Impacto do uso de contêineres em um serviço de nuvem HTC

Carlos H. Z. Nicodemus[1], **Cristina Boeres**[1], **Vinod E.F. Rebello**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói, RJ – Brasil

{cnicodemus,boeres,vinod}@ic.uff.br

Abstract. *Container-based virtualization has been gaining significant popularity in recent years by offering isolated virtual environments, with an efficient allocation of computing resources, to applications running on shared systems. This study aims to verify if the improved performance gains are always possible by comparing the use of containers and virtual machines for the execution of a widely used biodiversity application in the context of a cloud service. In particular, the scalability of these virtual environments to support High Throughput Computing is analysed.*

Resumo. *A virtualização por contêineres vem ganhando popularidade nos últimos anos devido à criação de ambientes virtuais isolados com uma alocação eficiente de recursos computacionais para a execução de aplicações considerando o compartilhamento de recursos. Este trabalho tem por objetivo verificar se esta melhoria é sempre possível através da comparação do uso de contêineres e máquinas virtuais para a execução de uma aplicação científica de biodiversidade usado em um serviço de nuvem. Em particular, sua eficiência na escalabilidade desses ambientes virtuais em Computação de Alta Vazão é analisada.*

1. Introdução

Um dos maiores problemas encontrados por pesquisadores dentro de universidades e centros de pesquisas é o compartilhamento de recursos computacionais entre vários usuários. A execução de aplicações de alto desempenho, geralmente aplicações paralelas ou distribuídas, requer grandes quantidades de recursos computacionais (processamento, memória e/ou armazenamento) que usualmente não estão disponíveis a todo momento. Além do mais, tais usuários almejam executar suas aplicações de forma dedicada, para que seus resultados e desempenho não sejam influenciados por outros processos. No entanto, ao considerar a execução de diferentes aplicações, um outro problema impactante é a compatibilidade das ferramentas disponibilizadas no ambiente (bibliotecas, sistema operacional, serviços), podendo existir conflitos que chegam até a inviabilizar a execução no ambiente considerado.

Atualmente, a criação de máquinas virtuais vai de encontro com a forma de reservar recursos computacionais dentro de um ambiente compartilhado para assim evitar conflitos entre as diversas aplicações, criando um ambiente isolado que contém uma certa quantidade de recursos limitados, bem como um ambiente compatível com cada aplicação [Pearce et al. 2013].

A tecnologia de virtualização mais utilizada nestes centros de pesquisa, assim como no mercado, é a virtualização completa, utilizando hipervisores como

Xen [Xen 2016], VMware [VmWare 2016] e KVM [KVM 2016]. Esta tecnologia permite criar ambientes virtuais completos, simulando máquinas reais, com sistema operacional próprio, recursos computacionais reservados e serviços específicos. É importante salientar que os recursos reservados nestes ambientes virtuais não são utilizados somente pela aplicação do usuário, mas também pelo próprio SO e até serviços. Assim, é notório atrasos na execução em ambientes virtuais quando comparados com o tempo de sua execução direta na máquina física [Felter et al. 2015]. Por outro lado, nos últimos anos, vem crescendo a utilização de virtualização por contêineres como uma alternativa aos *hipervisores*, prometendo uma alocação de recursos mais eficiente, com pouco ou nenhum atraso na execução de aplicações e também o mesmo nível de isolamento provido pelas máquinas virtuais [Felter et al. 2015].

Objetivando apontar as características de um ambiente virtual para usuários de aplicações de alto desempenho que necessitam de um ambiente exclusivo, mas ao mesmo tempo possuem recursos físicos limitados, este trabalho almeja analisar e comparar as tecnologias de virtualização completa e contêineres, para que seja possível indicar uma alternativa adequada para a criação de nuvens computacionais privadas em ambientes com poucos recursos. O intuito é estabelecer em qual ambiente as aplicações científicas de alto desempenho poderão se beneficiar simultaneamente, com uma menor sobrecarga sobre o tempo de execução e fornecendo um ambiente isolado e adequado para a aplicação.

O artigo está organizado da seguinte forma: Seção 2 apresenta alguns trabalhos relacionados enquanto a Seção 3 faz um comparativo entre virtualização por contêiner e completa. Na Seção 4, os resultados de experimentos são apresentados com o objetivo de avaliar a eficiência da virtualização por *contêiner* em comparação com a virtualização completa e a execução direta em máquinas físicas, avaliando também a escalabilidade de recursos. Para tal, uma aplicação científica contendo diversos algoritmos para a modelagem de nichos ecológicos, o OpenModeller [de Souza Munoz et al. 2011], é usada. Na última seção, as conclusões e trabalhos subsequentes são apresentados.

