. Acesso em: 15 out. 2018.

ELLER JUNIOR, Edgar. **Estudo de Tecnologias para Computação Paralela e Distribuída:** Implementação de um Cluster Beowulf. 2013. 163 f. Dissertação (Mestrado) - Curso de Modelagem Computacional em Ciência e Tecnologia, Universidade Federal Fluminense, Volta Redonda, 2013. Disponível em: http://www.professores.uff.br/gbenitez/wp-content/uploads/sites/98/2017/08/Dissertacao_8.pdf. Acesso em: 28 jun. 2019.

FERNÁNDEZ, César Castelo. **Computação de Alto Desempenho usando Clusters**. Campinas, Sp. jul. 2009. Disponível em: http://www.ic.unicamp.br/~ducatte/mo401/1s2009/T2/089028-t2.pdf. Acesso em: 29 jun. 2019.

JARAMILLO, David; NGUYEN, Duy V.; SMART, Robert. **Leveraging microservices architecture by using Docker technology**. In: SoutheastCon 2016. IEEE, 2016. p. 1-5. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7506647. Acesso em: 08 jun. 2019.

JARDIM, Igor Lorenson de Campos; SÁ, Paulo Sérgio Salla. **Aplicabilidade de um Sistema em Cluster**. Revistas Ensaios & Diálogos, São Paulo, v. 3, n. 1, p.49-60, dez. 2010. Disponível em: https://intranet.redeclaretiano.edu.br/download?caminho=upload/cms/revista/arquivos/32.pdf&arquivo=32.pdf. Acesso em: 21 nov. 2018.

JOHNSON, Eric et al. **Performance of a virtual cluster in a general-purpose teaching laboratory**. In: Cluster Computing (CLUSTER), 2011 IEEE International Conference on. IEEE, 2011. p. 600-604. Disponivel em: https://ieeexplore.ieee.org/abstract/document/6061157 Acesso em: 26 nov. 2018.

KIM, Tae-Hoon; JIANG, Keyuan; RAJPUT, V. S. **Adoption of container-based virtualization in it education**. In: ASEE's 123rd Annual-Conference and Exposition, New Orleans. 2016. Disponível em: https://peer.asee.org/adoption-of-*container*-based-virtualization-in-it-education.pdf. Acesso em: 03 nov. 2018.

KSHEMKALYANI, Ajay D.; SINGHAL, Mukesh. **Distributed Computing:** Principles, Algorithms, and Systems. Reino Unido: Cambridge University Pres, 2008. 756 p.

KUBERNETES **Design and Architecture**: Overview. Overview. Disponível em: https://github.com/kubernetes/community/commit/e00ed401dc94f542666cde7aa5b6886c90c23bed. Acesso em: 09 jun. 2019.

LEE, Hyungro. **Virtualization basics**: Understanding techniques and fundamentals. Artigo Científico - School of Informatics and Computing, Indiana University, 2014. Disponível em: https://pdfs.semanticscholar.org/68aa/d882f62d0bd0779395e7803c607340da3fa8.pdf. Acesso em: 09 nov. 2018.

LEÓN, Edgar A. et al. **Characterizing parallel scientific applications on commodity *clusters***: An empirical study of a tapered fat-tree. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press, 2016. 78 p. Disponível em: https://ieeexplore.ieee.org/document/7877156. Acesso em: 24 nov. 2018.

LUDIN, Mobeen et al. **LittleFe**: The high performance computing education appliance. In: IEEE INTERNATIONAL CONFERENCE ON *CLUSTER* COMPUTING (*CLUSTER*). IEEE, 2013. p. 1-1. Disponível em: https://ieeexplore.ieee.org/document/6702649. Acesso em: 04 jun. 2019.

MAIA, Régis Matheus Silveira. **Ferramenta para criação de *clusters* virtuais para o ensino de programação paralela e distribuída**. 46 f. Trabalho de Conclusão de Curso (Graduação em Redes de Computadores) - Universidade Federal do Ceará, Quixadá, 2016. Disponível em: http://www.repositorio.ufc.br/handle/riufc/24785. Acesso em: 25 nov. 2018.

MESTRE, Demetrio Gomes. **Leveraging the Entity Matching Performance through Adaptive Indexing and Efficient Parallelization**. 2018. 173 f. Tese (Doutorado) - Curso de Ciência da Computação, Universidade Federal de Campina Grande, Campina Grande, Paraíba, Brasil, 2018. Disponível em: http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/1683 Acesso em: 19 out. 2018.

MEYER, Luiz Antônio Vivacqua Corrêa. **Uma Visão Geral dos Sistemas distribuídos de Cluster e Grid e suas Ferramentas para o processamento paralelo de dados**. IBGE [sd]. Disponível em : https://ww2.ibge.gov.br/confest_e_confege/pesquisa_trabalhos/CD/palestras/368-1.pdf. Acesso em 17 out. 2018.

MONTEIRO, Luciano de aguiar; ALMEIDA, Washington Henrique Carvalho**. Cluster de Alta Disponibilidade com Docker Swarm**. Disponível em: http://www.eripi.com.br/2017/images/anais/minicursos/2.pdf Acesso em: 05 mai. 2019.

