UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

Leonardo da Silva Ferreira

Eficiência Energética na Consolidação de Servidores: Uma Comparação Entre a Virtualização e a Conteinerização

Natal
2018

LEONARDO DA SILVA FERREIRA

Eficiência Energética na Consolidação de Servidores: Uma Comparação Entre a Virtualização e a Conteinerização

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Computação e Automação da Universidade Federal do Rio Grande do Norte como parte dos requisitos para a obtenção do título de Engenheiro de Computação.

Natal, dezembro de 2018

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

LEONARDO DA SILVA FERREIRA

Esta Monografia foi julgada adequada para a obtenção do título de Engenheiro de Computação, sendo aprovada em sua forma final pela banca examinadora:

Orientador: Prof. Me. Wellington Silva de Souza
Universidade Federal do Rio Grande do Norte - UFRN

Prof. Dr. Carlos Manuel Dias Viegas
Universidade Federal do Rio Grande do Norte - UFRN

Prof. Dr. Danilo Curvelo de Souza
Universidade Federal do Rio Grande do Norte - UFRN

Prof. Dr. Samuel Xavier de Souza
Universidade Federal do Rio Grande do Norte - UFRN

Natal, 7 de dezembro de 2018

À minha mãe, por todo amor, motivação e exemplo que me deu desde sempre.

Agradecimentos

Agradeço, em primeiro lugar, a Deus, sem o qual nada disso seria possível. Pela Sua Graça tudo foi realizado. Obrigado Senhor por sempre cuidar de mim.

À minha família, em especial à minha mãe, Fátima, que sempre foi para mim um exemplo de luta, garra e determinação. Obrigado, mãe, por me mostrar como é importante o estudo e o trabalho duro para alcançarmos nossos objetivos na vida.

Agradeço imensamente ao meu amor, Klara, que chegou no meio dessa minha caminhada, mas que já me apoiou como nunca, sofreu comigo nas dificuldades e compartilhou as alegrias dessa jornada. Obrigado meu bem!

Meus agradecimentos ao meu orientador, professor Wellington Souza, pela sua paciência e incentivo na elaboração deste trabalho.

Ao professor Samuel Souza e demais contribuintes do Laboratório de Arquiteturas Paralelas para Processamento de Sinais - LAPPS, pela disponibilidade e acolhimento no laboratório, local onde desenvolvi grande parte das pesquisas referentes ao trabalho.

Resumo

Em diversas áreas científicas e profissionais, é crescente a demanda por recursos computacionais de alto desempenho, pois são de extrema importância para obtenção de melhores resultados na solução de problemas cada vez mais complexos. A virtualização, na última década, contribuiu de forma decisiva na otimização do uso de recursos computacionais seja em centros de dados (*datacenters*) privados, quanto em ambientes de provedores de serviço de nuvem. A partir do compartilhamento de recursos de *hardware*, houve significativo incremento da eficiência energética. Nestes ambientes, tradicionalmente dominados pela tecnologia de máquinas virtuais a partir de hipervisores, o uso de contêineres para provisionamento de aplicações tem se tornado popular nos últimos anos. Considerados como uma alternativa leve ao ambiente de hipervisores, os contêineres trazem diversos benefícios, tais como menor sobrecarga do *hardware*, redução de falhas em função da configuração, simplificação de atualizações e maior velocidade na inicialização. A escalada na adoção dessa tecnologia se deve principalmente pela evolução das ferramentas de orquestração, a exemplo do Docker, que simplificam a criação, execução e e gerenciamento de contêineres. Considerando estes aspectos, o presente trabalho se propõe a estabelecer uma análise comparativa de consumo e eficiência energética de aplicações executadas em ambientes virtualizados e contêineres.

Palavras-chave: análise de desempenho, contêineres de *software*, eficiência energética, hipervisores, máquinas virtuais, virtualização.

Abstract

In many scientific and professionals areas the demand for high performance computing resources is ever-increasing because of their extreme importance to obtain the best result in solving increasingly complex problems. In the last decade virtualization has contributed in a decisive way in optimizing computational resources usage, either in private datacenters and cloud-computing service providers. Traditionally dominated by virtual machine technology based on hypervisors, environments that make use of containers for application provisioning have become popular. Once hardware resources being shared there was a significant increase in energy efficiency. On these environments, traditionally dominated by technology virtual machines from hypervisors, the use of containers for application provisioning have become popular in past few years. Considered as a lightweight alternative to hypervisors, containers bring several benefits, such as lower hardware overhead, reduction of configuration-depending failures, simplification of upgrades and greater speed on startup. The steady increase in adoption of this technology is mainly due the evolution of the orchestration tools, Docker as an example, which simplify the process of creation, implementation and management of containers. Considering these aspects, the present study proposes to establish a comparative analysis of consumption and efficiency of applications running in virtualized and containerized environments.

Keywords: benchmarking, software containers, energy efficiency, hypervisors, virtual machines, virtualization.

Lista de ilustrações

Figura 1 – Exemplo de Virtualização *Bare-metal*. 17
Figura 2 – Exemplo de Virtualização *Hosted*. 17
Figura 3 – Estrutura para Contêineres 20
Figura 4 – Animações visualmente realistas de um rosto humano criado pelo *facesim*. 25
Figura 5 – Algorítmo do *ferret*. 26
Figura 6 – Acesso ao arquivo *power_now*. 27
Figura 7 – Ferramenta em execução. 28
Figura 8 – Função *get_power*. 29
Figura 9 – Função principal do código. 30
Figura 10 – Taxa de Watts a cada segundo na utilização normal do processador . . 32
Figura 11 – Taxa média de Watts a cada segundo na utilização da aplicação no ambiente nativo. 34
Figura 12 – Taxa média de Watts a cada segundo na utilização da aplicação na máquina virtual. 35
Figura 13 – Taxa média de Watts a cada segundo na utilização da aplicação no contêiner. 36
Figura 14 – Médias das taxas de Watts a cada segundo. 37
Figura 15 – Taxa média de Watts a cada segundo na utilização da aplicação no ambiente nativo. 38
Figura 16 – Taxa média de Watts a cada segundo na utilização da aplicação na máquina virtual. 39
Figura 17 – Taxa média de Watts a cada segundo na utilização da aplicação no contêiner. 40
Figura 18 – Médias das taxas de Watts a cada segundo. 40
Figura 19 – Imagem criada a partir do Dockerfile. 52
Figura 20 – Contêiner criado a partir da imagem. 53
Figura 21 – Configuração da Máquina Virtual. 54

Lista de tabelas

Tabela 1 – Domínio de aplicação e paralelização dos *benchmarks* PARSEC. 23
Tabela 2 – Conjunto de trabalho e uso de dados dos *benchmarks* PARSEC. 23
Tabela 3 – Fórmulas para cálculo da média, desvio padrão e CV. 31
Tabela 4 – Resultados obtidos da utilização normal do processador. 32
Tabela 5 – Resultados obtidos da execução do *facesim* no ambiente nativo. 33
Tabela 6 – Resultados obtidos da execução do *facesim* na máquina virtual. 34
Tabela 7 – Resultados obtidos da execução do *facesim* no contêiner. 35
Tabela 8 – Resultados obtidos da execução do *ferret* no ambiente nativo. 37
Tabela 9 – Resultados obtidos da utilização do *ferret* na máquina virtual. 38
Tabela 10 – Resultados obtidos da utilização do *ferret* no contêiner. 39
Tabela 11 – Uso da memória. 41
Tabela 12 – Uso do disco. 41
Tabela 13 – Tempos de execução do *facesim*. 42
Tabela 14 – Tempos de execução do *ferret*. 42
Tabela 15 – Eficiência energética entre máquina virtual e contêiner. 43
Tabela 16 – Comparação de consumo energético entre contêiner e ambiente nativo. 44
Tabela 17 – Comparação de consumo energético entre máquina virtual e ambiente nativo. 44

Lista de abreviaturas e siglas

APT	*Advanced Packaging Tool*
CMP	*Muiltiprocessor*
CPU	*Central Process Unit*
CV	Coeficiente de Variação
DPR	Desvio Padrão Relativo
EMD	*Earth Mover's Distance*
E/S	Entrada e Saída
GPG	*GNU Privacy Guard*
HPC	*High Performance Computing*
JVM	*Java Virtual Machine*
LSH	*Locality Sensitive Hashing*
LXC	*Linux Containers*
PARSEC	*Princeton Application Repository for Shared-Memory Computers*
PID	*Process Identification*
RMS	*Recognition, Mining and Synthesis*
SO	Sistema Operacional
TI	Tecnologia da Informação
VM	*Virtual Machine*
VMM	*Virtual Machine Monitor*

Sumário

1 INTRODUÇÃO 13
1.1 Justificativa 14
1.2 Objetivos 14
1.2.1 Objetivo Geral 14
1.2.2 Objetivos Específicos 14
1.3 Organização 15

2 *FUNDAMENTAÇÃO TEÓRICA* 16
2.1 Máquinas Virtuais 16
2.1.1 Hipervisor 16
2.1.1.1 Tipo 1 16
2.1.1.2 Tipo 2 17
2.1.2 Virtualização completa 18
2.1.3 Paravirtualização 18
2.2 Contêineres 19
2.2.1 Docker 20

3 METODOLOGIA 22
3.1 PARSEC 22
3.1.1 Facesim 24
3.1.2 Ferret 25
3.2 Obtenção da potência dissipada 27
3.2.1 Monitoramento de potência a partir do SO 27
3.2.2 Stress-Terminal UI 27
3.2.3 Rotina de monitoramento 28
3.3 Cálculos estatísticos 30

4 RESULTADOS 32
4.1 Facesim 33
4.1.1 Execução em ambiente nativo 33
4.1.2 Execução em máquina virtual 34
4.1.3 Execução em contêiner 35
4.1.4 Comparação dos gráficos com as médias das taxas 36
4.2 Ferret 37
4.2.1 Execução em ambiente nativo 37
4.2.2 Execução em máquina virtual 38

4.2.3 Execução em contêiner 39
4.2.4 Comparação dos gráficos com as médias das taxas 40
4.3 Uso da memória e do disco 41
4.4 Tempos de execução 42
4.5 Eficiência energética 43

5 CONCLUSÃO E TRABALHOS FUTUROS 45

REFERÊNCIAS 47

APÊNDICE A – DOCKER 51
A.1 Instalação 51
A.2 Dockerfile 51
A.3 Criação do contêiner 52

APÊNDICE B – MÁQUINA VIRTUAL 54
B.1 Instalação 54

APÊNDICE C – PARSEC 55
C.1 Execução dos programas 55

1 Introdução

As tecnologias de virtualização têm exercido uma função de destaque no fornecimento de ambientes destinados a execução de aplicações. Com o objetivo de diminuir os custos e propiciar uma melhora na eficiência dos centros de dados corporativos, os setores responsáveis pela infraestrutura de TI investiram significativamente, nos últimos dez anos, na consolidação de serviços computacionais. Com isso, vários servidores virtuais são executados em um mesmo *host*, o que resulta em uma melhor utilização da capacidade do *hardware*, reduzindo o consumo de energia nos *datacenters* e o espaço [Silva 2017].

Várias aplicações empresariais que geralmente funcionavam em servidores dedicados são consolidadas em um agregado compartilhado de servidores. Isso impõe novos desafios, incluindo a seleção da tecnologia de virtualização e a estrutura de consolidação adequadas para um conjunto específico de aplicativos [Padala et al. 2007]. A solução de virtualização mais utilizada é a tecnologia baseada em hipervisor (*hypervisor*), que, segundo Cabral [Cabral 2013], tem como principais representantes o VMware [Vmware 2018], Hyper-V [Microsoft 2018] e Xen [Xenproject 2013].

Segundo [Silva 2017], uma crescente adesão foi observada nos últimos anos, especialmente por provedores de computação em nuvem, da virtualização de aplicações a partir da utilização de contêineres. Esta retrata uma proposta contrária de abstração em termos de virtualização e isolamento quando comparado com hipervisores. Em particular, os contêineres podem ser classificados como uma alternativa leve à virtualização baseada em hipervisores. Estes abstraem a arquitetura alvo, o que resulta em sobrecarga decorrente da virtualização de *hardware* e *drivers* de dispositivos virtuais.

Na virtualização tradicional, um sistema operacional completo (Linux, por exemplo) é executado no topo de um *hardware* virtualizado. Todos os componentes necessários ao seu funcionamento (incluindo *kernel*, *drivers*, sistema de arquivos, escalonador de processos, gerenciamento de memória e E/S) são executados numa instância de máquina virtual. Esta estrutura é replicada para cada aplicação que se deseje isolar no ambiente. Em contraste, os contêineres implementam isolamento de processos no nível do sistema operacional, evitando assim tais sobrecargas. São executados sobre o mesmo *kernel* em operação na máquina *host* subjacente, compartilhado-o com esta. Assim como em máquinas virtuais, um ou mais processos podem ser executados dentro de cada contêiner [Morabito, Kjällman e Komu 2015].

