Comparativo entre DOCKER e LXC/LXD para a virtualização

Antonio R. L. Mendes[1], **Angelo A. Duarte**[1]

[1] Laboratório de Computação de Alto Desempenho/Departamento de Tecnologia
Universidade Estadual de Feira de Santana, Bahia, Brasil

raymendesjr2013@gmail.com, angeloduarte@uefs.br

Abstract. *In the world of cloud computing virtualization has gained more and more space by enabling the sharing of resources of a machine between multiple users, reducing costs and simplifying the installation and maintenance of infrastructure tasks.This paper presents a comparative study of computational performance of the most present container virtualisation platforms, DOCKER and LXC/LXD. Through performance testing, we quantitatively analyze efficiency in resource virtualization and container management and identify the appropriate scenarios for each platform.*

Resumo. *No mundo da computação em nuvem a virtualização tem ganhado cada vez mais espaço por possibilitar o compartilhamento dos recursos de uma máquina entre diversos usuários, reduzindo custos e simplificando as tarefas de instalação e manutenção da infraestrutura. Este artigo apresenta um estudo comparativo de desempenho computacional das plataformas de virtualização por container mais presentes no mercado, o DOCKER e o LXC/LXD. Através dos testes de desempenho, analisamos quantitativamente a eficiência na virtualização dos recursos e gestão de containers e identificamos os cenários adequados para cada plataforma.*

1. Introdução

A Computação em Nuvem (Cloud Computing) facilita e viabiliza o compartilhamento de serviços de processamento, armazenamento e rede. Esses recursos, muitas vezes de vários computadores, são compartilhados entre diferentes usuários dando a estes a sensação de estarem usando um único equipamento e que possuem o recurso exclusivamente para si. Essa é a essência por trás do conceito de virtualização.

A virtualização é a criação de uma versão virtual de algum recurso computacional, o que permite exclusividade para que os os usuários percebam que possuem uma recurso dedicado mesmo compartilhado este recurso com outros usuários do sistema. A virtualização ficou mais conhecida com o Hypervisor, um software que permite que executemos simultaneamente vários sistemas operacionais em uma única máquina, criando máquinas virtuais (MV), as quais possuem recursos isolados umas das outras. [Solution 2014].

Com a virtualização veio à necessidade de criar uma política de mapeamento que limitasse o acesso dos usuários apenas aos recursos que lhe fossem necessários. Esse conceito foi alavancado no ano de 2000, com o aparecimento do FreeBSD Jail, que criava “jaulas” para manter a atividade de um usuário confinada em espaços de recursos do

sistema. Quase simultaneamente, em 2001 Jacques Gélinas iniciou um projeto que visava executar vários servidores independentes numa mesma máquina. Em pouco tempo, outros pesquisadores juntaram os conceitos de grupos de controles (cgroups), que é uma funcionalidade do kernel para controlar e limitar o uso de recursos por um processo ou grupo de processos, e o systemd, que configura os espaços usados pelo usuário e pelos processos [Hat 2018].

Os *namespaces* entram em cena para mapear os usuários e grupos por nomes, o que permite definir privilégios para os mesmos. Com isso surge um novo conceito: o *container*, que se assemelha ao *jail*, porém com mais segurança e um isolamento maior entre os processos. Um *container* cria um novo nível de virtualização, e, como o nome já sugere, encapsula processos e os isola do sistema, bem como fornece os arquivos necessários para o seu correto funcionamento. Por conter essas informações, um *container* é portátil e permite a execução de uma mesma configuração em vários hosts.

Em 2008, a Canonical Ltd. e a Ubuntu começaram a elaboração de uma *Application Programming Interface* (API) para o gerenciamento de *containers*, originando o Linux *Container* (LXC). o LXC é uma interface de baixo nível que fornece serviços de contenção de recursos do *kernel* Linux, permitindo a gestão de *container* e ambientes [Containers 2018]. O projeto LXC contempla ferramentas, bibliotecas, associação de linguagens e modelos para facilitar a utilização de *containers* [Hat 2018].