MORENO, Edward; LIMA, Felipe Anjos; DIAS, Wanderson Roger Azevedo. **Performance Analysis of a Low Cost Cluster with Parallel Applications and ARM Processors**. Ieee Latin America Transactions, [s.l.], v. 14, n. 11, p.4591-4596, nov. 2016. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tla.2016.7795834. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7795834. Acesso em: 29 jun. 2019.

NAMESPACES, Company. ***Linux* Programmer's Manual**. Disponível em: http://man7.org/linux/man-pages/man7/namespaces.7.html. Acesso em: 20 jun, 2019

NARALE, Snehal A.; BUTEY, P. K. **Throttled Load Balancing Scheduling Policy Assist to Reduce Grand Total Cost and Data Center Processing Time in Cloud Environment Using Cloud Analyst**. In: Second International Conference on Inventive Communication and Computational Technologies (ICICCT). IEEE, 2018. p. 1464-1467. Disponível em: https://www.researchgate.net/scientific-contributions/2147727762_Snehal_A_Narale. Acesso em: 22 nov. 2018

NORDLIE, John. **Mini-Wulf.:** A small Beowulf Cluster running FreeBSD. 2003. Disponível em: https://docs.huihoo.com/hpc-cluster/mini-wulf/index.html. Acesso em: 03 jul. 2019.

NIX. **NFS:** Network File System. 2011. Disponível em: https://ivanix.wordpress.com/2011/07/19/servidor-nfs/. Acesso em: 03 jul. 2019.

NGUYEN, Nikyle; BEIN, Doina. **Distributed MPI Cluster with Docker Swarm Mode**. In: ANNUAL COMPUTING AND COMMUNICATION WORKSHOP AND CONFERENCE (CCWC), 7., 2017, Las Vegas. NV**.** Las Vegas: Ieee, 2017. p. 1 - 7. Disponível em: https://ieeexplore.ieee.org/document/7868429. Acesso em: 03 jun. 2019.

OLIVEIRA, Alexandre Tavares de. **Uma Plataforma de Rede Definida para Ambientes de Computação Paralela e Distribuída.** 2019. 67 f. Dissertação (Mestrado) - Curso de Pós-graduação em Ciência da Computação, Universidade Federal de Juiz de Fora, Juiz de Fora, 2019. Disponível em: http://hermes.cpd.ufjf.br:8080/jspui/bitstream/ufjf/10019/3/alexandretavaresdeoliveira.pdf. Acesso em: 29 jun. 2019.

OLIVEIRA, Nathan Batista de. **Aplicações em contêineres**. 47 f. Monografia (Graduação em Sistemas de Telecomunicações) - Instituto Federal de Santa Catarina – IFSC, São José - SC, 2017. Disponível em: https://wiki.sj.ifsc.edu.br/wiki/images/e/e0/TCC_Nathan_Oliveira.pdf. Acesso em: 16 out. 2018.

OUVERNEY, Kássio Gomes; MARTINS, Daves. **Automação do processo de implantação de *software* usando *Docker* e microserviços** – um estudo de casos. Artigo Científico (Graduação Ciência da Computação) – Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais, Juiz de Fora, MG, 2018. Disponível em: http://periodicos.jf.ifsudestemg.edu.br/revistabsi/article/view/187/93. Acesso em: 10

maio 2019.

PIOVESAN, Braian Jacomelli. **Análise da viabilidade de uso do protocolo IP Anycast em soluções de alta disponibilidade e balanceamento de carga**. 26 f. Trabalho de Conclusão de Curso (Graduação em Sistema de Informação) – Universidade Federal de Santa Maria, UFSM, Santa Maria, 2017. Disponível em: https://repositorio.ufsm.br/bitstream/handle/1/12919/TCCG_SIFW_2017_PIOVESAN_BRAIAN.pdf?sequence=1&isAllowed=y. Acesso em: 20 nov. 2018.

PITANGA, Marcos. **CONSTRUINDO SUPER COMPUTADORES COM LINUX.** 3. ed. Rio de Janeiro: Brasport, 2008. 400 p.

POPEK, G. J.; GOLDBERG, R. P. **Formal requirements for virtualizable third generation architectures.** Commun. ACM, ACM, New York, NY, USA, v. 17, n. 7, p. 412–421, jul. 1974. ISSN 0001-0782.

PRADO, Claudio Lima do; SILVA, João Messias Alves da. **Aplicação de *cluster Beouwulf* em instituições de ensino.** 119 f. Monografia (Graduação em Tecnologia em Informática) – Faculdade de Tecnologia de Guaratinguetá, Guaratinguetá, SP, 2010. Disponível em: <https://docplayer.com.br/992047-Aplicacao-de-*cluster*-*Beowulf*-em-instituicoes-de-ensino.html>. Acesso em: 15 abr. 2018.

PRPIC, Martin; LANDMANN, Rüdiger; SILAS, Douglas. **Guia de gerenciamento de recursos:** gerenciando recursos de sistema no Red Hat Enterprise *Linux* 6. 1.ed. Disponível em: https://access.redhat.com/documentation/pt-br/red_hat_enterprise_linux/6/html/resource_management_guide/. Acesso em: 27 nov. 2018.