1.1 Justificativa

Considerando as diferenças apresentadas, surge como hipótese o fato de que uma mesma computação apresentará desempenho e eficiência energética diferentes quando executada em máquinas virtuais e contêineres. Sendo assim, faz-se necessário proceder a uma análise teórica e experimental para validar esta hipótese, bem como, em caso afirmativo, mensurar as diferenças apresentadas.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral do trabalho é efetuar uma análise comparativa de consumo energético de aplicações executadas em ambientes virtualizados e conteinerizados. Para o presente estudo, foram utilizadas as ferramentas VirtualBox, como hipervisor, e Docker, como osquestrador de contêineres.

É esperado que outros hipervisores, como VMware Player e Server ofereçam desempenho semelhante ao VirtualBox, dado que se utilizam dos mesmos recursos de aceleração de *hardware*. Da mesma forma, outras ferramentas para contêineres devem apresentar desempenho similar ao Docker quando usam os mesmos mecanismos e recursos computacionais. No trabalho não foram avaliados o uso de contêineres dentro de máquinas virtuais e vice-versa, uma vez que consideramos redundante essa dupla virtualização (pelo menos do ponto de vista do desempenho). O fato de o Linux poder hospedar tanto máquinas virtuais quanto contêineres cria uma oportunidade de comparação adequada entre as duas tecnologias.

1.2.2 Objetivos Específicos

Considerando o desenvolvimento do trabalho e o objetivo geral apresentado, destacam-se os seguintes objetivos específicos:

- Realizar um estudo de tecnologias de virtualização;
- Realizar um estudo de tecnologias de contêineres de *software*;
- Especificar os pacotes da aplicação PARSEC *Benchmark* [PARSEC 2009];
- Realizar testes executando as aplicações em máquina virtual e contêiner;
- Analisar e apresentar os resultados obtidos.

1.3 Organização

O presente trabalho está organizado da seguinte forma: no capítulo 2 os conceitos relacionados a máquinas virtuais e contêineres são abordados. O capítulo 3 aborda a metodologia adotada para obtenção dos dados necessários ao estudo, incluindo as ferramentas utilizadas e códigos elaborados, além da suíte de aplicações PARSEC *Benchmark*, utilizada no estudo para a realização de testes de eficiência energética do sistema.. Na sequência, o capítulo 4 apresenta os resultados obtidos em conjunto com uma análise dos mesmos. Finalizando, o capítulo 5 apresenta a conclusão e avalia possíveis trabalhos futuros.

2 Fundamentação Teórica

2.1 Máquinas Virtuais

"Máquina virtual, do original em inglês *virtual machine* (VM), é uma implementação de ambiente computacional onde pode-se instalar e executar um sistema operacional", segundo [Santos 2014]. Uma VM também pode emular um *hardware* virtual, como o *Java Virtual Machine* (JVM), um programa que carrega e executa os aplicativos Java, convertendo os *bytecodes* em código executável de máquina.

"A máquina virtual emula um computador físico comum, onde os requerimentos de CPU, memória, disco, rede e outros *hardwares* são gerenciados pela camada de virtualização, que traduz esses requerimentos para o *hardware* original", afirma Santos [Santos 2014].

2.1.1 Hipervisor

O hipervisor é uma camada de *software* situada entre o nível de *hardware* e o sistema operacional. Tem a responsabilidade de controlar o acesso do sistema operacional visitante (máquina virtual) aos dispositivos de *hardware*.

"Todo o gerenciamento e destinação de recursos de *hardware* de uma VM é feito pelo *hypervisor* ou Monitor de Máquina Virtual (VMM – *Virtual Machine Monitor*)", afirma [Devmedia 2014].

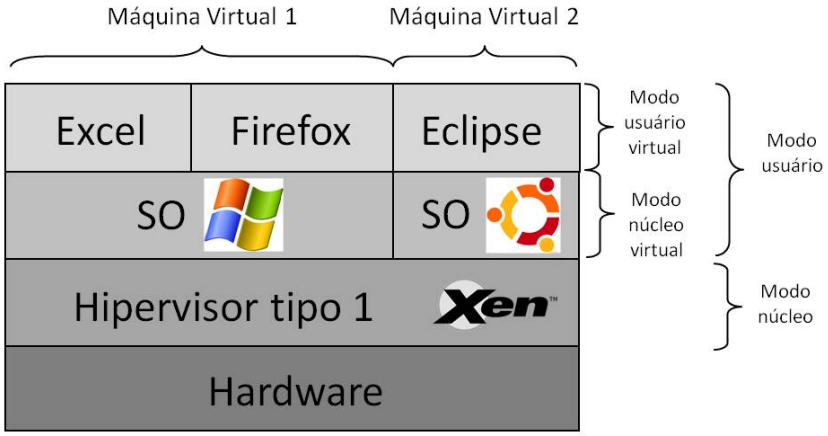
> O VMM é executado no estado de supervisor, no entanto as máquinas virtuais são executadas em modo de usuário. Quando estas tentam efetuar uma instrução privilegiada, é gerada uma interrupção e o VMM se encarrega de emular a execução desta instrução.
> [Mattos 2008]

A virtualização do tipo completa fornece dois tipos de hypervisor. O tipo 1, chamado de bare-metal e o tipo 2, chamado de *hosted*, o quais são discutidos a seguir.

2.1.1.1 Tipo 1

Também intitulado nativo, *unhosted* (não hospedado) ou *bare-metal* (sobre o "metal nu"), é o *software* que se executa diretamente sobre o *hardware* para propiciar funcionalidades de virtualização. Geralmente, já é um sistema operacional pronto para ser instalado no equipamento alvo, e a partir do qual são instanciadas as máquinas virtuais [Cleuber 2014]. A Figura 1 ilustra o hipervisor tipo 1.

Figura 1 – Exemplo de Virtualização *Bare-metal*.



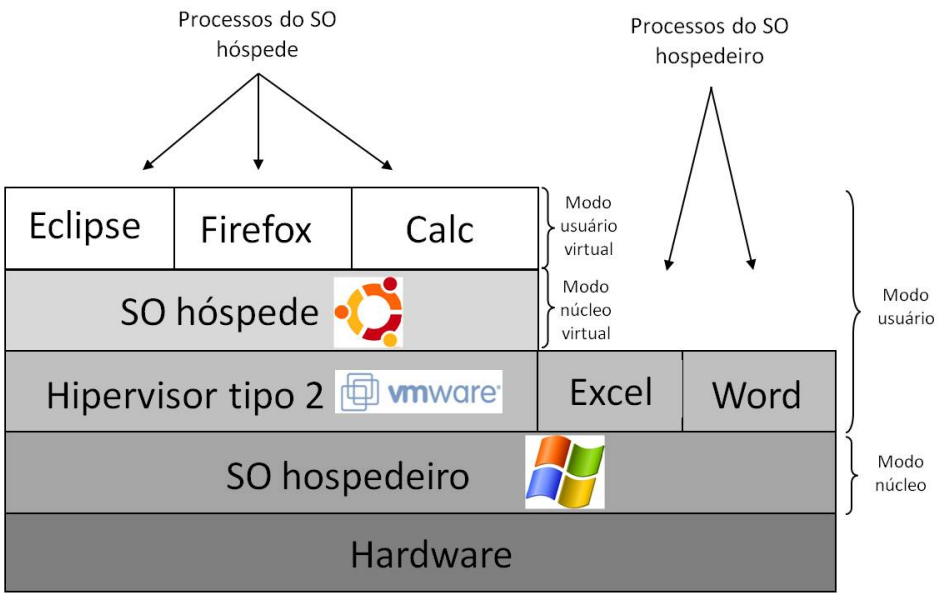
Fonte: [Cleuber 2014].

Alguns dos hipervisores tipo 1 mais conhecidos são os seguintes [Cleuber 2014]: VMware ESXi e Citrix XenServer (gratuitos); Microsoft Hyper-V Server, VMWare V-Sphere e ESX (softwares comerciais); e Xen (livre).

2.1.1.2 Tipo 2

Também denominado *hosted* (hospedado), é o *software* de virtualização que se executa sobre um sistema operacional pré-existente. Ao contrário do tipo 1, este precisa ser instalado no sistema operacional (Windows, Linux, macOS, entre outros). Após sua instalação, as máquinas virtuais podem ser elaboradas, tal como no hipervisor de tipo 1. A Figura 2 ilustra o hipervisor tipo 2.

Figura 2 – Exemplo de Virtualização *Hosted*.



Fonte: [Cleuber 2014].

Alguns dos hipervisores tipo 2 mais utilizados são os seguintes [Cleuber 2014]: Oracle VirtualBox e VMWare Player (gratuitos); Microsoft Virtual PC e VMware Workstation (softwares comerciais); e QEMU (livre).

2.1.2 Virtualização completa

A virtualização completa gera um sistema físico virtual (uma réplica do *hardware* subjacente), sobre o qual o sistema operacional convidado é executado. Não é preciso efetuar qualquer alteração no sistema operacional convidado ou em suas aplicações.

> O fato de o SO visitante não ter consciência de que está sendo executado sobre o VMM causa uma queda do desempenho, pois as instruções executadas pelo sistema operacional visitante devem ser testadas pelo VMM para que depois sejam executadas diretamente no *hardware*, ou executadas pelo VMM e simulada a execução para o sistema visitante.
>
> [Mattos 2008]

Tecnologias de aceleração de virtualização em *hardware*, como Intel VT [Intel 2018] e AMD-V [AMD 2018], reduzem a perda de desempenho.

2.1.3 Paravirtualização

Na técnica da paravirtualização, a máquina virtual não é equivalente ao equipamento físico de origem, justamente para que o sistema hospedado possa enviar instruções mais simples ao *hardware*, deixando apenas as instruções privilegiadas, de nível mais alto, para serem emuladas pelo *hypervisor*.

> Nesse modelo de virtualização, o sistema operacional é modificado para chamar o VMM sempre que executar uma instrução que possa alterar o estado do sistema, uma instrução sensível. Isso acaba com a necessidade de o VMM testar instrução por instrução, o que representa um ganho significativo de desempenho. Outro ponto positivo da paravirtualização é que os dispositivos de *hardware* são acessados por *drivers* da própria máquina virtual, não necessitando mais do uso de *drivers* genéricos que inibiam o uso da capacidade total do dispositivo.
>
> [Mattos 2008]

Um aspecto negativo da paravirtualização é a necessidade de adaptação do sistema operacional hospedado/convidado.

2.2 Contêineres

A tecnologia de contêineres visa providenciar uma unidade de computação autosuficiente para execução de uma aplicação em um ambiente computacional [Pahl 2015]. Um contêiner isola o aplicativo e suas dependências, permitindo que ele seja executado de forma separada das outras aplicações em um mesmo computador [Mouat 2016]. Para Merkel [Merkel 2014] os contêineres são executados a partir do sistema operacional (SO) hospedeiro em espaços próprios e isolados providos pelo núcleo (*kernel*).

Pahl [Pahl 2015] esclarece que os contêineres são organizados em camadas feitas de imagens individuais produzidas sobre uma imagem de base que pode ser estendida. As imagens completas formam contêineres de aplicativos portáteis.

Yu [Yu 2007] afirma que a virtualização a nível de SO é estabelecida como a existência de múltiplas instâncias isoladas de espaços de usuário (ao invés de apenas uma), que são administradas pelo *kernel* deste sistema. Estas instâncias, chamadas de contêineres de *software* (ou simplesmente contêineres), representam todo um ambiente de execução: uma aplicação, juntamente com todas suas dependências, bibliotecas e arquivos de configuração, agrupados em um único "pacote". Ao criar um contêiner de uma determinada aplicação e suas dependências, a diversidade de sistemas operacionais e infraestruturas subjacentes são abstraídas.

Máquinas virtuais e contêineres possuem funcionalidades semelhantes quanto ao isolamento e alocação de recursos computacionais. No entanto, a abordagem arquitetural específica, ilustrada na Figura 3, permite que os contêineres possuam uma maior portabilidade e eficiência.

Na referida ilustração, duas aplicações em contêineres distintos são executadas em um único sistema operacional e compartilham o *kernel* do sistema entre si. Isso faz com que os contêineres consumam menos recursos e sejam mais leves do que máquinas virtuais. A parte compartilhada é somente leitura, enquanto que cada contêiner possui o seu ambiente para escrita.

> No entanto, contêineres e máquinas virtuais podem ser vistos como tecnologias que se complementam ao invés de competir entre si. Isso se deve ao fato da possibilidade de executar contêineres em máquinas virtuais, facilitando, através da virtualização de *hardware*, a gerência da infraestrutura (tais como armazenamento e rede), além de aumentar o isolamento e a segurança.
>
> [RUBENS 2015]

Figura 3 – Estrutura para Contêineres

Fonte: [Softdesign 2017].