2. Trabalhos Relacionados

As tecnologias de virtualização vem sofrendo alterações ano a ano, onde as suas evoluções levam a uma melhor eficiência e novas tecnologias despontam no mercado. O artigo [Xavier et al. 2013] compara as tecnologias de contêineres como LXC e OpenVZ com a virtualização completa Xen, executando o *benchmark* IOZone para análise de I/O e *benchmarks* de aplicações HPC, e conclui que virtualização de contêineres leva a um melhor desempenho, sendo que para as aplicações consideradas, seu desempenho foi próximo de uma máquina nativa com as tecnologias de contêineres avaliadas. Em contrapartida, apontam que o isolamento ruim e a falta de segurança são os pontos negativos dos contêineres.

Felter *et. al.* em [Felter et al. 2015] compararam o desempenho de virtualização completa e em contêineres e também com Linux nativo, utilizando KVM e o Docker, respectivamente. O estudo de caso adota a execução de uma aplicação MySQL, objetivando avaliar o desempenho das tecnologias quanto ao uso de CPU, memória, rede e I/O. O trabalho concluiu que a implementação com o Docker apresentou melhores resultados, mas enfatizou que os experimentos foram configurados de forma que a aplicação MySQL teve memória suficiente e praticamente nenhum acesso a disco foi realizado.

Trabalhos como [Soltesz et al. 2007] e [Babu et al. 2014] também avaliam o de-

sempenho das tecnologias de virtualização utilizando *benchmarks* ou aplicações específicas, como em [Xavier et al. 2014], que comparou contêineres e máquinas virtuais para ambientes que suportem problemas de MapReduce. Tais artigos identificaram vantagens de utilização de contêineres com relação ao desempenho. No entanto, é difícil identificar o impacto do aumento da escalabilidade de recursos virtuais nos ambientes.

Será que o uso de contêineres sempre vai fornecer um melhor desempenho? Nosso trabalho diferencia destes artigos ao avaliar o comportamento e o desempenho das tecnologias de virtualização completa e de contêineres utilizando uma aplicação científica real, com a avaliação da escalabilidade de tais tecnologias em um ambiente com dados compartilhados, análise esta que não foi observada em trabalhos correlatos. O aumento de recursos virtuais vem de encontro com o comportamento de um ambiente compartilhado como a nuvem computacional, ao executar múltiplas instâncias da mesma aplicação, gerando uma maior carga no ambiente.

3. Virtualização usando *contêineres*

Na última década, as tecnologias de virtualização vem evoluindo e ganhando popularidade com a oferta de um ambiente virtualizado capaz de simular uma máquina real, compartilhando recursos computacionais dos servidores entre múltiplos usuários. Empresas e centros de pesquisa que oferecem um ambiente de nuvem computacional como serviço, utilizam a tecnologia de virtualização completa como base para sua infraestrutura. Em geral, hipervisores são utilizados para a criação de máquinas virtuais (MVs) para os clientes, máquinas estas consideradas como recursos computacionais reservados, sendo cobrado ao usuário o tempo em que cada MV ficou ativa.

Porém nos últimos anos, uma outra tecnologia de virtualização vem ganhando popularidade: a virtualização por contêineres. Estes contêineres podem ser vistos como MVs simplificadas, que compartilham o núcleo do sistema operacional com o seu servidor físico, tendo apenas as aplicações do usuário a serem executadas e as bibliotecas e ferramentas necessárias, independentes do servidor. A seguir, as principais diferenças entre estas tecnologias de virtualização e como é possível aproveitar as características de um contêiner para aplicações HPC são discutidas.

3.1. Arquitetura

A arquitetura de um sistema de virtualização por contêineres é baseado no isolamento das aplicações, serviços e bibliotecas necessários. As demais funções relacionadas ao sistema operacional (SO) ou especificamente, ao seu núcleo, são compartilhados entre contêineres, sendo executados em um mesmo servidor.

Na Figura 1, podemos observar como a arquitetura de contêineres diferencia da arquitetura de virtualização completa. Uma MV é composta por um sistema operacional (*SO MV*) independente do sistema operacional do hospedeiro (*SO Hospedeiro*), além das bibliotecas, ferramentas e serviços que compõem um ambiente virtualizado completo. Desta forma, é possível executar um SO diferente do SO hospedeiro dentro da MV, como por exemplo, uma máquina virtual Windows dentro de um servidor Linux.