ROSTIROLLA, Gustavo. **ANÁLISE DE DESEMPENHO E CONSUMO ENERGÉTICO DE UM CLUSTER BASEADO EM COMPUTADORES DE PLACA ÚNICA.** 2014. 92 f. TCC (Graduação) - Curso de Ciências Exatas e Tecnológicas, Centro Universitário Univates, Lageado, 2014. Disponível em: https://www.univates.br/bdu/bitstream/10737/579/1/2014GustavoRostirolla.pdf. Acesso em: 29 jun. 2019.

SHAFI, Aamir et al. **Teaching parallel programming using java**. In: Proceedings of the Workshop on Education for High-Performance Computing. IEEE Press, 2014. p. 56-63. Disponível em: https://ieeexplore.ieee.org/document/7016359Acesso em 24 set. 2018.Acesso em 24 set. 2018**.**

SHENG, Yanxiu et al. **Layered Models for General Parallel Computation Based on Heterogeneous System**. 2012 13th International Conference On Parallel And Distributed Computing, Applications And Technologies**,** [s.l.], p.45-49, dez. 2012. IEEE. http://dx.doi.org/10.1109/pdcat.2012.85. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6589239. Acesso em: 29 jun. 2019.

SILVA, Luís Moura e; BUYYA, Rajkumar. **Parallel programming models and paradigms.** In: High performance *cluster* computing, v.1, pp 4-26, Prentice Hall,

1999. Disponível em: http://www.buyya.com/*cluster*/v2chap1.pdf. Acesso em: 23 out. 2018.

SSH COMMUNICATIONS SECURITY (Waltham). **SSH PROTOCOL.** Disponível em: https://www.ssh.com/ssh/protocol/. Acesso em: 29 jul. 2019.

TANENBAUM, Andrew S.; BOS, Herbert. **MODERN OPERATING SYSTEMS**. 4. ed. Amsterdam, The Netherlands: Pearson Prentice Hall, 2015. 1137 p.

THOMÉ, Bruna; HENTGES, Eduardo; GRIEBLER, Dalvan. **Computação em Nuvem: análise comparativa de ferramentas open source para iaas**. 11th Escola Regional de Redes de Computadores (ERRC), p. 4, 2013. Disponível em: https://jornada.ifsuldeminas.edu.br/index.php/jcmch2/jcmch2/paper/viewFile/1780/1217. Acesso: 28 jun. 201

APÊNDICE(S)

APÊNDICE A – ARTIGO

Cluster de Alto Desempenho Para Uso na disciplina de Computação Paralela e Distribuída Utilizando Contêineres

Michael B. Nascimento[1], Paulo João Martins[1]

[1]Universidade do Extremo Sul Catarinense (UNESC) – Criciúma – SC – Brasil

mb-nascimento@hotmail.com, pjm@unesc.net

Abstract. *The growing need for ever more computational power, drives companies and universities to seek alternatives, such as the Beowulf cluster for having a lower cost compared to supercomputers. To further reduce costs, they also use shared environments, but this brings some application dependency and compatibility issues as container solutions emerge, which are a lighter virtualization and capable of isolating applications. The present work looks for alternatives to implement a high-performance cluster based on containers, to be used in the parallel and distributed computing discipline, and with this to avoid problems of compatibility, lack of infrastructure and delay in the cluster installation process, and with this, students will have the experience of developing parallel programs, in an environment close to the real one.*

Keywords*: Containers, Docker, Parallel Computing, Virtualization, Cluster.*

Resumo. *A crescente a necessidade de se obter cada vez mais poder computacional, faz com que empresas e universidades busquem alternativas, como por exemplo o cluster Beowulf, por ter um custo menor em relação aos supercomputadores. Para reduzir ainda mais os custos também utilizam ambientes compartilhados, porem isso traz alguns problemas de dependências e compatibilidade das aplicações, como solução surgem os contêineres, sendo estes uma virtualização leve e capaz de isolar as aplicações. O presente trabalho busca alternativas de implementar um cluster de alto desempenho baseada em contêineres, para ser utilizado na disciplina de computação paralela e distribuída, e com isso evitar problemas de compatibilidade, falta de infraestrutura, e demora no processo de instalação do cluster, e com isso dar aos alunos a experiência de desenvolverem programas paralelos, em um ambiente próximo ao real.*

Palavras-chave*: Contêineres, Docker, Computação Paralela, Virtualização, Cluster.*

1. Introdução

Nos mais variados campos científicos e profissionais, tem-se notado um aumento na necessidade de recursos computacionais de alto desempenho, tendo em vista que estes são de extrema importância na solução de problemas cada vez maiores (CORRÊA, 2016). Segundo Carneiro e Duarte (2018 apud, PITANGA 2008), por conta dos custos elevados da compra de supercomputadores, tem-se a necessidade de uma busca por alternativas mais viáveis. Uma solução popular de baixo custo é o cluster Beowulf, que pode vir a ser utilizados por

universidades e empresas menores na obtenção de poder computacional de alto desempenho, a um custo relativamente menor que um supercomputador.