2.2.1 Docker

O Docker [Docker 2018] é uma plataforma livre para elaboração, entrega e execução de aplicativos por meio de contêineres, produzida e mantida pela empresa Docker Inc. O seu ecossistema abrange: um método de criação de imagens e de execução de contêineres chamado de Docker Engine; os repositórios para armazenamento e a distribuição de imagens, conhecidos como registros (Registry) - o que inclui um serviço em nuvem chamado Docker Hub [Hub.docker 2016]; e uma ferramenta de gerenciamento de *clusters* e orquestração de contêineres, o Swarm. Quando nos referimos ao Docker de forma geral trata-se do Docker Engine.

"A propagação da tecnologia de contêineres deve-se principalmente ao Docker, que facilitou o seu uso para os desenvolvedores por meio de uma interface de usuário amigável e imagens portáteis", afirma [Silva 2017].

> A execução de um contêiner com o uso de uma tecnologia como o LXC (Linux Container, ferramenta de virtualização a nível de sistema operacional que permite executar múltiplos sistemas Linux - denominados contêiners - usando um único *kernel*) demandava conhecimento especializado e trabalho manual significativo.

[Mouat 2016]

O Docker Hub disponibiliza um grande número de imagens de contêineres públicas para *download*, permitindo que os usuários comecem a fazer uso rapidamente da tecnologia.

Os componentes principais da plataforma Docker, de acordo com Jernigan [Tiffanyfay 2016] e Turnbull [TURNBULL 2016], são:

- **Docker Engine:** O *daemon* que gerencia imagens, contêineres, volumes, redes, entre outros.
- **Docker Client:** É o componente que interage com o daemon do Docker *Engine*.
- **Dockerfiles:** É um arquivo de texto simples, composto por uma sequência de comandos, que funciona como uma "receita" para a criação de Docker Images.
- **Docker Images:** A base da criação de contêineres, uma imagem é composta da união de camadas de sistemas de arquivos, empilhadas umas sobre as outras.
- **Docker Registry:** São repositórios de imagens.
- **Docker Containers:** Os contêineres propriamente ditos, instâncias criadas a partir de imagens.
- **Docker Compose:** Possibilita a criação conjunta de grupos de contêineres para um propósito comum.
- **Docker Swarm:** Possibilita a orquestração de clusters de contêineres, permitindo a execução escalável de aplicações.

3 Metodologia

Para executar os experimentos foi configurado um ambiente de testes com a seguintes configurações:

- Sistema Operacional: Ubuntu 16.04 LTS
- Versão do *kernel*: Linux 4.15.0-39-generic x86_64
- Processador: Intel Core i5 2.30GHz 2 núcleos (2 lógicos por 1 físico) [Intel 2018]
- Memória RAM: DDR3 8GB
- Disco rígido: 500GB

Foi configurado um ambiente virtualizado, hipervisor tipo 2, com o software livre VirtualBox, além de um o orquestrador de contêiner pelo software Docker. Ambos foram utilizados nos experimentos e executados sobre o sistema operacional nativo.

3.1 PARSEC

Para a realização de avaliações de desempenho é necessário escolher um conjunto de aplicações, comparando os resultados de suas execuções no mesmo ambiente alvo. Para este trabalho foi escolhida a suite PARSEC [PARSEC 2009], a qual é largamente adotada na literatura para avaliação e comparação de desempenho em ambientes multiprocessados, como computação de alto desempenho (HPC) e nuvens computacionais.

> É um Repositório de Aplicações de Princeton para Computadores de Memória Compartilhada (PARSEC) e também, um conjunto de referência para estudos de microprocessadores (CMPs). O PARSEC inclui aplicativos emergentes em reconhecimento, mineração e síntese (RMS), bem como aplicativos de sistemas que imitam programas comerciais de vários segmentos em larga escala.
>
> Adaptado de [Bienia et al. 2008]

"O *benchmarking* é a base quantitativa da pesquisa em arquitetura de computadores. O tempo de execução do programa é a única maneira exata de medir o desempenho", adaptação de [Hennessy e Patterson. 2003].

Um dos propósitos do PARSEC é estabelecer uma seleção de programas que fosse grande e diversa para ser suficientemente representativa para estudos científicos.

Fundamenta-se em 9 aplicações e 3 *kernels* que foram escolhidos de um amplo conjunto de domínios de aplicação. As Tabelas 1 e 2 apresentam um resumo qualitativo de suas principais características. As cargas de trabalho do PARSEC foram selecionadas para incluir diferentes combinações de modelos paralelos, requisitos de máquina e comportamentos de tempo de execução.

Tabela 1 – Domínio de aplicação e paralelização dos *benchmarks* PARSEC.

Programa	Domínio de Aplicação	Paralelização	
		Modelo	Granularidade
blackscholes	Análise Financeira	dados	grosseira
bodytrack	Visão Computacional	dados	médio
canneal	Engenharia	não estruturada	boa
dedup	Armazenamento Empresarial	pipeline	médio
facesim	Animação	dados	grosseira
ferret	Pesquisa de Similaridade	pipeline	médio
fluidanimate	Animação	dados	boa
freqmine	Mineração de Dados	dados	médio
streamcluster	Mineração de Dados	dados	médio
swaptions	Análise Financeira	dados	grosseira
vips	Processamento de Mídia	dados	grosseira
x264	Processamento de Mídia	pipeline	grosseira

Fonte: Adaptado de [Bienia et al. 2008].

Tabela 2 – Conjunto de trabalho e uso de dados dos *benchmarks* PARSEC.

Programa	Conjunto de Trabalho	Uso de Dados	
		Compartilhamento	Permuta
blackscholes	pequeno	baixo	baixa
bodytrack	médio	alto	alta
canneal	ilimitado	alto	alta
dedup	ilimitado	alto	alta
facesim	amplo	baixo	média
ferret	ilimitado	alto	alta
fluidanimate	amplo	baixo	média
freqmine	ilimitado	alto	média
streamcluster	médio	baixo	média
swaptions	médio	baixo	baixa
vips	médio	baixo	média
x264	médio	alto	alta

Fonte: Adaptado de [Bienia et al. 2008].

As cargas de trabalho são diversas e foram escolhidas de diversas áreas, como visão computacional, processamento de mídia, finanças computacionais, servidores corporativos e física de animação.

O PARSEC define 6 conjuntos de entrada para cada *benchmark*:

- ***Test*:** Um conjunto de entrada muito pequeno para testar a funcionalidade básica do programa.

- ***Simdev*:** Um conjunto de entrada muito pequeno que garante um comportamento básico do programa semelhante ao comportamento real, destinado ao teste e desenvolvimento do simulador.
- ***Simsmall, simmedium* e *simlarge*:** Conjuntos de entrada de diferentes tamanhos adequados para estudos de microarquiteturas com simuladores.
- ***Native*:** Um grande conjunto de entrada destinado à execução nativa.

Test e *simdev* são meramente destinados a testes e desenvolvimento e não devem ser usados para estudos científicos. As três entradas do simulador para estudos variam em tamanho, mas a tendência geral é que conjuntos de entrada maiores contenham conjuntos de trabalho maiores e mais paralelismo. Finalmente, o conjunto de entrada *native* é destinado a medições de desempenho em máquinas reais e excede as demandas computacionais que são geralmente consideradas viáveis para simulação por ordens de grandeza. Do ponto de vista científico, o conjunto de entrada *native* é o mais interessante porque se assemelha mais a entradas de programas reais.

Para o presente trabalho, foram escolhidos os programas *facesim* e *ferret*. Entre os critérios para a escolha podem ser destacados: aplicações de domínios heterogêneos, modos de paralelização distintos e cargas de trabalho diferentes.

3.1.1 Facesim

Um dos programas escolhidos para as análises de eficiência foi o *facesim*. Este aplicativo Intel RMS foi originalmente desenvolvido pela Universidade de Stanford. É preciso um modelo de uma face humana e uma sequência temporal de ativações musculares para calcular uma animação visualmente realista da face modelada, simulando o comportamento real [Sifakis, Neverov e Fedkiw 2005] [Teran et al. 2005].

A carga de trabalho foi inserida no pacote de *benchmark* porque um número crescente de jogos de computador e outras formas de animação empregam simulação física para criar um ambiente virtual mais realista. Os rostos humanos, em particular, são observados com mais atenção pelos usuários do que outros detalhes de um mundo virtual, tornando sua apresentação realista um elemento-chave para as animações.

Os conjuntos de entrada do *facesim* usam a mesma malha de faces. A redução da resolução da malha para criar tamanhos de entrada mais tratáveis é impraticável. Uma redução do número de elementos no modelo resultaria em sub-resolução da ação muscular e causaria problemas para detecção de colisão [Hughes et al. 2007]. Os conjuntos de entrada para o *facesim* são definidos da seguinte maneira:

- ***Test*:** Imprime a mensagem de ajuda.

Figura 4 – Animações visualmente realistas de um rosto humano criado pelo *facesim*.

Fonte: [Parsec-tutorial 2009].

- ***Simdev*:** 80.598 partículas, 372.126 tetraedros, 1 *frame*.
- ***Simsamll, simmedium* e *simlarge*:** O mesmo que `simdev`.
- ***Native*:** 80.598 partículas, 372.126 tetraedros, 100 *frames*.

3.1.2 Ferret

Outro programa escolhido para as análises de eficiência e desempenho foi o *ferret*. Esta aplicação é utilizada para pesquisa de similaridade baseada em conteúdo de dados ricos em recursos, como áudio, imagens, vídeo, formas 3D e assim por diante [Lv et al. 2006].

O *ferret* tem um mecanismo de pesquisa que encontra um conjunto de imagens semelhantes a uma imagem de consulta, analisando seu conteúdo. É paralelizado usando o modelo de *pipeline* com seis etapas. O primeiro e o último estágio são para entrada e saída. Os quatro estágios intermediários são para segmentação da imagem de consulta, extração de recurso, indexação de conjuntos de imagens semelhantes com *Locality Sensitive Hashing* (LSH) [Lv et al. 2007] e classificação. Cada estágio possui seu próprio conjunto de *threads* e a unidade base de trabalho do *pipeline* é uma imagem de consulta.

Segmentação é o processo de decompor uma imagem em áreas separadas que exibem objetos diferentes. A lógica por trás dessa etapa é que, em muitos casos, apenas partes de uma imagem são de interesse, como o primeiro plano. A segmentação permite que os estágios subseqüentes atribuam um peso maior às partes da imagem que são consideradas relevantes e parecem está juntas. Após a segmentação, o *ferret* extrai um vetor de recursos de cada segmento. Um vetor de recurso é uma descrição matemática multidimensional do conteúdo do segmento. Codifica propriedades fundamentais como cor, forma e área.

Depois que os vetores de recursos são conhecidos, o estágio de indexação pode consultar o banco de dados de imagens para obter um conjunto de imagens semelhantes. O banco de dados é organizado como um conjunto de tabelas de hash indexadas com LSH [Lv et al. 2007] com várias ligações.

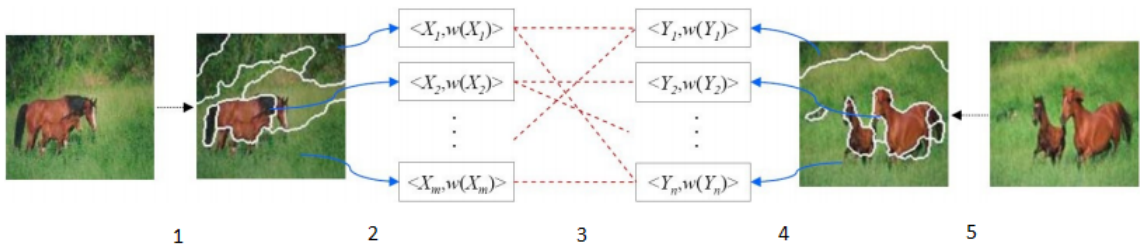
A estimativa de similaridade é obtida analisando e pesando os pares de distância entre os segmentos da imagem de entrada e as imagens semelhantes. A métrica subjacente empregada é a Earth Mover's Distance (EMD) [Rubner, Tomasi e Guibas 2000].

Para duas imagens X e Y, é definido como

$$\boxed{EMD(X,Y) = min\sum_i\sum_j f_{ij}\mathrm{d}(X_i,Y_j)}$$

onde X_i e Y_j denotam segmentos de X e Y e f_{ij} é a extensão na qual X_i é correspondido a Y_j. Os passos fundamentais do algoritmo podem ser vistos na Figura 5. As transições 1 e 5 representam a segmentação, 2 e 4 a extração de recurso e a 3 a distância computacional.

Figura 5 – Algorítmo do *ferret*.



Fonte: Adapatado de [Parsec-tutorial 2009].

Os conjuntos de entrada para o *ferret* são definidos da seguinte maneira:

- ***Test*:** 1 consulta de imagem, banco de dados com 1 imagem, encontrar 1 imagem.
- ***Simdev*:** 4 consultas de imagens, banco de dados com 100 imagens, encontrar as 5 principais imagens.
- ***Simsmall*:** 16 consultas de imagens, banco de dados com 3.544 imagens, encontrar as 10 principais imagens.
- ***Simmedium*:** 64 consultas de imagens, banco de dados com 13.787 imagens, encontrar as 10 principais imagens.
- ***Simlarge*:** 256 consultas de imagens, banco de dados com 34.973 imagens, encontrar as 10 principais imagens.