Na arquitetura de contêineres, há o compartilhamento do núcleo e de bibliotecas do SO Hospedeiro entre os vários contêineres em uma mesma máquina física. Deste modo, dependendo do SO Hospedeiro, é possível executar contêineres compatíveis com

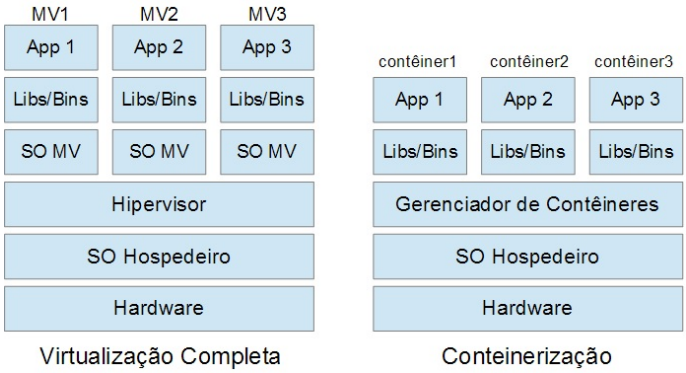

Figura 1. Virtualização Completa x Virtualização por Contêineres

este, como por exemplo, contêineres Ubuntu dentro de um hospedeiro CentOS, mas não é possível executar um contêiner Windows em um hospedeiro Linux. O gerenciamento do ambiente de contêineres é realizado pelo *Gerenciador de Contêineres*, responsável pela criação e execução destes. As aplicações do usuário a serem executadas nos contêineres são vistas pelo SO Hospedeiro como processos. Sendo assim, não há necessidade de modificação do núcleo do SO Hospedeiro para executar suas funções de gerenciamento [Docker 2016].

Já no caso de virtualização completa, os hipervisores são responsáveis pelo gerenciamento das MVs e também, há a necessidade de modificação do núcleo do SO Hospedeiro (realizada durante a instalação dos hipervisores), quando módulos responsáveis ao acesso aos recursos de hardware e funcionalidades do SO Hospedeiro são adicionados. Esta diferença também influencia no hardware, pois para o contêiner, não é necessário que o hardware do hospedeiro tenha suporte a virtualização, enquanto o hipervisor necessita, como o Intel VT [Intel 2016] e AMD-V [AMD 2016].

3.2. Armazenamento em Disco

As MVs tradicionais armazenam seu SO, suas bibliotecas e aplicações dentro de drives virtuais no hospedeiro, chamados de *imagens de disco*, cujo formato pode variar de acordo com o hipervisor utilizado [Su et al. 2010]. Esta imagem funciona como um sistema de armazenamento isolado de cada MV e permitem que os dados armazenados dentro de uma MV não sejam acessíveis pelo SO Hospedeiro. Este arquivo é definido no momento de criação da MV, quando é especificado o seu formato, tamanho máximo, assim como o método de atualização e a persistência de seus dados.

Esta imagem de disco pode ocupar grandes quantidades de espaço no disco da máquina hospedeira, como um único arquivo ou dividido em arquivos menores. Ainda, na etapa de criação da MV, a sua imagem de disco pode ocupar todo o espaço definido se for definido como fix-sized file ou ocupar incrementalmente durante a vida da MV se copy-on-write file [Su et al. 2010] for utilizada. Nos experimentos apresentados mais tarde, todas as MVs foram criadas com tamanho fixo especificadas através de fix-sized file.

Os contêineres utilizam uma imagem como forma de armazenamento dos seus dados. Estas imagens possuem as bibliotecas e serviços necessários para a execução de uma aplicação e são organizadas em camadas, como observado na Figura 2. As camadas

de uma imagem possui um hash criptografado para identificá-las e são utilizadas pelo gerenciador de contêiner para o endereçamento de seu conteúdo. Cada camada de uma imagem é definida somente para leitura, o que permite que uma mesma imagem seja compartilhada por mais de um contêiner, reduzindo o espaço necessário para armazená-los.

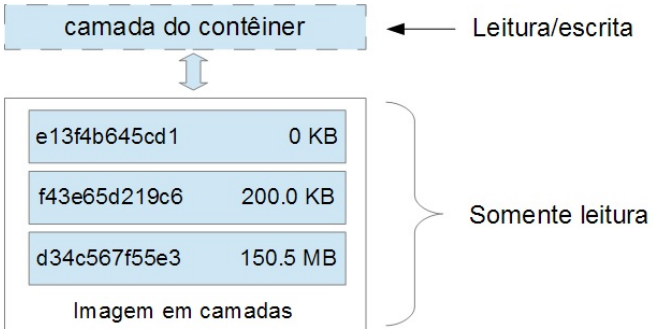

Figura 2. Armazenamento em Contêineres

Quando um contêiner é inicializado, o gerenciador de contêineres cria uma nova camada, a *camada do contêiner*, com propriedades de leitura e escrita em disco, onde serão salvas todas as alterações feitas pelo contêiner. Esta camada é individual para cada contêiner, mesmo que compartilhem uma mesma imagem base, e persiste durante todo seu ciclo de vida. Quando queremos criar uma atualização de uma imagem, é a camada de leitura e escrita que será adicionada à imagem base utilizada e uma nova cópia da imagem será gerada, em um processo chamado *commit* da imagem, como em um banco de dados.