Após o lançamento do LXC, surgiu outro projeto que o complementa: o Linux Container Daemon (LXD). Este se define como um *daemon*, ou seja, um processo que fica em segundo plano para ajudar na escolha dos processos necessários para a execução e gerenciamento dos *containers*.

Em 2008 entra em cena o Docker, que é um facilitador do uso de *containers* projetado pela Docker Inc., configurando-se com base no LXC, só que com ferramentas ainda mais aprimoradas. O Docker tornou-se um dos projetos *open source* mais conhecidos e mais utilizado no mercado para o gerenciamento e execução de containers. [Hat 2018]

Para avaliar as vantagens e desvantagens destas abordagens em relação ao desempenho computacional e à facilidade de instalação/manutenção, neste artigo comparamos a eficiência na virtualização dos recursos e gestão de *containers* usando o LXC/LXD e o Docker.

2. Materiais e Métodos

O trabalho reproduz a metodologia que foi indicada no artigo de Gupta e Gera [Gupta and Gera 2016], que aborda uma comparação do desempenho das três plataformas de virtualização mais comuns no mercado, utilizando como medidas a largura de banda, a quantidade de acessos simultâneos e a velocidade de memória das plataformas. Diante disso selecionamos cinco aplicações para o *benchmark* neste trabalho:

1. Dbench [Andrew Tridgell 2008]: Gera cargas de trabalho de entrada e saída em um sistema de arquivos local ou servidor. Podendo ser usado para mensurar a quantidade de acessos simultâneos que o sistema suporta até ficar sobrecarregado;
2. GZIP Compression [Free Software Foundation 2018]: Mensura o desempenho de entrada e saída do sistema para compactar um arquivo de 2GB;

3. John the Ripper (blowfish) [Designer 2014]: Utilizada junto com o BlowFish para encriptografar e descriptografar dados;
4. RAM Speed [R. Hollander 2018]: Mensura a velocidade de comunicação de leitura e escrita entre a memória e suas caches;
5. Stream [John D. McCalpin 2007]: Mede a largura de banda estressando ao máximo os núcleos do processador.

O computador utilizado foi um Desktop Dell com processador Intel i5, 8GB de RAM e HD 500GB, com sistema operacional Linux Ubuntu v 16.04 (Xerus) kernel 4.13.0-36-generic, Docker v18.09 e LXD v3.8.

Cada teste foi selecionado para avaliar uma área específica do desempenho das máquinas. O *Dbench* e o *GZIP* foram usados para análise do acesso ao sistema de arquivo, entrada e saída de cada *container*. O *Stream* e o *Blowfish* averíguam a velocidade dos núcleos do CPU, enquanto o *RAM Speed* testa a memória do sistema. Todos os testes foram encontrados no Phoronix Test Suite (PTS) [Media 2014], software que automatiza a execução dos *benchmarks*.

Nos experimentos, iniciamos com o computador apenas com o sistema operacional instalado. Em seguida foi instalada uma das plataformas (Docker ou LXD) e criado um *container* com o PTS. Na sequência foram executadas as baterias de testes.

Para a plataforma Docker, a instalação e a execução do *container* foi bem mais simples, haja visto a existência de uma imagem do PTS disponível no Docker Hub (https://hub.docker.com/). Já na execução com o LXD foi necessária a criação e configuração do *container* com todas as dependências indispensáveis para a realização dos testes.

3. Resultados

Para cada teste foram feitas 5 (cinco) execuções, com os valores obtidos encontramos o desvio padrão, o coeficiente de variação e a média eliminando o maior e o menor tempo de execução.

3.1. Teste de Desempenho de CPU

Para testar a velocidade de processamento usamos o *benchmark* John the Riper com o Blowfish, um algoritmo *open source* desenvolvido por Bruce Schneier, que utiliza sistemas de blocos, chaves de 32 a 448bits e 16 iterações para criptografar e descriptografar strings[Schneier 1993]. O aplicativo "John the Ripper"é usado para calcular o tempo que a CPU leva para criptografar e descriptografar um conjunto de Strings.