- ***Native*:** 3.500 consultas de imagens, banco de dados com 59.695 imagens, encontrar as 50 principais imagens.

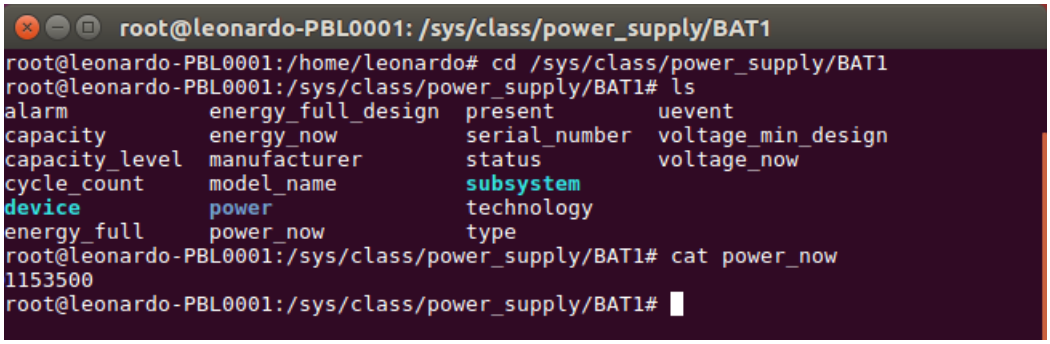
3.2 Obtenção da potência dissipada

Para fazer a comparação da eficiência energética entre máquina virtual e contêiner, foi necessário obter a taxa da potência dissipada em tempo real. Foram analisados dois métodos, o primeiro e o que foi utilizado nos testes práticos, foi acessar um arquivo do sistema operacional que informa a potência dissipada em microwatts (μW). O segundo método foi a instalação de uma interface de usuário no próprio terminal para monitoramento, chamada Stress-Terminal UI [Edivaldobrito 2018] ou "s-tui".

3.2.1 Monitoramento de potência a partir do SO

Existe um arquivo no sistema operacional Linux que registra a potência dissipada em tempo real: *"/sys/class/power_supply/BAT1/power_now"*. A Figura 6 mostra o procedimento realizado para o acesso ao referido recurso.

Figura 6 – Acesso ao arquivo *power_now*.



Fonte: Autoria Própria.

3.2.2 Stress-Terminal UI

Stress-Terminal UI ou "s-tui" é uma ferramenta de terminal projetada para facilitar o teste e monitoramento do *hardware* e um computador executando Linux. Por padrão, s-tui tentará mostrar todos os sensores que puder detectar no sistema (ilustrados na Figura 7) que são seguintes:

- Temperatura;
- Frequência;
- Utilização;
- Potência.

Figura 7 – Ferramenta em execução.

Fonte: Autoria Própria.

3.2.3 Rotina de monitoramento

Para automatizar a coleta de dados, foi elaborado um código em *Shell script* para gerar um arquivo .txt contendo os valores registrados a cada segundo. O código da Figura 8 mostra a função *get_power*, a qual extrai o valor atual da potência dissipada, em microwatts (μW).

Figura 8 – Função *get_power*.

```
1  #!/bin/sh
2  set -eu
3  get_power(){
4       local power_uW
5       local voltage_uV
6       local current_uA
7
8       if [ -f "sys/class/power_supply/BAT1/power_now"        ];
   then
9            power_uW=$(cat sys/class/power_supply/BAT1/
   power_now)
10      else
11           voltage_vW=$(cat sys/class/power_supply/BAT1/
   voltage_now)
12           current_uA=$(cat sys/class/power_supply/BAT1/
   current_now)
13           power_uW=$(echo "scale=2; $current_uA *
   $voltage_uV /1000000.0" | bc)
14      fi
15      echo "scale=2; $power_uW /1000000.0" | bc
16      }
```

Fonte: Autoria Própria.

O código da Figura 9, mostra que o laço *while* foi programado para quando a variável `contador` chegar a 20, o script parar de coletar os dados, ou seja, serão obtidos 20 valores de potência na saída do código. Outro fator importante é o *status* da bateria, caso esteja com a bateria totalmente carregada, irá retornar que a capacidade da bateria está *"full"*, se estiver carregando (com o carregador conectado ao notebook), irá informar a porcentagem da bateria. A única opção que informará o desejado (*status=Discharging*), é se o notebook não estiver com o carregador conectado a bateria.

Figura 9 – Função principal do código.

```
1  status=$(cat $sys/class/power_supply/BAT1/status)
2  capacity=$(cat $sys/class/power_supply/BAT1/capacity)
3  contador=0
4  while [ $contador -lt 20 ]
5  do
6      #echo "$contador"
7      sleep 1
8      contador=$((contador+1))
9      case $status in
10         Full)
11             printf '%d%%\tFull\n' "${capacity}"
12             ;;
13         Discharging)
14             power_now=$(get_power)
15             printf '%.4fW\n' "${power_now}"
16             ;;
17         Charging)
18             printf '%d%%\tCharging\n' "${capacity}"
19             ;;
20         *)
21             echo "Battery has unknow status: '$status'"
       >&2
22             exit 1
23             ;;
24     esac
25 done
```

Fonte: Autoria Própria.

Para executar o código, deve-se inserir os seguintes comandos:

```
1 chmod a+x script.sh
2 ./script >> resultados.txt
```

Na primeira linha de comando o `chmod` muda as permissões, o modo como um arquivo é tratado, o `a` indica *all* de todos os usuários e o `+x` é para tornar o arquivo .sh executável. Com o arquivo executável gerado, basta executá-lo com o comando da linha 2 para obter os dados. O ">> resultados.txt" significa que a saída dos dados será armazenada em um arquivo .txt, para posterior análise e elaboração de gráficos e tabelas.

3.3 Cálculos estatísticos

Considerando o ambiente multitarefa do sistema operacional Linux, a avaliação de resultados deve levar em consideração a interferência de outros processos em execução

no mesmo ambiente. Em especial, *daemons* e estruturas de gerenciamento auxiliares do *kernel* estão sempre em execução e podem interferir em qualquer avaliação quantitativa.

Desta forma, para efetuar uma comparação adequada, todos os testes foram executados múltiplas vezes e ferramental estatístico foi adotado para avaliar a confiabilidade dos dados obtidos.

Em teoria das probabilidades e estatística, o coeficiente de variação (CV), também conhecido como desvio padrão relativo (DPR), é uma medida padronizada de dispersão de uma distribuição de probabilidade ou de uma distribuição de frequências. É frequentemente expresso como uma porcentagem, sendo definido como a razão do desvio padrão pela média. O CV ou DPR é comumente usado em campos como engenharia e física quando se fazem estudos de garantia de qualidade e avaliações de repetitividade e reprodutibilidade [Lima 2001].

Para o cálculo do coeficiente de variação, deve-se saber primeiro o desvio padrão e consequentemente a média. Estes cálculos foram realizados com as fórmulas mostradas na Tabela 3.

Tabela 3 – Fórmulas para cálculo da média, desvio padrão e CV.

Média	Desvio Padrão	Coeficiente de Variação (CV)
$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$	$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}}$	$\mathrm{C}_v = \frac{\sigma}{\overline{x}} * 100$

Fonte: Adaptado de [Lima 2001].

4 Resultados

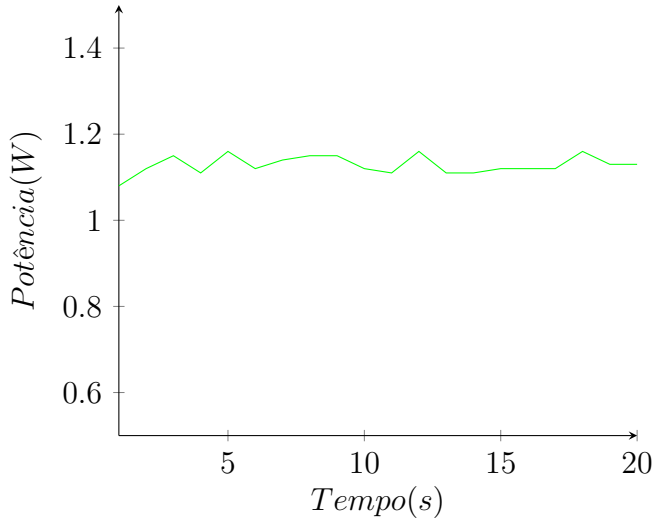
Para o trabalho, foi realizada, primeiramente, uma avaliação do consumo energético do ambiente em estado ocioso. O script apresentado na seção 3.2.3 foi executado 10 vezes e, para cada execução, foi gerado um arquivo `.txt` com os valores de potência dissipada. Em seguida, foi calculada a potência média dissipada em Watts a cada segundo, além do cálculo do coeficiente de variação.

Tabela 4 – Resultados obtidos da utilização normal do processador.

Tempo (s)	Média (W)	CV	Tempo (s)	Média (W)	CV
1	1,08	$\pm 1,33\%$	11	1,11	$\pm 5,99\%$
2	1,12	$\pm 4,74\%$	12	1,16	$\pm 4,97\%$
3	1,15	$\pm 4,68\%$	13	1,11	$\pm 6,19\%$
4	1,11	$\pm 5,72\%$	14	1,11	$\pm 5,83\%$
5	1,16	$\pm 5,70\%$	15	1,12	$\pm 5,54\%$
6	1,12	$\pm 7,13\%$	16	1,12	$\pm 4,77\%$
7	1,14	$\pm 5,93\%$	17	1,12	$\pm 6,55\%$
8	1,15	$\pm 5,05\%$	18	1,16	$\pm 6,94\%$
9	1,15	$\pm 5,49\%$	19	1,13	$\pm 5,30\%$
10	1,12	$\pm 4,90\%$	20	1,13	$\pm 5,90\%$

Fonte: Autoria Própria.

Figura 10 – Taxa de Watts a cada segundo na utilização normal do processador



Fonte: Autoria Própria.

Os resultados, mostrados na Tabela 4 e na Figura 10, apresentam médias entre 1,08 e 1,16 W, considerando que não havia "perturbação" no sistema e, de acordo com os cálculos, o CV não foi considerado alto.

Os testes práticos foram feitos no sistema operacional nativo, máquina virtual e no contêiner, para dois programas do PARSEC, o *facesim* e o *ferret*, apresentados nas seções 3.1.1 e 3.1.2 respectivamente.

4.1 Facesim

4.1.1 Execução em ambiente nativo

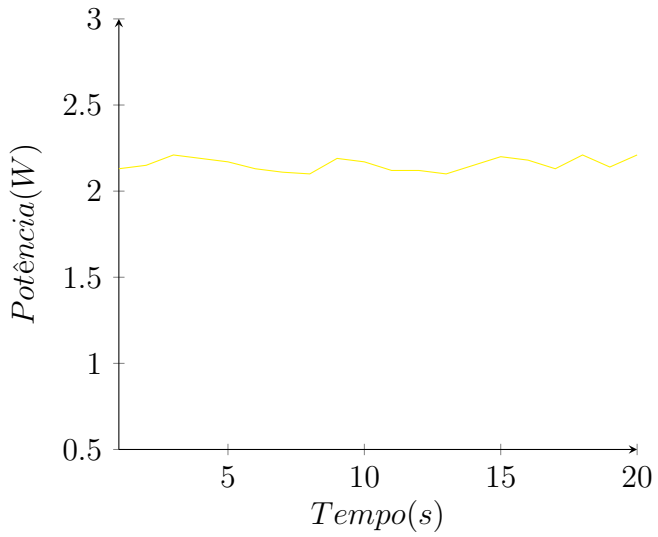
Após os testes iniciais sem perturbação do sistema, foram feitas as práticas utilizando a aplicação PARSEC *benchmark* no ambiente nativo, executando 10 vezes o script apresentado na seção 3.2.3 para obter a média da potência e o coeficiente de variação. Os resultados são mostrados na Tabela 5 e na Figura 11, em comparação com os valores obtidos no estado normal da máquina, sem utilizar as aplicações da suíte PARSEC, as taxas médias de Watts aumentaram em aproximadamente 1 W, visto que a aplicação exige mais dos recursos do *hardware*.

Tabela 5 – Resultados obtidos da execução do *facesim* no ambiente nativo.

Tempo (s)	Média (W)	CV	Tempo (s)	Média (W)	CV
1	2,13	$\pm 2,50\%$	11	2,12	$\pm 6,33\%$
2	2,15	$\pm 5,32\%$	12	2,12	$\pm 4,79\%$
3	2,21	$\pm 4,28\%$	13	2,10	$\pm 2,56\%$
4	2,19	$\pm 7,03\%$	14	2,15	$\pm 5,95\%$
5	2,17	$\pm 5,44\%$	15	2,20	$\pm 1,71\%$
6	2,13	$\pm 5,56\%$	16	2,18	$\pm 2,52\%$
7	2,11	$\pm 4,17\%$	17	2,13	$\pm 4,85\%$
8	2,10	$\pm 3,87\%$	18	2,21	$\pm 5,12\%$
9	2,19	$\pm 6,01\%$	19	2,14	$\pm 4,62\%$
10	2,17	$\pm 6,24\%$	20	2,21	$\pm 4,39\%$

Fonte: Autoria Própria.