Porém mesmo com custos menores, a aquisição de um cluster Beowulf não é barato, pois são necessários investimentos para se adquirir os recursos necessários a montagem do mesmo, além de gastos com manutenção. Por conta disso, tende-se a utilizar ambientes compartilhados por grupos de pesquisa e empresas, as quais precisam de um alto poder computacional para suas aplicações, no intuito de diminuir as despesas (CORRÊA, 2016).

Grupos menores enfrentam problemas como: conflito de dependências entre aplicações, recursos indisponíveis, entre outros. Com o avanço das tecnologias surgem os contêineres de software, possibilitando uma abordagem mais flexível, quanto a disponibilização de aplicações (CORRÊA, 2016). Existem vários softwares gerenciadores de contêineres, tais como: Docker, Kubernetes, Azure Container Service e OpenShift Container, dentre outros.

Muitas instituições possuem pouco ou nenhum acesso a um ambiente de computação paralela e distribuída onde se possa aplicar os conceitos aprendidos em sala de aula (LUDIN et al., 2013). Pensando nisso, este trabalho tem como objetivo a implementação de um cluster de alto desempenho, utilizando como base o sistema operacional Linux e se propondo a estudar métodos simplificados para que os alunos possam criar o cluster no próprio laboratório utilizado na disciplina, para isso, pretende-se utilizar softwares gerenciadores de contêineres buscando uma melhor opção para o caso.

A ideia do projeto é dar suporte a disciplina de computação paralela e distribuída, pois a criação dos clusters convencionas costuma tomar um bom tempo de instalação e configuração, tornando o processo custoso pois todo o semestre os computadores da universidade são formatados e consequentemente o cluster precisa ser novamente instalado.

2 Instalação do Docker

A solução adotada no projeto foi desenvolvida por Nikyle Nguyen e Doina Bein, por ser uma ferramenta bem completa e com scripts que ajudam a realizar a instalação de forma automatizada, além de permitir a modificação dos scripts básicos, dando a possibilidade de customização da implementação. O artigo do projeto foi publicado na IEEE Xplore Digital Library no ano de 2017. E é possível encontra-lo no site GitHub pois está em modo público, e utiliza a ferramenta Docker na criação do cluster.

A primeira etapa necessária do projeto, foi instalação do Docker em todas as máquinas que fariam parte do cluster. A implementação desse projeto foi realizada no laboratório de Computação da Universidade, em função da disciplina de Computação Paralela e Distribuída, ser ministrada neste local.

2.1 Iniciando o Docker Swarm

Foi inicializado o modo Swarm do Docker na máquina escolhida para vir a ser o nó mestre, utilizando o comando docker swarm init tornando-a um nó gerenciador ou Manager, do modo Swarm. Foi necessário copiar a saída resultante do comando, para executa-la no terminal de todas as máquinas que fariam parte do cluster, onde deve ser adicionado os trabalhadores ou workers.

2.2 Download e Configuração

Após ter sido feita a instalação do Docker, seguiu-se ao próximo passo, tendo sido feito o download do projeto disponível no GitHub, executando o seguinte comando:

```
git clone https://github.com/NLKNguyen/alpine-mpich
```

O próximo passo foi acessar a pasta alpine-mpich e executar um comando do Docker responsável por ler e interpretar o Dockerfile presente no diretório em questão, e a partir disso gerar uma imagem. Nesse arquivo, é utilizada como base o OS Alpine Linux 3.4, e sobre ele é feita a instalação e configuração do MPICH na versão 3.2 por padrão (sendo possível alterar), também é adicionando um usuário padrão com privilégios root de nome mpi e sem senha, e cria-se o diretório chamado project, após finalizado, a imagem resultante estará salva na máquina local, sendo inclusive possível inicializar um contêiner Docker a partir dela, semelhante a uma máquina virtual convencional.

```
cd alpine-mpich
docker build -t nlknguyen/alpine-mpich base/
```

O comando docker build é usado para gerar uma imagem a partir do Dockerfile, já o argumento -t é usado para nomear a imagem criada que no caso mostrado acima é nlknguyen/alpine-mpich, e base/ corresponde ao nome do diretório do arquivo.

Depois de criada a imagem base, foi necessário ler um segundo Dockerfile, desta vez, localizado no diretório onbuild. Como base, é utilizada a imagem criada anteriormente. Caso o nome tenha sido alterado na etapa anterior, será necessário alterar a primeira linha o arquivo Dockerfile.

```
docker build -t nlknguyen/alpine-mpich:onbuild onbuild/
```

2.3 Configuração Final e Inicialização do Cluster

Por fim, é necessário acessar o diretório de nome cluster, contendo este, alguns arquivos incluindo duas pastas, sendo uma delas de nome project que contém o código fonte de uma programa MPI que serve para testar o funcionamento de cada contêiner criado, e também como exemplo de código, e a outra de nome ssh cujo o conteúdo são dois arquivos com chaves SSH pública e privada, que servirão para dar permissão a máquina host local de acessar o contêiner master do cluster, é necessário alterar a permissão do conteúdo da pasta para que seja possível realizar o acesso ao nó mestre posteriormente, para isso basta utilizar os comandos a seguir.

```
cd cluster
chmod -R 600 ssh/*
```

