**CENTRO PAULA SOUZA**
**FACULDADE DE TECNOLOGIA DE MOCOCA**
**CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**Bruno Sebastião Flamini**

# ANÁLISE DE DESEMPENHO ENTRE FERRAMENTAS DE VIRTUALIZAÇÃO

**Mococa-SP**
**Abril/2017**

**BRUNO SEBASTIÃO FLAMINI**

# ANÁLISE DE DESEMPENHO ENTRE FERRAMENTAS DE VIRTUALIZAÇÃO

Trabalho de Conclusão de Curso apresentado à FATEC - Faculdade de Tecnologia de Mococa, para obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas.

Orientador: **Prof. Vinicius Henrique Porto Brisighello**

**Mococa-SP**
**Abril/2017**

**Dedico este trabalho a meu pai, que tanto fez por mim e ainda faz e, sem ele não seria possível realizar mais este objetivo da minha vida.**

## AGRADECIMENTOS

- ✓ **À Deus, por todas as coisas boas que coloca em minha vida todos os dias, e por mais esse objetivo alcançado.**

- ✓ **Ao professor Vinícius pela oportunidade e pela paciência durante a construção deste projeto, agradecendo principalmente a atenção e dedicação a mim e ao meu trabalho.**

- ✓ **Aos meus amigos de faculdade, que sempre me ajudaram nos momentos em que precisei durante todo esse tempo.**

- ✓ **Ao Centro Paula Souza pela oportunidade que me foi dada.**

**“Eu aprendi que todos querem viver no topo da montanha, mas toda felicidade e crescimento ocorrem quando você está escalando-a.”**

**(William Shakespeare)**

## RESUMO

Esta pesquisa teve por finalidade apresentar os resultados do estudo de caso baseado na comparação de desempenho através de um *benchmark* com as mais variadas ferramentas de teste de desempenho entre três das principais ferramentas de virtualização, muito utilizadas tanto em computadores pessoais, como em empresas de pequeno e médio porte, operacionalizadas por usuários de diversos graus de conhecimento, desde usuários leigos até grandes especialistas na área de computação, a fim de gerar um resultado com as principais vantagens, benefícios, pontos fortes e pontos fracos de uma ferramenta em relação às demais. O estudo de caso constituíra de um desktop com as ferramentas VirtualBox, Qemu-KVM e VMware Workstation Player instaladas para a execução dos testes. Também será abordado como a virtualização de computadores funciona e como esta se comporta depois de configurada, especificando-se os processos realizados para que uma máquina virtual possa ser criada e executada corretamente dentro da complexa arquitetura de um computador. Por fim, serão discutidos os resultados dos testes em que o VirtualBox obteve o melhor desempenho.



**Palavras-chave**: Benchmarking. Desempenho. Hypervisor. Máquina Virtual. Virtualização.

## ABSTRACT

This project aims to present the results of the case study based on the comparison of performance through a benchmark with the most varied performance testing tools among three of the main virtualization tools, much used in both personal computers and small and medium-sized enterprises, operated by users of different degrees of knowledge, from lay users to major specialists in the field of computing, in order to generate a result with the main advantages, benefits, strengths and weaknesses of one tool in relation to the others. The case study shall consist of a desktop with the tools VirtualBox, Qemu-KVM and VMware Workstation Player installed for the execution of the tests. The functioning of computer virtualization and how it behaves after it is configured will also be shown by specifying the processes performed so that a virtual machine can be created and executed correctly within the complex architecture of a computer. Finally, the results of the tests in which VirtualBox has obtained the best performance will be discussed.