3.3. Alocação de recursos de CPU e Memória

Na virtualização completa, a alocação de recursos computacionais para uma MV é feita através de sua reserva, utilizando-se drives virtuais para simular o hardware da máquina hospedeira. O hipervisor controla e reserva os recursos computacionais para cada MV, utilizando funções do núcleo do SO Hospedeiro e os conjuntos de instruções de virtualização de hardware oferecido pelo Intel VT ou o AMD-V, por exemplo.

Nos contêineres são utilizadas funcionalidades do núcleo do SO Hospedeiro para a criação de um espaço isolado para a execução de suas aplicações, sendo esse espaço definido por um *namespace* [Xavier et al. 2013]. Os recursos computacionais de um contêiner são alocados conforme a necessidade da aplicação que está sendo executada no contêiner e de acordo com as funcionalidades de escalonamento do SO hospedeiro.

Por padrão, os contêineres não possuem limitação de recursos para um processo, só a quantidade de recursos disponíveis no hardware do hospedeiro. Porém é possível limitar a quantidade de recursos computacionais aos quais um contêiner pode ter acesso, utilizando o recurso de Linux *control groups* (*cgroups*). Com *cgroups*, é possível limitar os recursos que um determinado *namespace* tem acesso, como o número de CPUs, a quantidade de memória e I/O que um contêiner pode utilizar, uma vez que suas aplicações executam dentro do seu *namespace*.

4. Experimentos Realizados

Como estudo de caso, utilizamos uma importante aplicação da área de biodiversidade, devido sua caraterística de ter uma grande quantidade de dados a serem processados porém ter estes compartilhados entre todos as instancias da aplicação que fazem parte do experimento científico.

A modelagem de nichos ecológicos – ENM (*Ecological Niche Modelling*) – é um procedimento comum para determinar a extensão da distribuição geográfica das espécies, sendo poderoso para prever a potencial distribuição de espécies em diferentes contextos geográficos e temporais, assim como para estudar outros aspectos da evolução e ecologia. ENM tem sido bastante usado em várias situações, como por exemplo: busca de espécies raras ou ameaçadas de extinção; identificação de regiões adequadas para a (re)introdução de espécies; previsão do impacto das mudanças climáticas sobre a biodiversidade; definição e avaliação de áreas protegidas; prevenção da propagação de espécies invasoras, entre outras aplicações importantes.

Aplicações de ENM combinam informações sobre a ocorrência de espécies com bases de dados ambientais na forma de camadas rasterizadas geo-referenciadas (tais como temperatura, precipitação e salinidade) para gerar potenciais modelos de distribuição. O openModeller (OM) é uma ferramenta para ENM desenvolvida pelo Centro de Referência em Informação Ambiental (CRIA), junto com outros parceiros nacionais e internacionais [de Souza Munoz et al. 2011], bastante difundida entre a comunidade biológica e ecológica [Geller and Melton 2008, Zarco-González et al. 2013]. É provável que esta aplicação desempenhe um papel fundamental para que países, e especialmente o Brasil, cumpram seus acordos com as Nações Unidas no que diz respeito à avaliação da biodiversidade da Terra até 2020. Para ajudar nesta meta, o instituto dos autores deste trabalho disponibiliza um serviço de ENM em nuvem para a comunidade científica.

Nos experimentos, executamos as três etapas do OM sequencialmente: Modelagem através do *om_model*, para geração de modelos de distribuição a partir de um arquivo de requisição em XML contendo as informações necessárias para sua execução (algoritmo de modelagem, pontos de localização das espécies e camadas ambientais); Teste através do *om_test*, que avalia os modelos de distribuição gerados na etapa anterior, verificando se os pontos de entrada possuem as mesmas referências espaciais dos pontos gerados na modelagem; e Projeção através do *om_project*, que utiliza o modelo gerado pelo *om_model* e os *templates* utilizados para a geração de um mapa de distribuição e o resultado é armazenado em um arquivo de imagem, para facilitar a visualização dos resultados da modelagem. O OM possui diversos algoritmos de modelagem, entre os quais utilizamos em nossos experimentos os seguintes: o algoritmo GARP-BS [Stockwell 1999], uma implementação de um algoritmo genético para criação de modelos de nichos ecológicos; o algoritmo MAXENT [Phillips et al. 2004], um algoritmo de aprendizado de máquinas para calcular a probabilidade de distribuição epistêmica; e o algoritmo ENVDIST, um algoritmo genético baseado em métricas de dissimilaridades ambientais.