Figura 11 – Taxa média de Watts a cada segundo na utilização da aplicação no ambiente nativo.



Fonte: Autoria Própria.

4.1.2 Execução em máquina virtual

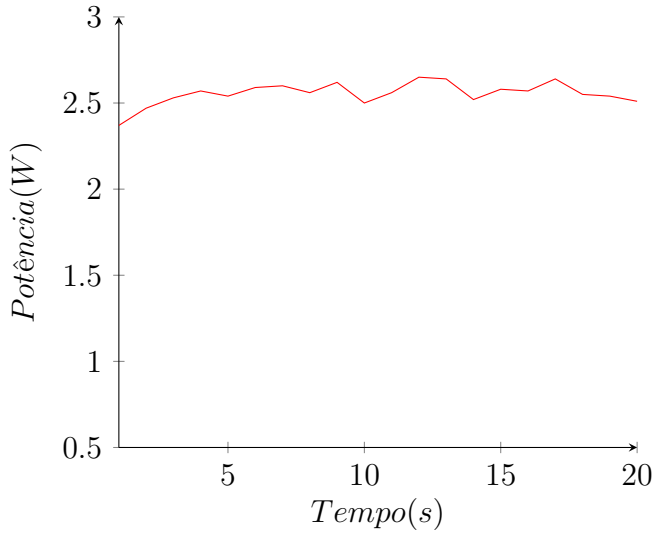
Os dados obtidos pela execução do *facesim* na máquina virtual, são mostrados na Tabela 6 com as médias da taxa de potência e o coeficiente de variação. A Figura 12 exibe os valores no tempo.

Tabela 6 – Resultados obtidos da execução do *facesim* na máquina virtual.

Tempo (s)	Média (W)	CV	Tempo (s)	Média (W)	CV
1	2,37	$\pm 2,28\%$	11	2,56	$\pm 7,16\%$
2	2,47	$\pm 4,26\%$	12	2,65	$\pm 4,37\%$
3	2,53	$\pm 6,18\%$	13	2,64	$\pm 4,56\%$
4	2,57	$\pm 6,31\%$	14	2,52	$\pm 5,59\%$
5	2,54	$\pm 7,46\%$	15	2,58	$\pm 5,39\%$
6	2,59	$\pm 5,34\%$	16	2,57	$\pm 6,52\%$
7	2,60	$\pm 5,33\%$	17	2,64	$\pm 4,41\%$
8	2,56	$\pm 1,99\%$	18	2,55	$\pm 5,82\%$
9	2,62	$\pm 6,37\%$	19	2,54	$\pm 5,62\%$
10	2,50	$\pm 6,24\%$	20	2,51	$\pm 4,39\%$

Fonte: Autoria Própria.

Figura 12 – Taxa média de Watts a cada segundo na utilização da aplicação na máquina virtual.



Fonte: Autoria Própria.

Os resultados, apresentados na Tabela 6 e na Figura 12, foram mais expressivos (taxas maiores de Watts no tempo), como era de se esperar, uma vez que a aplicação estava em execução, consumindo mais recusrso da máquina. A média variou entre 2,37 e 2,65 W, mais que o dobro da média em comparação a utilização normal. Os cálculos também mostraram que o CV teve oscilação na média entre $\pm 1,99\%$ e $\pm 7,46\%$.

4.1.3 Execução em contêiner

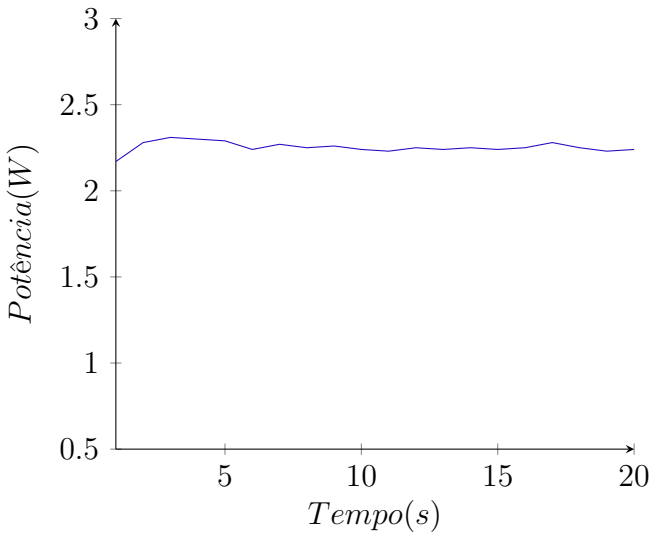
Após os resultados obtidos da utilização normal do processador e da aplicação na máquina virtual, foi feito o mesmo procedimento para obtenção dos dados da utilização da aplicação no contêiner.

Tabela 7 – Resultados obtidos da execução do *facesim* no contêiner.

Tempo (s)	Média (W)	CV	Tempo (s)	Média (W)	CV
1	2,17	$\pm 2,21\%$	11	2,23	$\pm 3,48\%$
2	2,28	$\pm 2,60\%$	12	2,25	$\pm 2,88\%$
3	2,31	$\pm 1,87\%$	13	2,24	$\pm 3,15\%$
4	2,30	$\pm 3,40\%$	14	2,25	$\pm 3,34\%$
5	2,29	$\pm 3,69\%$	15	2,24	$\pm 4,38\%$
6	2,24	$\pm 3,72\%$	16	2,25	$\pm 4,56\%$
7	2,27	$\pm 3,65\%$	17	2,28	$\pm 3,84\%$
8	2,25	$\pm 3,83\%$	18	2,25	$\pm 3,65\%$
9	2,26	$\pm 3,07\%$	19	2,23	$\pm 3,74\%$
10	2,24	$\pm 4,12\%$	20	2,24	$\pm 2,85\%$

Fonte: Autoria Própria.

Figura 13 – Taxa média de Watts a cada segundo na utilização da aplicação no contêiner.



Fonte: Autoria Própria.

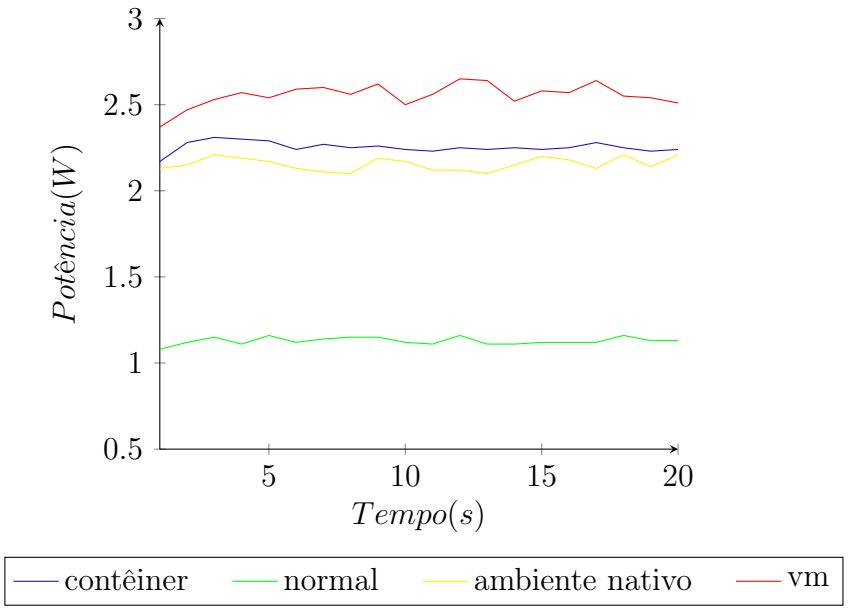
Os números, apresentados na Tabela 7 e na Figura 13, foram mais expressivos em relação a utilização normal do processador, no entanto, foram satisfatórios comparado a utilização da aplicação na máquina virtual. As médias da taxa de Watts ficaram entre 2,17 e 2,31 (médias inferiores às da Tabela 6) e, visualmente na Figura 13, pode-se perceber uma maior linearidade na curva em comparação aos outros gráficos.

4.1.4 Comparação dos gráficos com as médias das taxas

A Figura 14, que une os 4 gráficos anteriores, apresenta a diferença dos valores obtidos em cada teste. A curva traçada em verde representa a utilização normal do processador, em amarelo mostra a utilização do facesim no ambiente nativo, em azul apresenta a da utilização da aplicação no contêiner e a curva vermelha é a da utilização da aplicação na máquina virtual.

Analisando o gráfico, percebe-se que a execução em contêiner apresenta maior regularidade, além de menor consumo energético em comparação a máquina virtual. A curva de potência da execução no ambiente nativo, é bem similar a da execução no contêiner, pelo fato de que o contêiner não nescessita instanciar um sistema operacional completo como a máquina virtual do hipervisor tipo 2.

Figura 14 – Médias das taxas de Watts a cada segundo.



Fonte: Autoria Própria.

4.2 Ferret

4.2.1 Execução em ambiente nativo

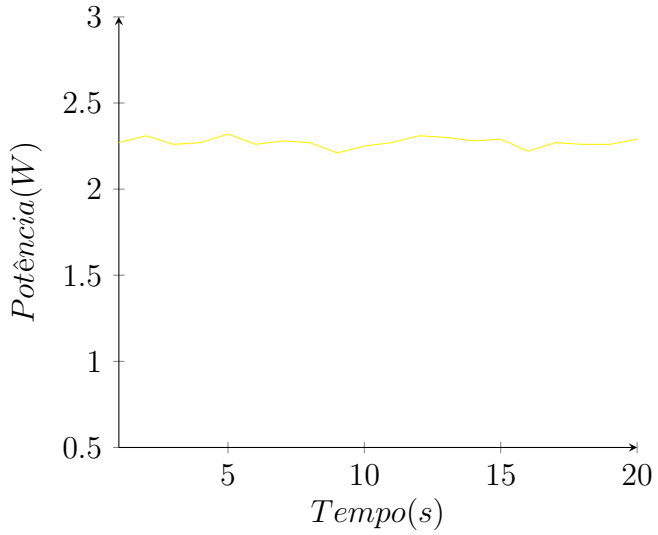
Concluídos os experimentos com o *facesim*, bastou repetir os mesmos procedimentos com o programa *ferret*. Começando pela execução no ambiente nativo, a Tabela 8 exibe as médias da taxa de potência e o coeficiente de variação. A Figura 15 exibe os valores no tempo.

Tabela 8 – Resultados obtidos da execução do *ferret* no ambiente nativo.

Tempo (s)	Média (W)	CV	Tempo (s)	Média (W)	CV
1	2,27	±4, 21%	11	2,27	±7, 13%
2	2,31	±2, 72%	12	2,31	±4, 79%
3	2,26	±5, 30%	13	2,30	±4, 56%
4	2,27	±1, 89%	14	2,28	±2, 45%
5	2,32	±5, 04%	15	2,29	±4, 39%
6	2,26	±5, 89%	16	2,22	±2, 22%
7	2,28	±5, 17%	17	2,27	±6, 57%
8	2,27	±4, 02%	18	2,26	±4, 19%
9	2,21	±5, 38%	19	2,26	±3, 23%
10	2,25	±3, 24%	20	2,29	±5, 03%

Fonte: Autoria Própria.

Figura 15 – Taxa média de Watts a cada segundo na utilização da aplicação no ambiente nativo.



Fonte: Autoria Própria.

4.2.2 Execução em máquina virtual

Os dados obtidos pela execução do *ferret* na máquina virtual, são mostrados na Tabela 9 com as médias da taxa de potência e o coeficiente de variação. A Figura 16 exibe os valores no tempo.

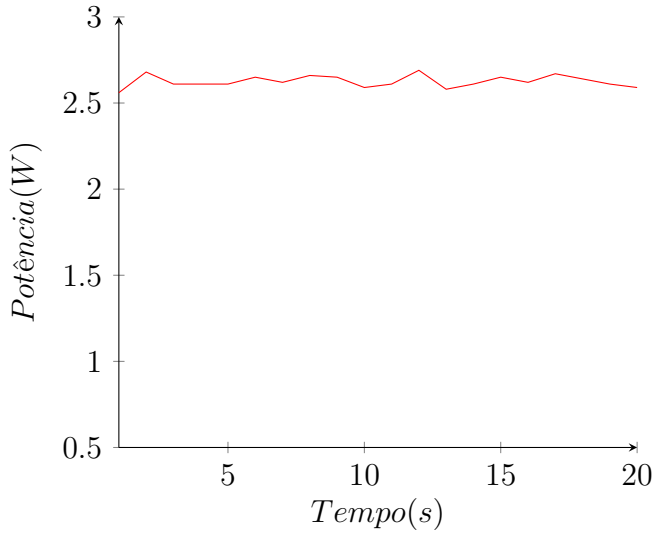
Comparado ao teste feito com a outra aplicação na máquina virtual, o *ferret* teve um aumento na média das taxas de potência em função do tempo. Pode-se justificar pelo alto compartilhamento e permuta no uso de dados durante a execução do programa.