**Keywords**: Benchmarking. Hypervisor. Performance. Virtual Machine. Virtualization.

## LISTA DE FIGURAS

FIGURA PÁGINA

## LISTA DE QUADROS

QUADROS PÁGINA

## LISTA DE ABREVIATURAS E SIGLAS

*AMD – ADVANCED MICRO DEVICES*
*API – APPLICATION PROGRAMING INTERFACE*
*ATA – ADVANCED TECHNOLOGY ATTACHMENT*
*BIOS – BASIC INPUT OUTPUT SYSTEM*
*BSD – BERKELEY SOFTWARE DISTRIBUTION*
*CPU – CENTRAL PROCESSING UNIT*
E/S – ENTRADA/SAÍDA
EX – EXEMPLO
*FSF – FREE SOFTWARE FOUNDATION*
*GB – GYGABYTE*
*GCC – GNU C COMPILER*
*GNU – GNU'S NOT UNIX*
*GPL – GENERAL PUBLIC LICENSE*
*HD – HARD DISK*
*HTTP – HYPER TEXT TRANSFER PROTOCOL*
*IBM – INTERNATIONAL BUSINESS MACHINE*
*ISA – INSTRUCTION SET ARCHITECTURE*
*JIT – JUST-IN-TIME*
*KB – KILOBYTE*
*KVM – KERNEL-BASED VIRTUAL MACHINE*
*MB*/S – *MEGABYTES* POR SEGUNDO
*MACOS - MACINTOSH OPERATING SYSTEM*
*MIT – MASSACHUSETTS INSTITUTE OF TECHNOLOGY*
*PC – PERSONAL COMPUTER*
*PTS – PHORONIX TEST SUITE*
*RAM – RANDOM ACCESS MEMORY*
*SCSI – SMALL COMPUTER SYSTEM INTERFACE*
SO – SISTEMA OPERACIONAL
*SSL – SECURITY SOCKET LAYER*
*TLS – TRANSPORT LAYER SECURITY*
TPS – TRANSAÇÃO POR SEGUNDO

*VGA – VIDEO GRAPHICS ARRAY*

*VM –VIRTUAL MACHINE*

*VMM – VIRTUAL MACHINE MONITOR*

**SUMÁRIO**

# 1 INTRODUÇÃO

Desde o seu surgimento, em 1930, o computador eletrônico se desenvolveu de forma rápida, podendo ser visto no século XXI através de supermáquinas, de todos os tipos existentes, com um alto poder de processamento, economizando espaço físico e energia elétrica. Com isso, foi proporcionada uma grande expansão do uso de computadores pessoais[1], tanto para pequenas empresas quanto para usuários domésticos, que podem usufruir de suas vantagens, permitindo utilização de recentes sistemas operacionais (SOs) e seus serviços, sem comprometer o desempenho das máquinas.

Mesmo possuindo computadores de última geração, os usuários encontram alguns problemas relacionados aos SOs, como a execução de apenas um sistema operacional por vez em um determinado *hardware*, ou seja, em uma máquina pode haver mais de um sistema operacional instalado, porém os sistemas não podem ser executados simultaneamente. Outro problema é a incompatibilidade entre *hardware* e *software*, na qual um *hardware* passa a ser atualizado com mais frequência e mais rápido em relação aos programas que nele devem ser executados, impossibilitando a utilização ao máximo do *hardware* em que está instalado. Além do mais, alguns SOs não são compatíveis com determinados *hardwares*, o que limita a experiência de usuários com novos tipos de sistemas.

[1] Em inglês, utiliza-se a sigla PC (*Personal Computer)*.

Como o poder de processamento evoluiu, vários SOs podem ser executados simultaneamente em um único computador, através do procedimento denominado Virtualização.

Já o processo de abstrair um *hardware* convencional, dedicando total ou parcialmente suas funcionalidades a um *software*, é chamado de Máquina Virtual (em inglês *virtual machine*, ou simplesmente VM), isto é, um *software* que representa as funcionalidades de um *hardware* (BARHAM, 2003).

VM permite aos usuários testar diversos sistemas operacionais, instalar em suas máquinas e utilizá-los, a fim de descobrir suas utilidades, ferramentas, capacidade, interface, entre outros, para se ambientar com o sistema e descobrir qual melhor o atende, de acordo com suas necessidades.

A Virtualização consegue diminuir as limitações impostas pelos sistemas operacionais aos usuários, pois permite a interação com diversos tipos de SOs e *softwares*, utilizando apenas uma máquina física, eliminando quaisquer restrições impostas. Através da virtualização, esses problemas são eliminados e os usuários podem executar qualquer sistema operacional em sua máquina, como por exemplo, um usuário de SO GNU/Linux que deseja fazer a utilização de um *software* existente apenas para o sistema operacional Windows apenas uma vez, não tem a necessidade de instalar o sistema na sua máquina para utilizá-lo, pois o usuário simplesmente pode virtualizar, e quando não mais precisar, simplesmente excluir o sistema da máquina. Um bom exemplo é o famoso sistema operacional da Apple, o Macintosh Operating System (macOS), que vem instalado em um equipamento de alto custo, pode ser emulado a partir de outras plataformas por curiosidade, a fim de conhecer suas funcionalidades, sua interface e suas ferramentas, permitindo assim, aos usuários de outros sistemas terem a mesma experiência de um usuário que possui o equipamento original.