O ambiente experimental utilizado consiste de um servidor com dois processadores Xeon hexacore X5650 com 2.67GHz 12 cores e com o *hyperthreading* ativado, gerando 24 cores virtuais, contendo 24 GB de memória RAM DDR3 a 1333MHz e 2 TB de armazenamento em disco. O SO hospedeiro é o CentOS Linux 7.2 64 bits com kernel 3.10.0. Foi utilizado o hipervisor Qemu/KVM 1.5.3 [KVM 2016] para a criação de MVs e

para a criação dos contêineres, o Docker 1.11. Foi disponibilizado um conjunto de 32 GB de dados para as *camadas ambientais*, informações sobre o ambiente em que as espécies vão ser projetadas, utilizando o OM. Para as máquinas virtuais (MVs), estes dados foram fornecidos através do protocolo *Network File System* (NFS v3) com o hospedeiro e, para os contêineres, foi utilizado um mecanismo de mapeamento direto de diretórios entre o contêiner e o hospedeiro.

As máquinas virtuais criadas utilizam um núcleo virtual (vCPU), 1 GB de RAM e 9 GB de armazenamento em disco. O SO da MV é o CentOS Linux 7.2 64 bits, igual ao do hospedeiro. Para o contêiner, a configuração padrão do Docker foi utilizada, que aloca um núcleo por aplicação ativa do contêiner e sem reservas de memória na criação, sendo a quantidade de memória RAM utilizada àquela necessária pela aplicação do usuário. Uma imagem do Linux CentOS 7.2 foi utilizada como base, sendo adicionado o OM e as bibliotecas necessárias para sua execução (do próprio OM e OpenMPI).

O experimento inicial compara os tempos de criação de um recurso virtual (MV ou contêiner), ou seja, o tempo necessário para que o respectivo recurso virtual se torne disponível para a aplicação com seu respectivo espaço de armazenamento. Ainda, testes comparativos de desempenho entre contêineres, MVs e a execução direta no hospedeiro foram executados. Nestes experimentos, cada execução do OM corresponde às três etapas descritas anteriormente e consiste de três processos executados estritamente em sequência dentro de um mesmo recurso virtual (MV ou contêiner). Finalmente, para avaliar a escalabilidade das tecnologias, um *script* na linguagem C e OpenMPI [OpenMPI 2016] foi especificado para criar 1, 4, 8, 12, 16, 20 e 24 recursos virtuais (MVs e contêineres) na mesma máquina física. Cada execução foi realizada 10 vezes e a média dos tempos foi obtida para a validação e análise.

4.1. Teste de Disponibilidade

Para análise dos tempos de disponibilidade, o experimento compara os tempos de criação de uma máquina virtual (a partir da imagem de uma MV existente) e de um contêiner (também a partir de uma imagem equivalente ao sistema existente na MV sendo comparada). A criação da MV foi realizada utilizando o comando virt-clone() para o KVM a partir do CentOS e o tempo desta criação foi medido. No caso do contêiner, ele se torna ativo somente se uma aplicação é executada no referido contêiner e, para coletar tal tempo de disponibilidade, foi necessário executar um processo simples, por exemplo, ”listagem de diretório”e logo depois de sua execução, o contêiner é finalizado.

A média dos tempos obtidos na clonagem de MVs foi de 29,95 segundos, com um desvio padrão de 8,67, enquanto a média dos tempos obtidos na criação, execução do comando e finalização do contêiner foi de 2,72 segundos com um desvio padrão de 0,31. Os resultados mostram que um contêiner pode ser criado e disponibilizado muito mais rápido do que o tempo para se clonar uma máquina virtual equivalente.

Isso ocorre devido a natureza do formato de armazenamento de ambos recursos de virtualização, pois cada MV possui um arquivo de disco robusto, de tamanho previamente fixado e será totalmente copiado para uma nova MV (processo de clonagem). Enquanto em um contêiner, sua imagem não precisa ser clonada e é apenas gerado uma nova camada de armazenamento para os novos dados que poderão ser gerados no novo contêiner. Esta estrutura de disco em camadas permite que um contêiner ocupe menos espaço e diminui

drasticamente o tempo necessário para que um novo contêiner esteja pronto para uso.

A clonagem de MVs foi utilizada ao invés de criar uma MV a partir do zero, caso em que o tempo de criação seria muito maior, o que não encaixaria em uma proposta de nuvem computacional. Imagens de máquinas virtuais previamente criadas se encontram disponibilizadas para o rápido provisionamento do serviço para o usuário.

4.2. Avaliação de Desempenho

Este experimento tem como intuito avaliar o desempenho de ambientes com MVs e com contêineres, comparando também com a execução direta no hospedeiro para aplicações científicas. Utilizamos o OpenModeller como aplicação de caso de estudo, selecionando três de seus algoritmos para esta avaliação: GARP, MAXENT e ENVDIST.