Neste ponto, é necessário criar toda a infraestrutura para garantir a comunicação entre nós, e para automatizar o processo, existe o script chamado swarm.sh, que contém comando Docker, bem como funções e variáveis para a inicialização do cluster de contêineres, com ele é possível realizar as configurações finais e definir o número de nós. O comando a seguir serve para salvar as configurações em variáveis de ambiente, que que servirão para a criação de um arquivo de configuração.

```
./swarm.sh config set \
  IMAGE_TAG=comp8/cluster    \
  PROJECT_NAME=my-mpi-project \
  NETWORK_NAME=mpi-network    \
  NETWORK_SUBNET=10.0.9.0/24  \
  SSH_ADDR=192.168.65.131     \
  SSH_PORT=2222
```

É necessário trocar o endereço IP da variável SSH_ADDR, colocando o da máquina local, que é o mesmo que é mostrado quando se inicia o modo Swarm do Docker, a mudança nas outras variáveis é opcional. IMAGE_TAG, corresponde ao nome da imagem criada. O script possui um comando para enviar a imagem gerada para o repositório Docker Hub, para isso é necessário estar "logado", basta digitar Docker login e informar o nome de usuário e senha, PROJECT_NAME, será o nome dos contêineres, NETWORK_NAME, nome da sub rede, NETWORK_SUBNET definirá a faixa da rede e SSH_PORT é referente a porta que será aberta.

```
docker login
Username: comp8
Password: comp8@cluster
```

Para inicializar é necessário entrar com o comando ./swarm.sh up size=4, esse cria um cluster com 4 nós, sendo um deles o master e os demais workers, o número 4 pode ser mudado, até mesmo para um número maior que os de máquinas físicas associadas ao cluster, podendo inclusive funcionar em apenas uma máquina física, os contêineres MPI que representam nós físicos ficam alocados no computador.

Para acessar o contêiner master basta digitar, ./swarm.sh login, onde existe um arquivo teste, que foi copiado da máquina local e compilado para dentro de todos os contêineres. Também existem outros comandos do swarm.sh como: ./swarm.sh reload size=3, que reseta todos os contêineres, ./swarm.sh scale size=2, esse exclui nós caso o novo tamanho escolhido seja menor que o anterior e acrescenta novos caso seja maior, sem realizar a reinicialização da sub rede criada anteriormente, ./swarm.sh down, para excluir o cluster. Por trás de tudo isso existem comandos Docker, que criam a rede virtual, os nós master e worker ligando-os a rede recém-criada, além de outras configurações, geram chaves SSH, constroem a imagem final e enviam para o repositório Docker Hub. Com um arquivo chamado Docker-compose.yml, que é utilizado para ajudar, juntamente com o arquivo mpi_bootstrap, a definir qual nó será master e qual será worker. Utilizando o comando exit é possível sair do nó master.

Docker Compose é uma ferramenta para definir e executar aplicativos Docker com vários contêineres. Com ele é possível utilizar um arquivo YAML para configurar os serviços da aplicação. Com um único comando, ele inicia todos os serviços e configurações desejadas (Docker, 2019, tradução nossa).

3. Resultados e discuções

MPI é uma tecnologia bem estabelecida que é amplamente utilizada em ambientes computacionais de alto desempenho. Embora tenha baixa tolerância (se não zero) para erros

de hardware ou de rede, ela oferece uma flexibilidade e mecanismo bem definido para os desenvolvedores construírem programas distribuídos ou para construir estruturas ou ferramentas customizadas sobre ele. Tradicionalmente, a configuração de um cluster de computadores, por exemplo, um cluster MPI, é uma tarefa desafiadora que exige que os administradores de sistemas gastem um tempo considerável para configurar o sistema e a rede (NGUYEN, BEIN, 2017, tradução nossa).

Com ajuda da tecnologia de virtualização baseada em contêineres essa tarefa acaba ficando mais fácil, já que eles executam de maneira isolada, podendo ser criada uma sub rede para a comunicação entre eles, além de evitar problemas de compatibilidade, com uso em multiplataforma, e sem a necessidade de alterar nenhuma configuração das máquinas físicas. Por conta dessas vantagens trazidas pela tecnologia de contêineres, o tempo de implementação cai drasticamente. Como já mencionado neste trabalho existem algumas dificuldades em se implementar um cluster convencional, já muitas das configurações são realizadas manualmente, podendo levar algum tempo para deixar tudo pronto. Já está solução leva em média 30 minutos para ser finalizada.