Analisando a Figura 1 e sabendo que o coeficiente de variação para o docker foi de 0.00269 e para o LXD foi de 0.00969, podemos ver que o LXD é mais rápido, porém a diferença entre as plataformas é pequena, aproximadamente 0,65% (inferior a 40 C/s (*cripts per second*) em média). Com esse teste percebemos que o LXD é mais eficaz em explorar a CPU.

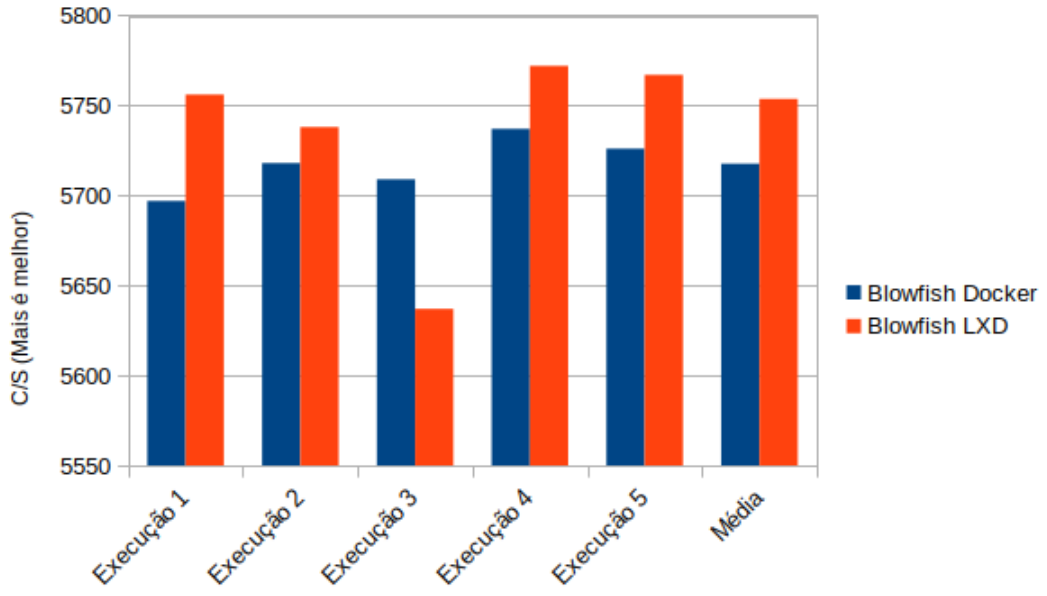

Figura 1. Execução do John the Riper(blowfish) nas duas plataformas colocadas lado a lado; OS DADOS SÃO AVALIADOS PELA QUANTIDADE DE CRIPTOGRAFIAS POR SEGUNDO (C/S);

3.2. Teste de Desempenho de Memória

O Stream é responsável por estimar a largura de banda (*bandwith*) em MB/s utilizada entre a memória do sistema e a CPU para realização de cálculos matemáticos. A Tabela 1 detalha a quantidade de bytes e de FLOPS (*Float Operations per Second*) para cada interação do Stream, em que **a(i)**, **b(i)** e **c(i)** são posições da memória para a iteração "i"e **q** é um escalar. As ações são de copiar o valor de *b* para *a* (Copy), somar *a* com *b* (Add), multiplicar *a* por *q* (Scale) e somar *a* com a multiplicação de *b* e *q* (Triad).

Tabela 1. Tabela de operações Stream

Nome	Função	Bytes/iteração	FLOPS/iteração
Copy	a(i) = b(i)	16	0
Add	c(i) = a(i) + b(i)	16	1
Scale	c(i) = q*a(i)	24	1
Triad	c(i) = a(i)+q*b(i)	24	2

Na Figura 2 podemos ver resultado da média de tempo das execuções para cada plataforma. Notamos que a diferença entre as duas plataformas é de, aproximadamente, 1,5%, sendo o LXD o mais eficaz. Na Tabela 2 podemos ver o restante dos dados e constatar a superioridade do LXD em acesso à memória.