Tabela 1. GARP: Média dos tempos e desvio padrão das três etapas

GARP (tempo em segundos)			
	Modelagem	Teste	Projeção
Máquina Física	655.128 ± 2.50	54.186 ± 0.96	291.429 ± 32.65
Máquina Virtual	690.563 ± 3.07	81.220 ± 2.43	313.727 ± 28.60
Contêiner	654.305 ± 2.43	53.686 ± 0.65	291.967 ± 28.95

Observando Tabelas 1, 2 e 3, a média dos tempos do contêiner, em cada uma das três etapas nos três algoritmos, foi próxima da média de tempo de execução na máquina física, sendo ligeiramente melhor, (na casa dos milissegundos) para as etapas de Modelagem e Teste. Isso ocorre devido à execução de um processo no contêiner ser equivalente a execução de um processo da própria máquina hospedeira, pois suas aplicações possuem apenas um isolamento lógico ocasionado pelo uso de *namespaces* nos contêineres.

Tabela 2. MAXENT: Média dos tempos e desvio padrão das três etapas

MAXENT (tempo em segundos)			
	Modelagem	Teste	Projeção
Máquina Física	61.557 ± 0.52	53.116 ± 0.75	35.505 ± 0.18
Máquina Virtual	90.785 ± 3.02	81.156 ± 2.85	40.135 ± 0.27
Contêiner	61.490 ± 0.60	52.907 ± 0.44	35.835 ± 0.48

Já na MV, os tempos obtidos na etapa de Modelagem e Teste, que exigem extensas leituras de arquivos de entrada do OpenModeller, foram até 55,2% piores na média para o caso da modelagem utilizando o algoritmo ENVDIST e 52,8% na etapa Teste do algoritmo MAXENT, algoritmos considerados menos exigentes quanto ao processamento, enquanto no GARP, esta perda foi de 5,4% na etapa de Modelagem e quase 50% durante o Teste. Isso pode ocorrer devido às características da virtualização completa, realizando verificações de segurança nas instruções antes de executá-las, o que gera atrasos no tempo de execução nas aplicações em comparação com a mesma aplicação sendo executada diretamente no hospedeiro. Em algoritmos mais rápidos, perde-se muito tempo realizando tais verificações, enquanto em algoritmos mais complexos, tal verificação não impacta tanto no tempo final de execução de uma aplicação. Podemos observar também que durante a Projeção, a perda com relação a máquina física ficou em torno de 13% no pior caso (algoritmo MAXENT), validando a mesma conclusão tirada em relação às outras etapas.

Tabela 3. ENVDIST: Média dos tempos e desvio padrão das três etapas

ENVDIST (tempo em segundos)			
	Modelagem	Teste	Projeção
Máquina Física	52.561 ± 0.07	53.080 ± 0.71	30.968 ± 0.42
Máquina Virtual	81.589 ± 3.17	78.939 ± 2.87	34.499 ± 0.40
Contêiner	52.695 ± 0.21	52.607 ± 0.11	31.081 ± 0.27

Logo, para a execução de uma aplicação, em um ambiente isolado do hospedeiro, o contêiner mostrou-se tão eficiente quanto a própria máquina física, sem gerar grandes atrasos para a aplicação do usuário e sobrecarga do próprio sistema do contêiner.

4.3. Avaliação de Escalabilidade

Como aplicações de alto desempenho executadas em pesquisas científicas possuem como característica a possibilidade de processar grandes volumes de dados paralelamente, usando ao seu favor clusters e nuvens computacionais para redução do tempo de execução da aplicação, é imperativo a análise da escalabilidade das plataformas de virtualização.

Para tal, o algoritmo GARP do OpenModeller foi utilizado por ser o algoritmo mais exigente quanto ao consumo de recursos computacionais entre os três algoritmos testados. O experimento foi repetido 10 vezes para 4, 8, 12, 16, 20 e 24 MVs/contêineres, correspondendo a um ambiente paralelo com alta carga de trabalho, onde cada um dos processos equivale a uma execução completa do processo do OpenModeller (Modelagem, Teste e Projeção) sobre o mesmo conjunto de dados (as camadas ambientais). Cada processo é executado em um contêiner ou MV, isolando os processos como se fossem requisições feitas por múltiplos usuários simultaneamente em um ambiente de cluster ou de nuvem. Utilizamos o OpenMPI para inicializar cada instância do processo OpenModeller simultaneamente em todos os contêineres ou MV. Nos três gráficos seguintes, não é possível visualizar as barras de erro por estarem escondidas pelos pontos.