Tabela 9 – Resultados obtidos da utilização do *ferret* na máquina virtual.

Tempo (s)	Média (W)	CV	Tempo (s)	Média (W)	CV
1	2,56	$\pm 3,04\%$	11	2,61	$\pm 4,17\%$
2	2,68	$\pm 3,15\%$	12	2,69	$\pm 2,83\%$
3	2,61	$\pm 4,57\%$	13	2,58	$\pm 3,88\%$
4	2,61	$\pm 5,34\%$	14	2,61	$\pm 5,21\%$
5	2,61	$\pm 3,57\%$	15	2,65	$\pm 2,40\%$
6	2,65	$\pm 4,33\%$	16	2,62	$\pm 3,73\%$
7	2,62	$\pm 3,55\%$	17	2,67	$\pm 3,76\%$
8	2,66	$\pm 4,55\%$	18	2,64	$\pm 4,04\%$
9	2,65	$\pm 3,87\%$	19	2,61	$\pm 3,64\%$
10	2,59	$\pm 3,15\%$	20	2,59	$\pm 2,14\%$

Fonte: Autoria Própria.

Figura 16 – Taxa média de Watts a cada segundo na utilização da aplicação na máquina virtual.



Fonte: Autoria Própria.

4.2.3 Execução em contêiner

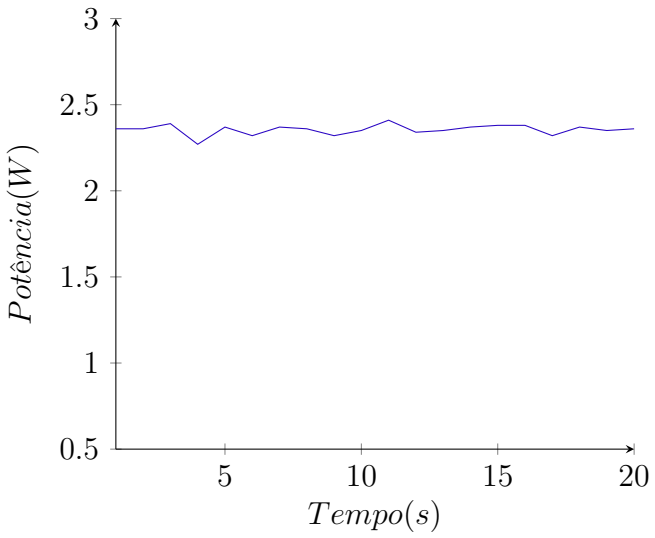
Em seguida, foi feito o mesmo procedimento para obtenção dos dados da utilização da aplicação no contêiner. A Tabela 10 e a Figura 17 mostram os resultados. A média da potência em função do tempo teve um pequeno aumento em relação ao uso do *facesim* no contêiner, entretanto, foi inferior ao uso da máquina virtual, provando novamente um melhor desempenho e eficiência energética. A média variou entre 2,27 e 2,41 W e, os cálculos também mostraram que o CV não teve uma alta taxa, oscilando na média entre $\pm 2,33\%$ e $\pm 4,84\%$.

Tabela 10 – Resultados obtidos da utilização do *ferret* no contêiner.

Tempo (s)	Média (W)	CV	Tempo (s)	Média (W)	CV
1	2,36	$\pm 3,12\%$	11	2,41	$\pm 2,88\%$
2	2,36	$\pm 4,07\%$	12	2,34	$\pm 4,01\%$
3	2,39	$\pm 3,65\%$	13	2,35	$\pm 2,33\%$
4	2,27	$\pm 3,15\%$	14	2,37	$\pm 3,54\%$
5	2,37	$\pm 3,42\%$	15	2,38	$\pm 4,20\%$
6	2,32	$\pm 4,17\%$	16	2,38	$\pm 3,64\%$
7	2,37	$\pm 2,86\%$	17	2,32	$\pm 4,84\%$
8	2,36	$\pm 3,91\%$	18	2,37	$\pm 4,17\%$
9	2,32	$\pm 4,41\%$	19	2,35	$\pm 3,78\%$
10	2,35	$\pm 2,97\%$	20	2,36	$\pm 2,95\%$

Fonte: Autoria Própria.

Figura 17 – Taxa média de Watts a cada segundo na utilização da aplicação no contêiner.

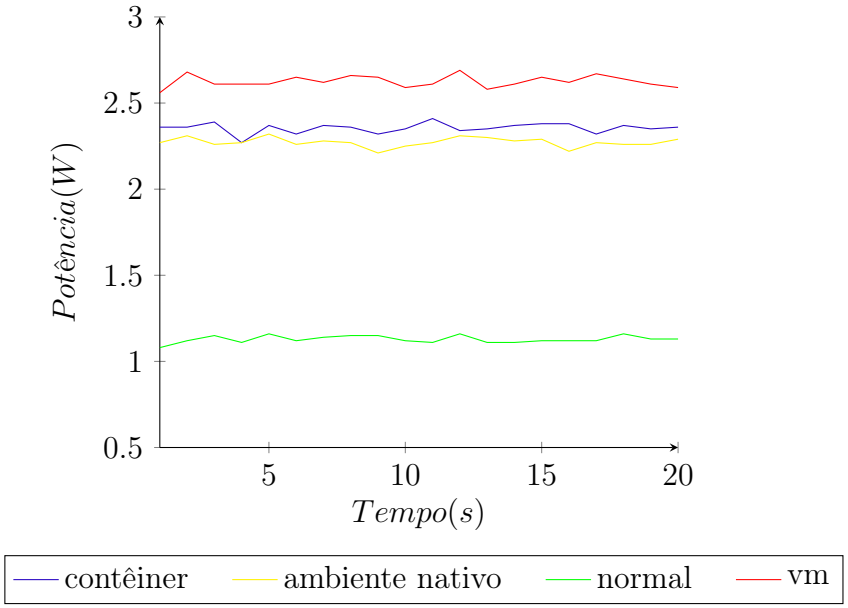


Fonte: Autoria Própria.

4.2.4 Comparação dos gráficos com as médias das taxas

Assim como no gráfico comparativo do programa *facesim*, o mesmo aconteceu na Figura 18 para o *ferret*, a curva em vermelho, que representa a Figura 16, mostra o consumo energético do programa na máquina virtual, com taxas médias acima de 2,5 W, enquanto a curva em azul faz referência à Figura 17 e indica o contêiner, com valor consumo máximo, em média, de 2,41 W. Assim como verificado na comparação de execuções do *facesim* (4.1.4), no *ferret* a execução em contêiner apresenta resultados muito próximos à execução nativa.

Figura 18 – Médias das taxas de Watts a cada segundo.



Fonte: Autoria Própria.

4.3 Uso da memória e do disco

Para monitorar o uso da memória durante a execução do *facesim* e do *ferret* na máquina virtual e no contêiner foi utilizado o utilitário *pmap* [Elias Praciano 2017] que determina a taxa de memória utilizada por um processo no Linux. Basta determinar o PID através do *htop* [Vivaolinux 2010] e executar o seguinte comando:

```
1 pmap [PID] | grep total
```

e ele exibirá o total de memória (em kB) usada pelo processo. O | grep total é utilizado para obter uma visualização mais resumida, contendo apenas o valor total da memória. Os resultados do uso da memória são apresentados na Tabela 11.

O monitoramento do disco foi realizado pelo *iotop* [Oanalista 2016]. O comando:

```
1 iotop -p [PID]
```

informa em tempo real a taxa da escrita no disco (em kB/s). Foram extraídos os valores máximos de cada teste. Os resultados da taxa de escrita no disco são apresentados na Tabela 12.

Tabela 11 – Uso da memória.

	Máquina Virtual	Contêiner	Diferença
Facesim	3.797.500 kB	336.524 kB	1028%
Ferret	3.934.024 kB	478.164 kB	723%

Fonte: Autoria Própria.

Tabela 12 – Uso do disco.

	Máquina Virtual	Contêiner	Diferença
Facesim	324,39 kB/s	3,99 kB/s	8030%
Ferret	1138,75 kB/s	23,97 kB/s	4650%

Fonte: Autoria Própria

A utilização da memória pelo processo do *facesim* na máquina virtual foi aproximadamente 11 vezes maior que no contêiner e, o do *ferret* também teve a utilização muito superior na máquina virtual. O uso do disco, em ambas as aplicações tiveram a taxa muito superior na prática da máquina virtual.

É óbvio que a discrepância entre os valores apresentados não decorre somente por conta da diferença de execução da mesma tarefa entre o contêiner e a máquina virtual, mas principalmente da distinção estrutural entre elas. Máquinas virtuais precisam de muito mais memória e acesso a disco por conta do sistema operacional completo que precisam executar para dar suporte ao funcionamento da aplicação. No contêiner, isto não é necessário.

4.4 Tempos de execução

Além das obtenções de média da taxa de W em função do tempo, do coeficiente de variação, uso de memória e disco, também foram extraídos as médias dos tempos de execução dos programas, executados 10 vezes para cada entrada disponível, na máquina virtual e no contêiner. Os resultados são compilados nas Tabelas 13 e 14.

Tabela 13 – Tempos de execução do *facesim*.

Entradas	Máquina Virtual	Contêiner	Ambiente nativo
Simdev	0m6.385s	0m5.592s	0m5.469s
Simsamall	0m6.434s	0m5.641s	0m5.526s
Simmedium	0m6.657s	0m5.671s	0m5.501s
Simlarge	0m6.572s	0m5.660s	0m5.542s
Native	7m26.737s	6m37.556s	6m29.613s

Fonte: Autoria Própria.

Tabela 14 – Tempos de execução do *ferret*.

Entradas	Máquina Virtual	Contêiner	Ambiente nativo
Simdev	0m0.077s	0m0.028s	0m0.035s
Simsamall	0m0.850s	0m0.338s	0m0.335s
Simmedium	0m2.885s	0m1.089s	0m1.024s
Simlarge	0m14.959s	0m4.701s	0m4.302s
Native	11m17.843s	6m32.724s	6m0.564s

Fonte: Autoria Própria.

Como foi explicado na seção 3.1.1 a aplicação utilizada tem 6 tipos de entradas definidas (uma delas é o *test* que imprime apenas uma mensagem de ajuda na tela), todos os testes realizados para análise de potência foram com a entrada *native*, destinada a medições de desempenho em máquinas reais e excede as demandas computacionais que são geralmente consideradas viáveis para simulação por ordens de grandeza. Do ponto de vista científico, esse conjunto de entrada é o mais interessante porque se assemelha mais a entradas de programas reais. Os outros conjuntos de entrada podem ser considerados aproximações mais grosseiras que sacrificam a precisão para tratabilidade.

Tomando os tempos de utilização das aplicações no ambiente nativo, para servir apenas de teste base, mais uma vez o comparativo apontou por uma eficiência maior na utilização por contêiner em relação a máquina virtual. Em todos os testes do *facesim* e do *ferret*, com os 5 tipos de entradas disponíveis, o tempo de execução da aplicação no contêiner foi inferior ao da máquina virtual. Isso se justifica pela estrutura enxuta do contêiner, permitindo que a aplicação demande muito menos recursos, como disco e memória e, consequentemente, menos potência.

4.5 Eficiência energética

Para determinar a eficiência energética, é necessário avaliar o consumo das aplicações em sua execução completa. No entanto, a métrica disponível a partir da leitura dos sensores disponíveis no ambiente de testes adotado é a potência média a cada segundo (em microwatts). Sendo assim, faz-se necessário converter os resultados obtidos para uma métrica de energia (joules).

Calçada [Calçada 2008] afirma que a potência constante (P) é dada pela razão do trabalho total realizado (W) no intervalo de tempo Δt :

$$P = \frac{W}{\Delta t}$$

$$W = P.\Delta t$$

Na Figura 17 (execução do *ferret* em contêiner), podemos afirmar que a área sob a curva é numericamente igual ao trabalho realizado no intervalo de tempo. Logo, seu valor é dado pela integral da potência em relação ao tempo:

$$W = \int_{t_1}^{t_2} P dt$$

Esta fórmula foi aplicada aos resultados obtidos durante a execução completa de cada aplicação em máquina virtual, contêiner e ambiente nativo. Todos os testes foram feitos com a entrada *native*. Os resultados entre a máquina virtual e o contêiner estão compilados na Tabela 15.

Tabela 15 – Eficiência energética entre máquina virtual e contêiner.

	Máquina virtual	Contêiner	Diferença
Facesim	1115,58 J	892,61 J	25%
Ferret	1776,81 J	924,04 J	92%

Fonte: Autoria Própria.

O comparativo energético apontou por uma eficiência maior na utilização por contêiner. Nos testes do *facesim* e do *ferret*, com a entrada *native*, o gasto energético da execução das aplicações no contêiner foi inferior ao da máquina virtual.