Tabela 2. Tabela de resultados Stream

Teste	Desvio Padrão Docker	Desvio Padrão LXD	Coef. de Variação Docker	Coef. de Variação LXD
Add	43,16480	10,67708	0,00202	0,00049
Copy	18,29481	2,68328	0,00071	0,00010
Scale	456,73844	4,14729	0,02396	0,00022
Triad	44,58363	7,36206	0,00209	0,00034

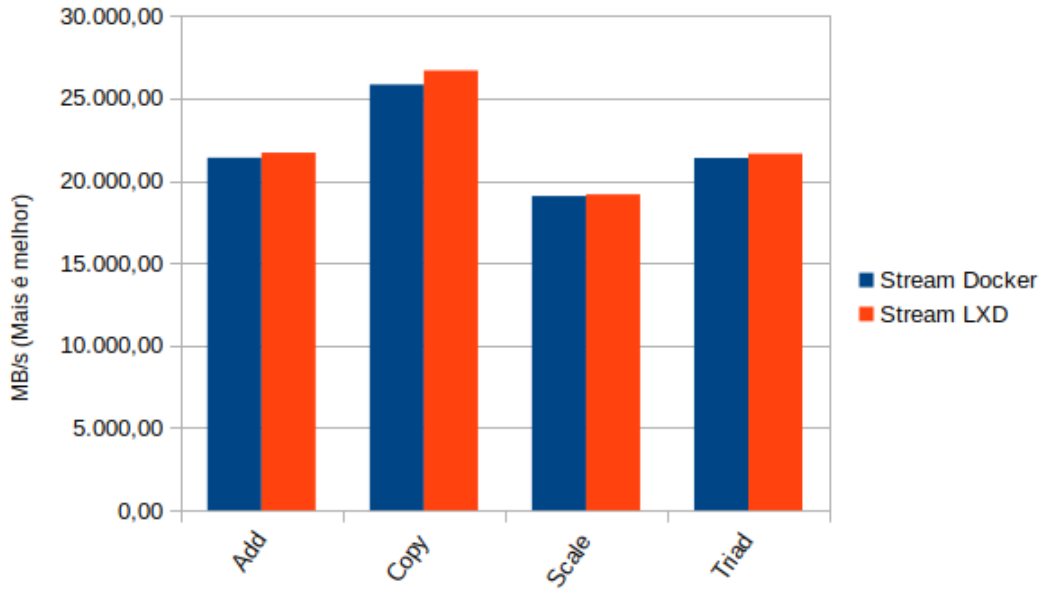

Figura 2. Execução do Stream nas duas plataformas colocadas lado a lado;

Usando o benchmark RAMSpeed, que executa uma série de testes semelhantes ao Stream com foco em quantos MB/s são necessários para fazer a comunicação entre os níveis de memória do sistema (largura de banda - *bandwith*). As funções são idênticas às mostradas na Tabela 1, adicionando apenas a função de média aritmética. Todas as funções são executadas para números inteiros e de ponto flutuante.

Analisando a Figura 3 e a Tabela 3 percebemos que a execução com o LXD obteve uma eficácia maior, com quase 15% a mais de largura de banda. Dessa forma, pode-se concluir que a largura de banda das transações com o LXD é maior, e isso implica em maior fluxo de dados entre as memórias, impactando positivamente no desempenho do *container*.