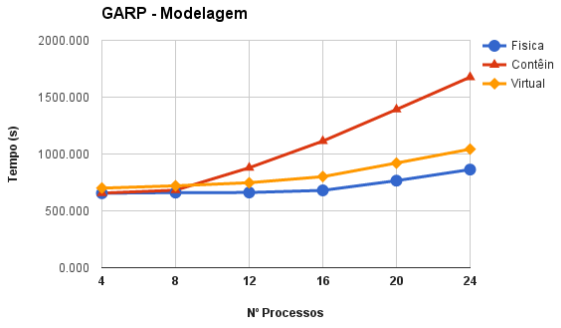

Figura 3. Escalabilidade do GARP para a etapa de modelagem

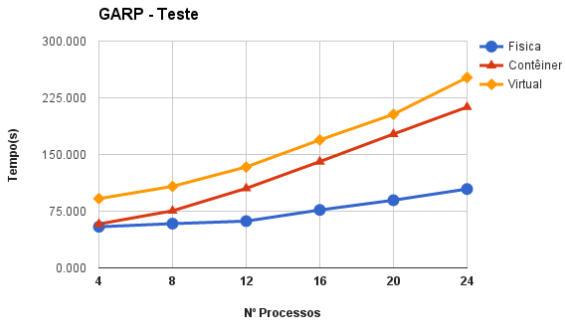

Figura 4. Escalabilidade do GARP para a etapa de teste

No gráfico da Figura 3, pode-se observar que a execução nos contêineres obteve um desempenho similar a execução nativa na máquina física enquanto o número de processos era menor que o número de cores físicos disponíveis no hospedeiro (8 processos), porém ao chegar ao limite de cores físicos (12 cores) e sobrecarregá-los (até 24 cores),

a execução nos contêineres sofreu uma perda significativa sendo até 94% pior do que a execução na máquina física e até 60% pior comparado a execução nas MVs (para 24 cores). Isso ocorre porque, durante a modelagem, o algoritmo do GARP executa diversas iterações do seu algoritmo genético para a obtenção de diversos conjuntos de resultados, selecionando os melhores como saída, o que necessita de muitas operações de acesso a disco para a leitura das camadas ambientais, que são compartilhados por todos os contêineres usando uma função de mapeamento de diretório do Docker. Nas MVs, estas camadas ambientais são compartilhadas pelo hospedeiro através do protocolo NFS, e não gerou uma sobrecarga sobre o acesso ao disco. A diferença de desempenho das MVs, em comparação a máquina física, envolve mais uma sobrecarga no processamento, comum para esta tecnologia de virtualização, devido a diversos fatores, tais como a abstração do hardware do hospedeiro para a MV, a interpretação de instruções do sistema entre MV e hospedeiro, entre outros, como discutido em [Xavier et al. 2013] e em [Felter et al. 2015].

Na Figura 4, podemos ver que o desempenho no contêiner foi melhor do que na MV, porém foi pior (chegando a duas vezes) se comparado a execução na máquina física. Na fase de Teste, há uma verificação se os pontos de ocorrência, utilizados para a geração do modelo, foram devidamente referenciados no resultado da modelagem e são usadas as mesmas camadas ambientais utilizadas na modelagem. Há menos operações de leitura em disco, assim a perda de desempenho envolve mais o processamento, pois houve concorrência de CPUs entre os contêineres, ao não utilizar os cores virtuais disponíveis através do *hyperthreading*. Já as MVs utilizaram os cores virtuais, mas sofreram uma perda de desempenho maior do que os contêineres. Este fenômeno pode ser observado utilizando um comando do sistema Linux chamado htop [Muhammad 2016] que exibe graficamente a utilização dos recursos de processamento (cores físicos e virtuais) além do consumo de memória e processos sendo executados.

Figura 5. Escalabilidade do GARP para a etapa de projeção

A Figura 5 mostra a repetição do comportamento observado na Figura 3, pois para realizar a projeção, o OpenModeller precisa ler um *template*, que um arquivo que contém informações como o tamanho das células e as referências espaciais usados para a geração do mapa de distribuição, correspondente a camada ambiental utilizada na modelagem. Posteriormente tem que gerar uma imagem, o que exige operações concorrentes de leitura sobre um mesmo arquivo (as camadas ambientais e *templates*) e a escrita em disco do arquivo de imagem do processo de cada contêiner.

Podemos concluir que o gerenciamento de contêineres do Docker não foi capaz de tratar eficientemente as múltiplas operações concorrentes de acesso ao disco sobre dados compartilhados, assim como não utilizou os cores virtuais disponíveis pelo *hyperthreading*, gerando concorrência por processamento entre os processos, enquanto a execução na MV sofreu uma perda de desempenho já apontada por outros trabalhos como os estudados neste artigo, para aplicações com processamento intensivo.