Também foram feitas as comparações entre as aplicações executadas no ambiente nativo e contêiner (Tabela 16) e entre o ambiente nativo e a máquina virtual (Tabela 17).

Tabela 16 – Comparação de consumo energético entre contêiner e ambiente nativo.

	Contêiner	Ambiente nativo	Diferença
Facesim	892,61 J	840,24 J	6%
Ferret	924,04 J	825,88 J	12%

Fonte: Autoria Própria.

Tabela 17 – Comparação de consumo energético entre máquina virtual e ambiente nativo.

	Máquina virtual	Ambiente nativo	Diferença
Facesim	1115,58 J	840,24 J	33%
Ferret	1776,81 J	825,88 J	115%

Fonte: Autoria Própria.

O tempo de execução de cada aplicação influencia no resultado da eficiência energética. Mantida a potência média, quanto maior for o tempo de execução, maior o gasto energético. A aplicação *ferret*, quando executada no ambiente virtualizado VirtualBox, apresenta significativo incremento de tempo para conclusão, pois faz buscas recorrentes no banco de dados (acesso ao disco) para fazer as comparações entre as imagens. Como a máquina virtual instancia um sistema operacional completo, a emulação de disco impacta no desempenho da aplicação, a qual demanda tempo de execução consideravalmente maior quando comparado ao contêiner e ao ambiente nativo.

5 Conclusão e trabalhos futuros

Recapitulando o que foi apresentado, este trabalho se propôs a realizar uma análise de eficiência energética e desempenho, entre contêineres e máquinas virtuais. Inicialmente neste estudo, preocupou-se em conhecer com maior profundidade a tecnologia de contêineres e máquinas virtuais, seu funcionamento e evolução. As principais diferenças entre tipos de virtualização e o uso da tecnologia de contêineres na computação em nuvem foram explorados. Em seguida, foi realizado um estudo sobre a suíte de *benchmarking* PARSEC, dando enfoque às aplicações utilizadas neste trabalho: *facesim* e *ferret*. Os conhecimentos adquiridos, juntamente com as informações obtidas a partir da revisão bibliográfica compuseram o embasamento científico e metodológico para a realização da avaliação de desempenho subsequente.

Os experimentos fizeram o uso de contêiner instanciado a partir da imagem Docker, especificada em arquivo *Dockerfile*, e elaborada especificamente para o trabalho; e do ambiente de virtualização Oracle VirtualBox, caracterizado por ser um hipervisor do tipo 2. A partir dos resultados obtidos nos experimentos foi concluído que a eficiência energética na execução de aplicações é maior na utilização em contêiner, em decorrência da menor sobrecarga que esta abordagem tem em comparação com máquinas virtuais. A potência média ao longo do tempo com a utilização de conteiners foi inferior em todos os testes realizados. O tempo médio de execução também foi inferior para todas as entradas disponíveis para o *facesim* e o *ferret*. As evidências apresentadas confirmam o que estava posto na literatura e a hipótese levantada no início do trabalho.

Além disso, utilizacão da memória e do disco também tiveram resultados satisfatórios a favor do contêiner. Desta forma, verifica-se que o potencial de consolidação de servidores é maior com o uso da conteinerização que da virtualização. Ainda neste sentido, a utilização da ferramenta Docker se mostrou intuitiva e capaz de promover a criação de uma infraestrutura simples e leve, mas capaz de hospedar os mesmos serviços que uma infraestrutura baseada em máquinas virtuais.

Algumas das principais dificuldades encontradas na realização deste trabalho foram devidos ao próprio ambiente de *hardware* adotado para os experimentos. Os problemas concentraram-se na falha de alguns componentes do *notebook* (como fontes de alimentação e placa de rede). Outra dificuldade, contornada ao longo dos estudos, foi encontrar uma meio de se extrair o valor da potência dissipada pelo equipamento a partir do sistema operacional Linux Ubuntu.

Dado que os experimentos foram realizados com a aplicação sendo executada em contêiner único, para trabalhos futuros pode-se considerar a avaliação do desempenho

dos contêineres trabalhando em *cluster* utilizando ferramentas como o Docker Swarm. Outro ponto que pode ser explorado é a utilização de hipervisores do tipo 1 (*bare metal*), comparado ao hipervisor tipo 2 e ao próprio contêiner. Além disso, foram considerados apenas dois programas da suíte PARSEC, escolhidos justamente por terem características bem distintas entre si. Dada a consistência dos resultados, julgamos a escolha adequada, especialmente levando-se em consideração o escopo deste trabalho. Outro ponto a ser aprimorado é o incremento do número de execuções, de modo a reduzir o coeficiente de variação, em conjunto com o uso de intervalo de confiança. Pesquisas futuras podem ser realizadas com outras aplicações do PARSEC bem como de outras ferramentas de *benchmarking*.

Referências

AMD. *AMD-V Technology for Client Virtualization*. 2018. Disponível em: <https://www.amd.com/en/technologies/virtualization>. Acesso em: 2 dec. 2018. Citado na página 18.

BIENIA, C. et al. The parsec benchmark suite: Characterization and architectural implications. In: *Proc. Int. Conf. Parallel Architectures and Compilation Techniques (PACT)*. [S.l.: s.n.], 2008. p. 72–81. Citado 2 vezes nas páginas 22 e 23.

CABRAL, F. C. B. L. T. C. H. L. M. S. M. L. Virtualização: Uma analise de desempenho das soluções mais utilizadas do mercado. In: *Tecnologus*. [S.l.: s.n.], 2013. Citado na página 13.

CALçADA, J. L. S. C. S. *Física*. [S.l.]: Editora Atual, 2008. Citado na página 43.

CLEUBER. *Introdução à Virtualização*. 2014. Disponível em: <http://www.cleuber.com.br/index.php/2014/08/11/introducao-a-virtualizacao>. Acesso em: 13 nov. 2018. Citado 3 vezes nas páginas 16, 17 e 18.

DEVMEDIA. *Hypervisor: Segurança em ambientes virtualizados*. 2014. Disponível em: <https://www.devmedia.com.br/hypervisor-seguranca-em-ambientes-virtualizados/30993>. Acesso em: 1 dec. 2018. Citado na página 16.

DOCKER. *Why Docker?* 2018. Disponível em: <https://www.docker.com/why-docker>. Acesso em: 07 nov. 2018. Citado na página 20.

EDIVALDOBRITO. *Como instalar o s-tui para monitorar a CPU no terminal*. 2018. Disponível em: <https://www.edivaldobrito.com.br/s-tui-para-monitorar-a-cpu-no-terminal/>. Acesso em: 08 nov. 2018. Citado na página 27.

ELIAS PRACIANO. *Como determinar a memória usada por um processo no Linux, com o utilitário pmap*. 2017. Disponível em: <https://elias.praciano.com/2017/11/como-determinar-a-memoria-usada-por-um-processo-no-linux-com-o-utilitario-pmap/>. Acesso em: 24 nov. 2018. Citado na página 41.

HENNESSY, J. L.; PATTERSON., D. A. *Computer Architecture: A Quantitative Approach*. [S.l.]: Morgan Kaufmann; 3 edition, 2003. Citado na página 22.

HUB.DOCKER. *Docker Hub*. 2016. Disponível em: <https://hub.docker.com/>. Acesso em: 07 nov. 2018. Citado na página 20.

HUGHES, C. J. et al. Physical simulation for animation and visual effects: parallelization and characterization for chip multiprocessors. *SIGARCH Computer Architecture News*, v. 35, p. 220, 2007. ISSN 0163-5964. Citado na página 24.

INTEL. *Intel Virtualization Technology (Intel VT)*. 2018. Disponível em: <https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html>. Acesso em: 2 dec. 2018. Citado na página 18.

INTEL. *Processador Intel® Core™ i5-2410M*. 2018. Disponível em: <https://ark.intel.com/pt-br/products/52224/Intel-Core-i5-2410M-Processor-3M-Cache-up-to-2-90-GHz->. Acesso em: 22 nov. 2018. Citado na página 22.

LIMA, M. N. M. e A. C. P. D. *Noções de Probabilidade e Estatística*. 3 edição. ed. [S.l.]: Editora USP, 2001. Citado na página 31.

LV, Q. et al. Ferret: A toolkit for content-based similarity search of feature-rich data. In: *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*. New York, NY, USA: ACM, 2006. (EuroSys '06), p. 317–330. ISBN 1-59593-322-0. Disponível em: <http://doi.acm.org/10.1145/1217935.1217966>. Citado na página 25.

LV, Q. et al. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In: *Proceedings of the 33rd International Conference on Very Large Data Bases*. VLDB Endowment, 2007. (VLDB '07), p. 950–961. ISBN 978-1-59593-649-3. Disponível em: <http://dl.acm.org/citation.cfm?id=1325851.1325958>. Citado 2 vezes nas páginas 25 e 26.

MATTOS, D. M. F. Virtualização: Vmware e xen. *GTA/POLI/UFRJ*, 2008. Citado 2 vezes nas páginas 16 e 18.

MERKEL, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, Belltown Media, Houston, TX, v. 2014, n. 239, mar. 2014. ISSN 1075-3583. Disponível em: <http://dl.acm.org/citation.cfm?id=2600239.2600241>. Citado na página 19.

MICROSOFT. *Virtualização*. 2018. Disponível em: <https://www.microsoft.com/pt-br/cloud-platform/server-virtualization?ocid=otc-c-br-loc--wikivirt>. Acesso em: 07 nov. 2018. Citado na página 13.

MORABITO, R.; KJäLLMAN, J.; KOMU, M. Hypervisors vs. lightweight virtualization: A performance comparison. In: *Proc. IEEE Int. Conf. Cloud Engineering*. [S.l.: s.n.], 2015. p. 386–393. Citado na página 13.

MOUAT, A. *Using docker*: Developing and deploying software with containers. First edition. Beijing: OReily, 2016. ISBN 9781491915769. Citado 2 vezes nas páginas 19 e 21.

MUNDODOCKER. *O que é Dockerfile*. 2015. Disponível em: <https://www.mundodocker.com.br/o-que-e-dockerfile/>. Acesso em: 09 nov. 2018. Citado na página 51.

OANALISTA. *Monitore a leitura/escrita em disco no Linux com o iotop*. 2016. Disponível em: <https://www.oanalista.com.br/2016/05/20/monitore-a-leituraescrita-em-disco-no-linux-com-o-iotop/>. Acesso em: 22 nov. 2018. Citado na página 41.

PADALA, P. et al. Performance evaluation of virtualization technologies for server consolidation. *HP Labs Tec. Report*, Citeseer, v. 137, 2007. Citado na página 13.

PAHL, C. Containerization and the paas cloud. *IEEE Cloud Computing*, v. 2, n. 3, p. 24–31, maio 2015. ISSN 2325-6095. Citado na página 19.

PARSEC. *Download Parsec 3.0*. 2009. Disponível em: <http://parsec.cs.princeton.edu/download/3.0/parsec-3.0.tar.gz>. Acesso em: 09 nov. 2018. Citado 3 vezes nas páginas 14, 22 e 55.

PARSEC-TUTORIAL. *The PARSEC Benchmark Suite Tutorial - PARSEC 2.0*. 2009. Disponível em: <http://parsec.cs.princeton.edu/download/tutorial/2.0/parsec-2.0-tutorial.pdf>. Acesso em: 08 nov. 2018. Citado 2 vezes nas páginas 25 e 26.

RUBENS, P. *What are containers and why do you need them?* 2015. Disponível em: <http://www.cio.com/article/2924995/enterprisesoftware/what-are-containers-and-why-do-you-need-them.html>. Citado na página 19.

RUBNER, Y.; TOMASI, C.; GUIBAS, L. J. *The earth mover's distance as a metric for image retrieval*. 2000. Citado na página 26.

SANTOS, F. U. dos. O que é uma máquina virtual? *Blog Blue Solutions*, 2014. Disponível em: <https://www.profissionaisti.com.br/2014/07/o-que-e-uma-maquina-virtual/>. Citado na página 16.

SIFAKIS, E.; NEVEROV, I.; FEDKIW, R. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Transactions onGraphics*, v. 24, p. 417, 2005. ISSN 0730-0301. Citado na página 24.

SILVA, F. H. R. e. *Avaliação de Desempenho de Contêineres Docker para Aplicações do Supremo Tribunal Federal*. Tese (Doutorado) — Universidade de Brasília, 2017. Citado 2 vezes nas páginas 13 e 20.

SOFTDESIGN. *O que é Docker? Agilize seu processo com Docker*. 2017. Disponível em: <http://www.softdesign.com.br/blog/agilize-seu-processo-com-docker/>. Acesso em: 12 nov. 2018. Citado na página 20.

TECMUNDO. *Crie uma máquina virtual a partir do sistema operacional presente no computador*. 2018. Disponível em: <https://www.tecmundo.com.br/maquina-virtual/4801-crie-uma-maquina-virtual-a-partir-do-sistema-operacional-presente-no-computador.htm>. Acesso em: 09 nov. 2018. Citado na página 54.