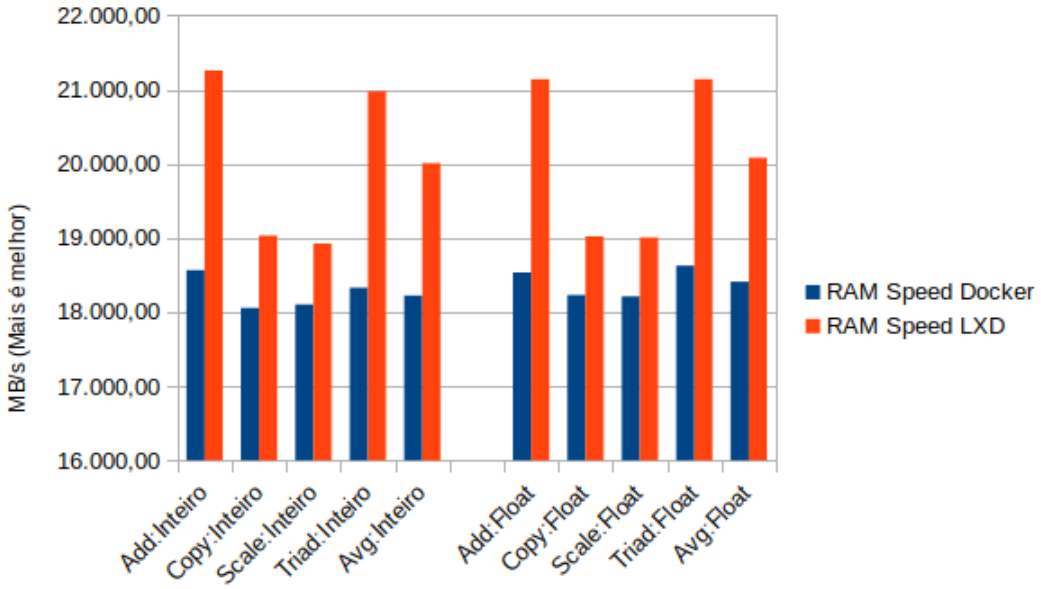

Figura 3. Execução do RAMSpeed nas duas plataformas colocadas lado a lado;

Tabela 3. Tabela de resultados RAM Speed

Teste	Desvio Padrão Docker	Desvio Padrão LXD	Coef. de Variação Docker	Coef. de Variação LXD
Add:int	253,599	273,147	0,0137	0,0128
Copy:int	353,060	57,801	0,0196	0,0030
Scale:int	695,234	40,257	0,0384	0,0021
Triad:int	801,267	184,154	0,0437	0,0088
AVG:int	247,872	96,344	0,0136	0,0048
Add:float	395,879	179,204	0,0214	0,0085
Copy:float	269,599	34,033	0,0148	0,0018
Scale:float	91,604	30,899	0,0050	0,0016
Triad:float	68,889	175,092	0,0037	0,0083
AVG:float	148,390	95,002	0,0081	0,0047

3.3. Teste Desempenho de Entrada/Saída

Na avaliação de entrada e saída e de quantidade de acessos simultâneos usamos o Dbench e GZIP Compress para a largura de banda em uma situação real.

O Dbench coleta informações sobre o desempenho do *container* quando há acessos simultâneos ao sistema de arquivos, por usuários ou por algoritmos. Isso é importante para saber quantos acessos o sistema suporta até começar a perder desempenho. O teste simula o acesso simultâneo de 6, 12, 128 e 256 clientes.

Na Figura 4 podemos notar uma grande diferença entre o LXD usando o sistema de arquico ZFS e o Docker usando o ”Device Mapper”. A diferença de desempenho é de quase 85%. Então, podemos dizer que o LXD seria a melhor opção ao usar o *container* para prover um serviço de acesso simultâneo.