3.1 Utilizando a Ferramenta

Quando o acesso ao nó mestre do cluster via SSH acontece, o usuário se encontra em um ambiente Alpine Linux, com sistema de arquivos e tudo mais. É possível digitar comando de atualização ou instalação de outras ferramentas no nó mestre, também como já mencionado, está instalado a ferramenta de desenvolvimento MPI, permitindo compilação e execução de programas. Por padrão, existe um programa MPI na pasta project dentro do diretório cluster do projeto que é compilado durante a criação de cada nó do cluster, também serve para testar cada nó durante a criação, caso haja algum erro na compilação desse arquivo, o contêiner MPI não inicializa. Para realizar a compilação de um código fonte MPI, basta inserir o comando mpicc seguido de -o, o nome desejado para o programa e o nome do código fonte.

```
113cb375aa2a:/project$ mpicc -o teste mpi_hello_world.c
```

Para realizar a execução do arquivo recém compilado, insira o código mpirun seguido de ./teste. Durante os testes, notou-se um problema, as vezes é necessário realizar uma cópia do programa para os outros contêineres, isso se dá por não haver instalado por padrão um programa de sistema de arquivos, como exemplo o servidor Network File System (NFS), que segundo Nix (2011), tem como função o compartilhamento de arquivos por meio da rede de comunicação, permitindo o acesso de outras maquinas a arquivos, como se estes estivessem em seus próprios arquivos locais. Por isso é necessário copiar para a pasta project de todos os nós. Basta utilizar a conexão SSH para realizar a cópia para os demais nós. Primeiro digita-se o comando para ver o IP dos nós. Depois digita-se scp seguido do nome do arquivo que se deseja copiar e por fim mpi@ o ip e o diretório.

```
9c26da5e3a72:/project$ get_hosts
9c26da5e3a72:/project$  scp teste mpi@10.0.9.11:/project
```

Só então é possível executar o programa, também pode ser usada uma outra alternativa para resolver o problema, sendo ela à adição do comando de compilação no arquivo Dockerfile localizado no diretório cluster, além da cópia do arquivo com o código-fonte para a pasta

project do projeto, dessa forma, evita-se a perda de tempo, pois caso sejam inicializados muitos contêineres, será necessário um bom tempo para copiar o programa manualmente para todos os nós.

```
RUN mpicc -o mpi_hello_world  mpi_hello_world.c \
&&mpicc -o teste  teste.c
```

Outro problema encontrado, é que o contêiner mestre acaba recebendo dois IPs, e isso impede que alguns programas MPIs funcionem corretamente. Como solução encontrada, basta realizar a exclusão de um endereço IP, sendo este o único que não segue uma sequência, por exemplo: se houverem quatro IPs com final 8, 15, 16 e 17, o IP de final 8 deverá ser excluído. Por padrão o Alpine possui o editor de texto Vi porem é possível instalar outro. E para realizar a exclusão use o editor para acessar o arquivo contendo todos os IPs, no diretório /etc/opt/hosts. Após realizada a exclusão tudo funcionará normalmente.

```
$ mpirun -np 4 ./teste
```

3.2 Testes de desempenho

Apenas para verificar o funcionamento do cluster, foi realizado um pequeno teste envolvendo este mesmo programa citado acima, utilizando uma, duas, três e quatro máquinas, apenas para verificar se haveria algum ganho no tempo de execução. Em todos os testes foram instanciados vinte nós, e 1000 processos, e os resultados obtidos se encontram na tabela 1.

Tabela 1 - Tempos obtidos

Número de maquinas	Tempo
1	7.58 Segundos
2	6.2 Segundos
3	4.9 Segundos
4	3.48 Segundos

Os resultados obtidos demonstram um ganho significativo no tempo de execução da tarefa, comprovando que o cluster está operando normalmente, apesar de suas limitações, por se tratar de uma virtualização.

Foi realizado um outro teste utilizando um programa modificado por Nordlie (2003) e nomeado por ele de flop.c, que realiza o cálculo do pi e calcula o desempenho do cluster em Megaflops, os resultados não retratam o desempenho real, pois o cálculo do pi não realiza nenhum cálculo complexo, e por tanto não deve ser considerado um guia definitivo para realizar uma medição mais precisa de desempenho.

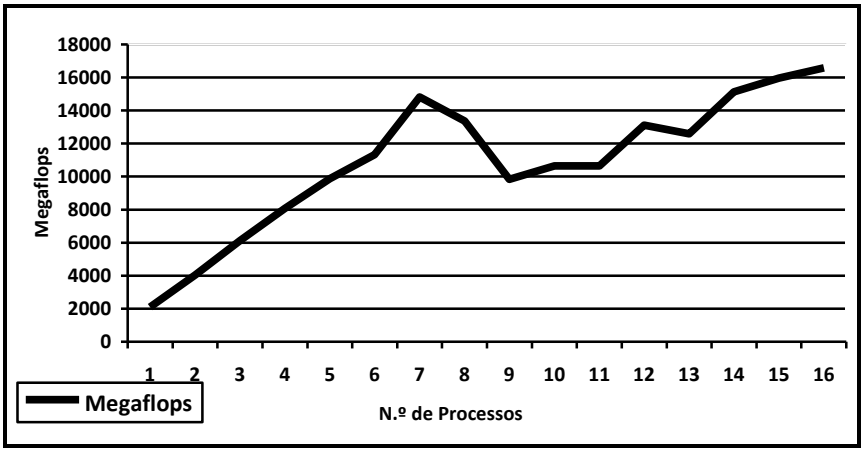
O teste consistiu em executar o flop.c, de um a dezesseis processos, repetindo cada teste três vezes e realizado a média aritmética, foi feito dessa forma apenas para se ter um resultado mais consistente, pois o resultado possuía uma variação considerável.