TERAN, J. et al. *Robust quasistatic finite elements and flesh simulation*. 2005. Citado na página 24.

TIFFANYFAY. *Docker 1.11 et plus: Engine is now built on runC and containerd*. 2016. Disponível em: <https://medium.com/@tiffanyfayj/docker-1-11-et-plus-engineis-now-built-on-runc-and-containerd-a6d06d7e80ef>. Acesso em: 07 nov. 2018. Citado na página 21.

TURNBULL, J. *The Docker Book: Containerization is the new virtualization*. [S.l.]: James Turnbull, 2016. Citado na página 21.

UBUNTU. *Alternative downloads*. 2018. Disponível em: <https://www.ubuntu.com/download/alternative-downloads>. Acesso em: 09 nov. 2018. Citado na página 54.

VIRTUALBOX. *Download VirtualBox*. 2018. Disponível em: <https://www.virtualbox.org/wiki/Downloads>. Acesso em: 09 nov. 2018. Citado na página 54.

VIVAOLINUX. *Monitorando processos no Linux com o Htop*. 2010. Disponível em: <https://www.vivaolinux.com.br/artigo/Monitorando-processos-no-Linux-com-o-Htop>. Acesso em: 22 nov. 2018. Citado na página 41.

VMWARE. *Virtualização*. 2018. Disponível em: <https://www.vmware.com/br/solutions/virtualization.html>. Acesso em: 08 nov. 2018. Citado na página 13.

XENPROJECT. *Virtualization*. 2013. Disponível em: <https://www.xenproject.org/users/virtualization.html>. Acesso em: 07 nov. 2018. Citado na página 13.

YU, Y. *OS-level Virtualization and Its Applications*. Tese (Doutorado) — Stony Brook University, 2007. Citado na página 19.

APÊNDICE A – Docker

A.1 Instalação

```
1 sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80
  --recv-keys
2 sudo apt-add-repository 'deb https://apt.dockerproject.org/repo
  ubuntu-xenial main'
3 sudo apt-get update
4 sudo apt-get install -y docker-engine
```

O primeiro passo foi adicionar ao sistema a chave GPG oficial do repositório Docker, em seguida o repositório do Docker às fontes do APT, o terceiro e quarto passos foram para atualizar o banco de dados de pacotes com os pacotes do Docker a partir do novo repositório adicionado e, a instalação do Docker respectivamente.

Com o Docker instalado e funcionando, têm-se em mente que ele consiste em passar uma cadeia de opções seguida de argumentos. A sintaxe assume essa forma:

```
docker [option] [command] [arguments]
```

.

A.2 Dockerfile

Dockerfile é um arquivo de definição onde é possível realizar ou preparar todo o ambiente desejado a partir de um script de execução. Em resumo, o Dockerfile é um arquivo texto com instruções, comandos e passos que seriam executados manualmente [MundoDocker 2015].

O Dockerfile elaborado para o presente trabalho é mostrado a seguir.

```
1 FROM ansible/ubuntu14.04-ansible
2 RUN apt-get update && apt-get install -y gcc && apt-get clean
3 COPY parsec-3.0 /tmp
```

As instruções utilizadas foram:

- *FROM* - Informa a partir de qual imagem será gerada a nova imagem;

- *RUN* - Especifica que o argumento seguinte será executado, ou seja, realiza a execução de um comando;
- *COPY* - Copia arquivos ou diretórios locais para dentro da imagem.

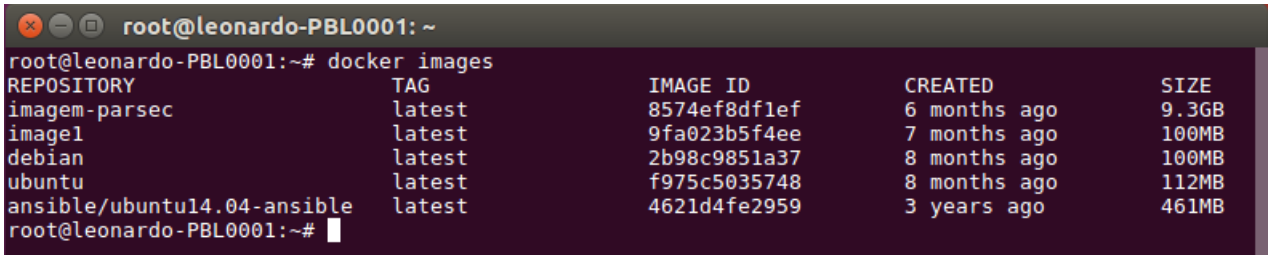
Com o arquivo *Dockerfile* configurado, a imagem será gerada com a execução do comando *build*, mostrado na linha 1 da caixa de texto mostrado abaixo.

```
1 docker build -t imagem-parsec .
2 docker run -it imagem-parsec
```

O comando *build*, é responsável por executar a construção da imagem base nas configurações do *Dockerfile* utilizado. O parâmetro `-t` é usado para informar que a imagem pertence ao usuário que executa o comando. Na sequência deve ser colocado o nome desejado para a imagem - foi utilizado `imagem-parsec` para facilitar sua localização no repositório de imagens. O último parâmetro indica o local onde se encontra o *Dockerfile* - neste caso o diretório corrente.

A Figura 19 mostra a imagem criada, fazendo parte do repositório do Docker.

Figura 19 – Imagem criada a partir do Dockerfile.



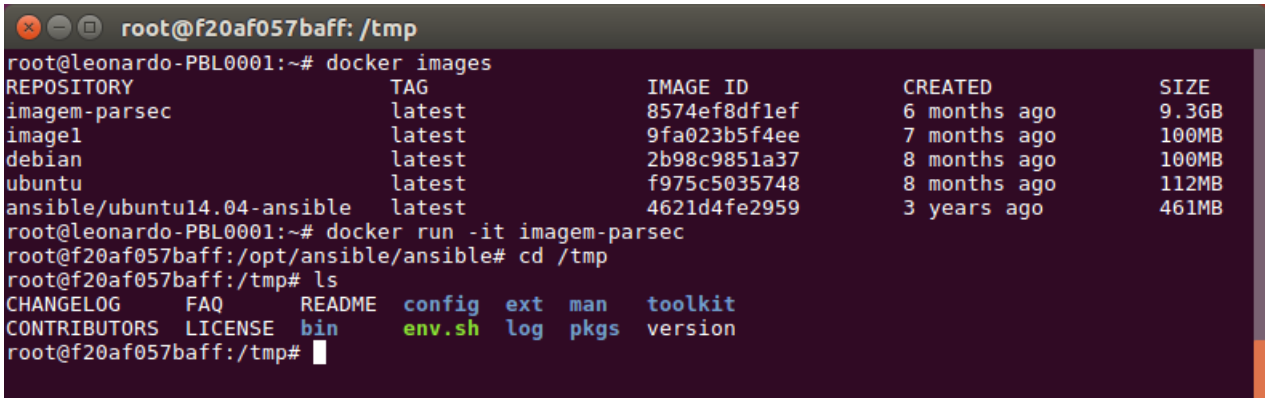
Fonte: Autoria Própria.

A.3 Criação do contêiner

Com a imagem pronta, o próximo passo é instanciar o contêiner. O contêiner é uma instância de uma imagem em execução naquele momento. Para isso o comando utilizado na linha 2 da caixa de texto anterior irá criar um novo contêiner. A opção *run* é para criar o contêiner. O parâmetro `-i` é usado para permitir utilizar o contêiner em modo interativo, e `-t` para que se tenha um terminal de console no contêiner. Em seguida, é especificado o nome da imagem desejada.

A Figura 20 mostra a execução dos comandos para listar as imagens disponíveis, em seguida a criação do contêiner, e o arquivo parsec-3.0 presente no diretório `/tmp`, concluíndo assim a instalação. Para executar os programas dentro do contêiner, basta seguir as mesmas instruções da seção 3.2.3.

Figura 20 – Contêiner criado a partir da imagem.



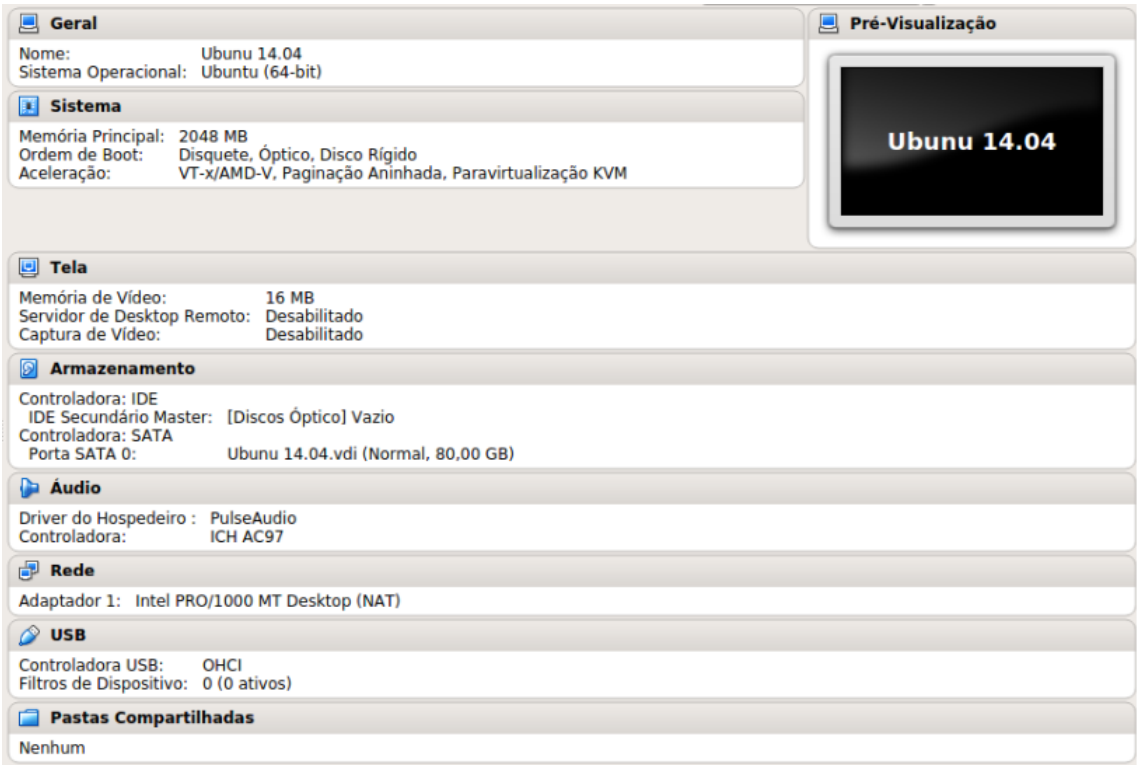
Fonte: Autoria Própria.

APÊNDICE B – Máquina virtual

B.1 Instalação

As máquinas virtuais são o ambiente perfeito para testar programas afinal, se der algum problema ou o aplicativo se revelar um perigoso vírus, nenhum arquivo importante corre risco de ser perdido [Tecmundo 2018]. Para criar um ambiente pronto para ser utilizado por uma máquina virtual a partir dos parâmetros do próprio computador será preciso instalar um *software* para criar e navegar pela máquina virtual, o *software* escolhido foi o VirtualBox. Após fazer o *download* do VirtualBox [VirtualBox 2018], basta ter a imagem `.iso` do sistema operacional desejado para instalar a VM. O sistema operacional escolhido foi o Ubuntu 14.04 (64 bits) [Ubuntu 2018], a Figura 21 exibe as configurações escolhidas para VM instalada.

Figura 21 – Configuração da Máquina Virtual.



Fonte: Autoria Própria.

APÊNDICE C – PARSEC

C.1 Execução dos programas

Para executar qualquer um dos programas do PARSEC, primeiramente foi feito o *download* [PARSEC 2009] do arquivo e inseridos os comandos a seguir:

```
1 tar -xzvf parsec-3.0.tar.gz
2 source env.sh
```

A instrução da primeira linha, é para extrair o arquivo e, a da segunda, para configurar o ambiente PARSEC e preparar as variáveis de ambiente do script *env.sh* que se encontra dentro da pasta extraída.

Para compilar e executar o programa, deve-se ter conhecimento da estrutura dos comandos:

parsecmgmt -a [ACTION] -p [PACKAGE] -c [BUILDCONF] -i [INPUT] -n [THREADS]

- *ACTION* - *build* para compilar e, *run* para executar;
- *PACKAGE* - o pacote desejado, no caso o *facesim*;
- *BUILDCONF* - por *default* gcc, gcc-hooks, gcc-serial;
- *INPUT* - uma das entradas estabelecidas, *test, simdev, simsmall, simmedium, simlarge* ou *native*;
- *THREADS* - número de *threads* para execução do programa.

Os comandos a seguir serviram para compilar e executar os programas, respectivamente.

```
1 parsecmgmt -a build -p [PACKAGE] -c gcc
2 parsecmgmt -a run -p [PACKAGE] -c gcc -i native
```