Tabela 4. Tabela de resultados Dbench

Clientes	Desvio Padrão Docker	Desvio Padrão LXD	Coef. de Variação Docker	Coef. de Variação LXD
6	2,819	1,095	0,1123	0,0082
12	4,994	2,702	0,1231	0,0117
128	5,927	39,480	0,0556	0,1229
256	5,454	30,542	0,0564	0,0972

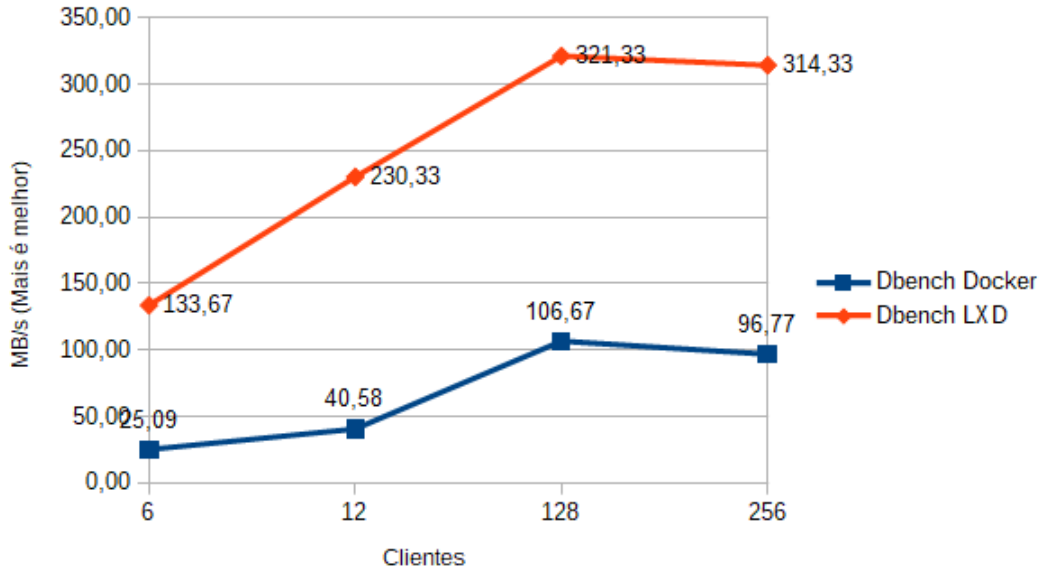

Figura 4. Execução do Dbench nas duas plataformas colocadas lado a lado;

Quanto ao GZIP, ele é utilizado para fazer a medição de largura de banda quando se trata de uma compressão de dados. Esse teste comprime um arquivo com mais de 2GB de tamanho e calcula o tempo necessário.

A Figura 5 mostra que a compressão de dados feita pelo Docker é mais eficiente, levando até 28 segundo a menos, os valores de variância - 0377 para o docker e 0,0035 para o LXD - comprovam essa superioridade. Analisando os dois testes chegamos à conclusão que o sistema de arquivos do ZFS é mais robusto e projetado para atender demandas enquanto o ”Device Mapper”é mais eficaz para acesso unitário.