Tabela 2 - Tempos obtidos do teste flop.c

Número de processos	Megaflops	Tempo
1	2104.408	2.851
2	4042.297	1.484
3	6122.928	0.979
4	8062.173	0.744
5	9873.825	0.607
6	11329.179	0.529
7	14814.519	0.405
8	13361.230	0.449
9	9830.342	0.610
10	10935.476	0.548
11	10649.311	0.563
12	13104.087	0.457
13	12589.546	0.476
14	15120.620	0.396
15	15958.532	0.375
16	16579.445	0.361

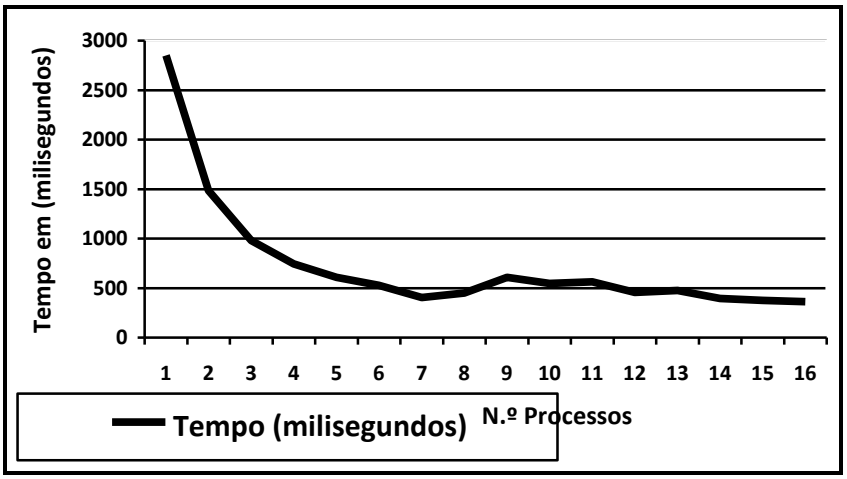
Os resultados obtidos presentes na Tabela 2 acima, apontam um aumento constante de cerca de 2 Gigaflops em média para cada novo processo adicionado ao teste, até o com sete processos, seguido de uma queda nos dois testes seguintes e uma certa tendência a se estabilizar até o teste com treze processos, e tendo o seu ápice com dezesseis processos, onde alcançou cerca de 16.6 Gigaflops.

Figura 1 - Gráfico dos resultados obtidos do teste flop.c (Processos x Megaflops)



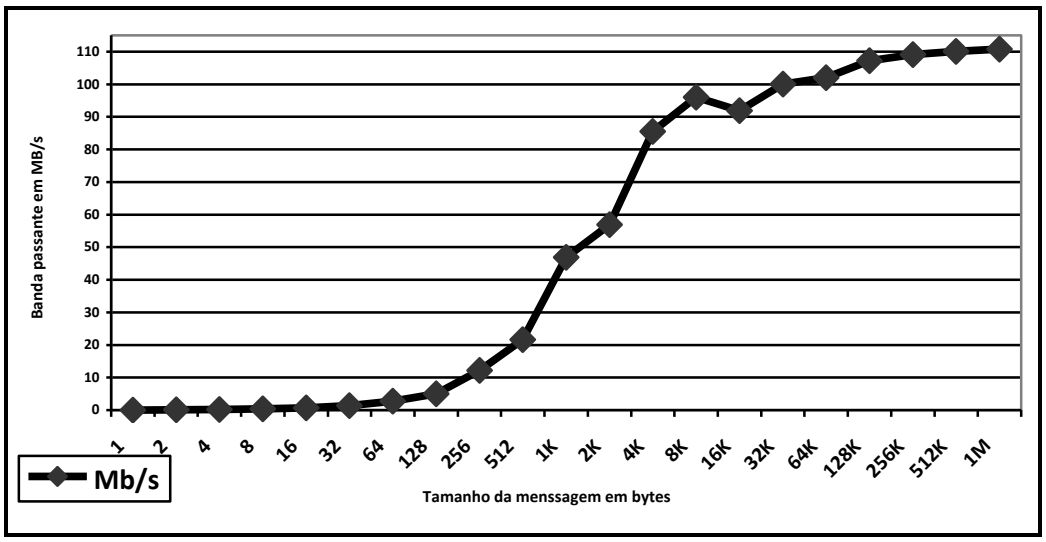
A figura 1 acima mostra o gráfico dos resultados obtido em relacionado o tempo ao número de processos em execução, sendo possível visualizar com mais clareza o escalonamento de desempenho obtido até o sétimo teste, quando é possível ver uma queda seguida de uma parente estabilização dos resultados.