5. Conclusão e Trabalhos Futuros

Com objetivo de obter uma melhora de desempenho em nosso serviço de nuvem que suporta a execução concorrente de experimentos de ENM feitos pela comunidade de biodiversidade, neste trabalho, investigamos e comparamos uma implementação nova com o uso de contêineres e uma existente de máquinas virtuais para a execução da aplicação openModeller em um ambiente com recursos compartilhados.

Observamos que os contêineres ocupam um menor espaço em disco, do que as MVs, ao utilizar uma imagem compartilhada em sua criação e somente gravar alterações feitas a esta imagem, reduzindo consideravelmente o tempo necessário para que o ambiente virtual esteja disponível. Ainda, a tecnologia de contêiner se mostrou mais eficiente do que as MVs quanto a execução unitária de uma aplicação científica, com resultados próximos aos obtidos com a execução na máquina física. Porém o mesmo desempenho não foi observado quando, gradativamente, a quantidade de trabalho realizado é escalada. Contêineres sofreram uma grande perda de desempenho com esta aplicação científica que exige grandes quantidades de operações de acesso a disco sobre dados compartilhados.

Como trabalhos futuros, pretendemos avaliar formas de compartilhamento de dados entre contêineres, utilizando outras tecnologias, assim como avaliar se este problema é exclusivo do Docker ou se ocorre em outras tecnologias de virtualização. É importante também analisar a influência do *hyperthreading* em tais ambientes para a execução de aplicações científicas que podem exigir grandes quantidades de processamento e/ou E/S, utilizando aplicações com comportamentos diferentes e/ou benchmarks conhecidos.

Referências

AMD (2016). AMD Virtualization. `http://www.amd.com/pt-br/solutions/servers/virtualization`. Acessado em: 10/08/2016.

Babu, S., Hareesh, M., Martin, J., and Sastri, Y. (2014). System performance evaluation of para virtualization, container virtualization, and full virtualization using Xen, OpenVZ, and XenServer. In *Advances in Computing and Communications (ICACC), 2014 Fourth International Conference on*, pages 247–250.

de Souza Munoz, M., De Giovanni, R., de Siqueira, M., Sutton, T., Brewer, P., Pereira, R., Canhos, D., and Canhos, V. (2011). Openmodeller: a generic approach to species' potential distribution modelling. *GeoInformatica*, 15(1):111–135.

Docker (2016). Docker engine user guide. `https://docs.docker.com/engine/userguide/`. Acessado em: 01/07/2016.

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172.

Geller, G. N. and Melton, F. (2008). Looking forward: Applying an ecological model web to assess impacts of climate change. *Biodiversity*, 9(3-4):79–83.

Intel (2016). Intel Virtualization Technology (Intel VT). `http://www.intel.com.br/content/www/br/pt/virtualization/virtualization-technology/intel-virtualization-technology.html`. Acessado em: 10/08/2016.

KVM (2016). Kernel virtual machine. `http://www.linux-kvm.org/page/Main_Page`. Acessado em: 10/08/2016.

Muhammad, H. (2016). htop - an interactive process viewer for unix. `http://hisham.hm/htop/`. Acessado em: 01/07/2016.

OpenMPI (2016). OpenMPI: Open source high performance computing. `https://www.open-mpi.org/`. Acessado em: 01/07/2016.

Pearce, M., Zeadally, S., and Hunt, R. (2013). Virtualization: Issues, security threats, and solutions. *ACM Comput. Surv.*, 45(2):1–39.

Phillips, S. J., Dudík, M., and Schapire, R. E. (2004). A maximum entropy approach to species distribution modeling. In *Proc. of the 21st International Conference on Machine Learning*, pages 83–90.

Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., and Peterson, L. (2007). Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proc. of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 275–287.

Stockwell, D. (1999). The garp modelling system: problems and solutions to automated spatial prediction. *International Journal of Geographical Information Science*, 13(2):143–158.

Su, Y., Liao, X., Jin, H., and Bell, T. (2010). Data hiding in virtual machine disk images. In *IEEE International Conference on Computer and Information Technology (CIT)*, pages 2278–2283.

VmWare (2016). vsphere e vsphere with operations management. `http://www.vmware.com/br/products/vsphere.html`. Acessado em: 10/08/2016.

Xavier, M. G., Neves, M. V., and Rose, C. A. F. D. (2014). A performance comparison of container-based virtualization systems for MapReduce clusters. In *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 299–306.

Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T., and Rose, C. A. F. D. (2013). Performance evaluation of container-based virtualization for high performance computing environments. In *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 233–240.

Xen (2016). Xen server. `http://xenserver.org/`. Acessado em: 10/08/2016.

Zarco-González, M. M., Monroy-Vilchis, O., and Alaníz, J. (2013). Spatial model of livestock predation by jaguar and puma in Mexico: Conservation planning. *Biological Conservation*, 159:80 – 87.