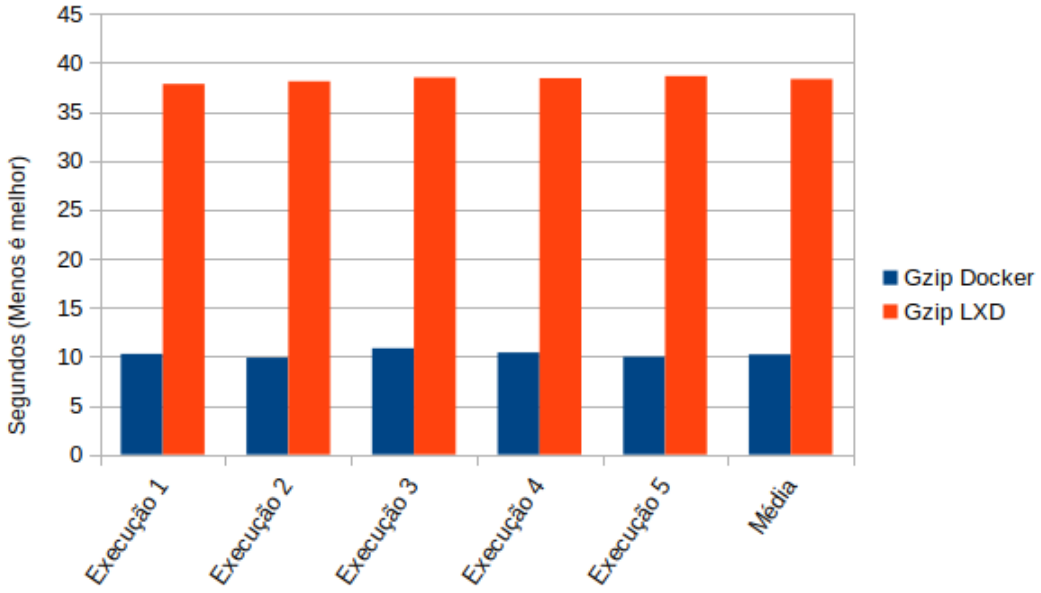

Figura 5. Execução do GZIP Compress nas duas plataformas colocadas lado a lado;

4. Conclusão

Este trabalho se propôs a fazer um comparativo de desempenho computacional entre as plataformas LXC/LXD e Docker.

Com base nos resultados obtidos nos testes, podemos dizer que o LXD apresentou maior desempenho para aplicações em que a largura de banda de memória é o requisito mais importante para o desempenho (aplicações *Memory Bound*). Não percebemos qualquer diferença expressiva entre as duas plataformas para aplicativos em que a CPU é o item chave para o desempenho (aplicações *CPU Bound*).

O Docker se destaca na facilidade de uso e compatibilidade com Sistemas Operacionais populares no mercado para servidores, pois para configurá-lo é necessário, na maioria das vezes, executar apenas um *script* de instalação. O LXD, por sua vez, só é compatível com distribuições baseadas em Debian, e todo o processo de instalação tem que ser minunciosamente feito pelo usuário.

Além da facilidade de uso, o Docker, por ser uma plataforma *open source*, conta com vários colaboradores e um acervo repleto de imagens disponíveis para download. Isso compensa o fato de que o Docker apresenta menor desempenho para aplicações *Memory Bound*.

5. Agradecimentos

O primeiro autor agradece à Fundação de Amparo à Pesquisa do Estado da Bahia (FAPESB) pela bolsa de Iniciação Científica recebida durante a realização deste trabalho.

Referências

Andrew Tridgell, R. (2008). Dbench. Disponível em: `https://dbench.samba.org`. Acesso em: 19 fev. 2019.

Containers, L. (2018). What's lxc? Disponível em: https://linuxcontainers.org/lxc/introduction/. Acesso em: 19 fev. 2019.

Designer, S. (2014). John the ripper password cracker. Disponível em: https://www.openwall.com/john/. Acesso em: 19 fev. 2019.

Free Software Foundation, I. (2018). Gzip compression. Disponível em: https://openbenchmarking.org/test/pts/compress-gzip/. Acesso em: 19 fev. 2019.

Gupta, S. and Gera, D. (2016). A comparison of lxd, docker and virtual machine. *International Journal of Scientific & Engineering Research*, 7(9).

Hat, R. (2018). O que é um container linux? Disponível em: https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container. Acesso em: 19 fev. 2019.

John D. McCalpin (1991-2007). Stream benchmark. Disponível em: http://www.cs.virginia.edu/stream/ref.html. Acesso em: 19 fev. 2019.

Media, P. (2014). Phoronix test suite v8.6.0. Disponível em: https://www.phoronix-test-suite.com/documentation/phoronix-test-suite.pdf. Acesso em: 19 fev. 2019.

R. Hollander, P. B. (2018). Ramspeed smp. Disponível em: https://openbenchmarking.org/test/pts/ramspeed-1.4.1. Acesso em: 19 fev. 2019.

Schneier, B. (1993). Description of a new variable-length key, 64-bit block cipher (blowfish). In *International Workshop on Fast Software Encryption*, pages 191–204. Springer.

Solution, B. (2014). O que é virtualização? Disponível em: https://www.bluesolutions.com.br/2014/07/o-que-e-virtualizacao/. Acesso em: 19 fev. 2019.