Figura 2 - Gráfico dos resultados obtidos do teste flop.c (Proxessos x Tempo)



Também foi gerado um gráfico relacionando o tempo de execução ao número de processos rodando representado pela figura 2, sendo possível observar mais uma vez uma brusca queda no tempo de execução até o sétimo teste, demonstrando a constância no aumento de desempenho em relação ao número de processos, seguido de uma estabilização nos tempos obtidos, tendo eles pouca diferença um em relação ao outro.

Figura 3 - Gráfico dos resultados obtidos do teste ping_pong.c



Foi realizado um terceiro teste com o programa ping_pong.c, e gerado um gráfico, representado pela figura 3, o mesmo serve para analisar o desempenho de rede do cluster. A partir dos resultados obtidos, é possível perceber um aumento na taxa de transferência entre o envio de 128 bytes a 16Kb, seguindo por uma estabilização no fluxo de mensagens.

4. Conclusão

Foi estudado e implementado neste trabalho, um modelo de cluster do tipo Beowulf utilizando contêineres Docker desenvolvido por Nguyen e Bein, com o objetivo de possibilitar aos alunos de computação paralela e distribuída, terem um maior contato com programação paralela, e buscando um ambiente o mais próximo possível do real. Com mais exemplos práticos em sala, os acadêmicos podem se interessar mais pela área de conhecimento. Sendo

essa uma área de pesquisa importante, pois segundo Corrêa (2016), clusters de alto desempenho são utilizados para ajudar a resolver problemas complexos.

Sendo assim, é importante que os alunos tenham um maior contato com essa tecnologia, e aprendam a desenvolver projetos de computação paralela. Porém, nem sempre a universidade dispõe de recursos para montar laboratórios exclusivos para que sejam implementados clusters, e não podem "desfigurar" os laboratórios existentes, pois na maioria das vezes são para dar suporte as disciplinas.

Foi possível, a partir dos testes realizados com o programa flop.c, que após um certo número de processos, o desempenho passa a se estabilizar, mantendo uma pequena diferença entre cada execução realizada.

Por conta da simplicidade, portabilidade e compatibilidade oferecida pela ferramenta Docker, é possível implementar esse projeto em pouco tempo, se comparado a outros modelos já citados. E apesar de ser tecnicamente um cluster virtual, ele oferece um desempenho.

O presente trabalho conseguiu atingir vários objetivos, dantes estipulados, mesmo com as tentativas e falhas, foi possível aplicar os conhecimentos aprendidos em sala, além de aprender muito durante a realização da pesquisa do mesmo.

Com a implementação foi possível criar um ambiente onde os alunos possam aplicar os conceitos aprendidos na disciplina, podendo ter um contato direto com programação paralela por meio de um cluster virtual, com características muito próximas de um modelo real.

O modelo de cluster utilizado, trouxe grande aprendizado ao acadêmico, que estudou o projeto minuciosamente, para que pudesse descrever cada etapa da implementação, buscando explicar de uma maneira clara todos os processos envolvidos.

Como trabalhos futuros, pode-se utilizar as ferramentas Docker disponíveis, para implementar clusters de outros tipos, usando o projeto utilizado neste trabalho como base, além de adicionar diretamente no arquivo Dockerfile, instalação de ferramentas didáticas de Banchmark para avaliação do desempenho, bem algum gerenciador de sistema de arquivos, como por exemplo o NFS, evitando o problema da necessidade de se copiar o programa MPI que se deseja executar para todos os contêineres do cluster, facilitando o uso.

Referências

CARNEIRO, Lucas Santana; DUARTE, Angelo Amancio. **Differences in the Performance of Parallel Applications in Physical and Virtual Clusters.** Ieee Latin America Transactions, [s.l.], v. 16, n. 2, p.604-612, fev. 2018. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tla.2018.8327419. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8327419&tag=1. Acesso em: 28 jun. 2019.

CORRÊA, José Norberto Guiz Fernandes. **LIFTER:** disponibilização de aplicações via *container*s de *software* em um *cluster* de alto desempenho. 93 f. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Universidade Federal de Santa Catarina, Florianópolis, 2016. Disponível em: https://repositorio.ufsc.br/xmlui/bitstream/handle/123456789/171410/TCC_Lifter_JoseNorberto.pdf?sequence=1&isAllowed=y. Acesso em: 03 mar. 2019.

LUDIN, Mobeen et al. **LittleFe**: The high performance computing education appliance. In: IEEE INTERNATIONAL CONFERENCE ON *CLUSTER* COMPUTING (*CLUSTER*). IEEE, 2013. p. 1-1. Disponível em: https://ieeexplore.ieee.org/document/6702649. Acesso

em: 04 jun. 2019.

NGUYEN, Nikyle; BEIN, Doina. **Distributed MPI Cluster with Docker Swarm Mode**. In: ANNUAL COMPUTING AND COMMUNICATION WORKSHOP AND CONFERENCE (CCWC), 7., 2017, Las Vegas. NV**.** Las Vegas: Ieee, 2017. p. 1 - 7. Disponível em: https://ieeexplore.ieee.org/document/7868429. Acesso em: 03 jun. 2019.