A virtualização provê benefícios em ambientes empresariais e domésticos. Pode-se destacar que é possível utilizar programas desenvolvidos para outros sistemas operacionais sem a necessidade de reiniciar a máquina para a troca de sistema, no qual é possível executar sistemas antigos em dispositivos modernos. Outro ponto importante é a redução de custos, como eletricidade e *hardware*, pois uma máquina pode desempenhar o papel de várias outras máquinas convencionais.

As máquinas virtuais são consideradas ferramentas primordiais para testes e análises de sistema que estão em fase de desenvolvimento, na qual, em casos de falhas ou perdas de dados, o sistema pode ser recuperado sem a necessidade de reconfiguração. Não menos importante, pode ser utilizado na área acadêmica para fins de estudos, em situações que máquinas corrompidas com sérios danos podem simplesmente ser descartadas e substituídas por máquinas novas (SIQUEIRA, 2010).

Com o passar dos anos, várias ferramentas de virtualização foram sendo desenvolvidas em paralelo com a evolução dos computadores. Com isso, supermáquinas foram sendo criadas e cada vez mais *softwares* de virtualização foram sendo desenvolvidos e aperfeiçoados.

Por isso, pode-se encontrar vários *softwares* que oferecem serviços de virtualização. Com tantas opções, os usuários podem optar pela ferramenta que melhor lhe convém. Tanto usuários domésticos quanto pequenas empresas exigem cada vez mais um *software* com desempenho satisfatório, que lhe atenderá de forma rápida, eficiente e segura.

Pensando em todo esse desenvolvimento ocorrido nos últimos tempos, esse projeto abordará algumas das principais ferramentas de virtualização utilizadas por usuários domésticos e pequenas empresas, especificando-as detalhadamente, sobre suas principais características e modos de uso. Será realizada uma abordagem completa de como funciona a virtualização, além de comparar as máquinas virtuais através de testes de desempenho a partir de *benchmark* (ferramentas de testes), a fim de gerar um resultado com as principais vantagens, benefícios, pontos fortes e pontos fracos de uma ferramenta em relação às demais.

# 2 REFERENCIAL TEÓRICO

Neste capítulo, serão abordados conceitos de arquitetura computacional e sistemas operacionais, conceitos estes diretamente ligados ao entendimento de Virtualização, para entender seu funcionamento e sua estrutura.

## 2.1 Virtualização

Máquina Virtual é um conceito antigo que vem sendo utilizado desde os primórdios da computação. Começou no Massachusetts Institute of Technology (MIT) nos anos 50, com o intuito de desenvolver sistemas operacionais de tempo compartilhado (*time-sharing*: utilização do mesmo computador por vários usuários simultaneamente). A primeira máquina virtual foi desenvolvida pela International Business Machine (IBM), na década de 60, chamada CP-67, com o intuito de resolver o problema que surgiu junto com o desenvolvimento dos sistemas *time-sharing*, o compartilhamento de um único computador com aplicações suscetíveis a falhas. Muitas soluções foram encontradas, como por exemplo, o uso de vários computadores para aumentar o desempenho e garantir o isolamento das aplicações, mas todas essas soluções exigiam um alto desperdício de recursos e dinheiro, pois as máquinas ficavam a maior parte do tempo ociosas. Depois do grande sucesso do CP-67, a IBM lançou o VM/370, um VMM (Virtual Machine Monitor) para computadores com arquiteturas que permitiam a realização da virtualização através

de uma arquitetura chamada /370, sendo esses os primeiros registros de tentativas de virtualização conhecidos (MATTOS, s.d.).

Um exemplo do poder que as ferramentas de virtualização possuem é a tecnologia de multiprocessamento de convidado, na qual cada máquina virtual pode possuir até 32 processadores virtuais, independentemente do processador do *host system* (sistema anfitrião).

A virtualização está presente em todas as plataformas desde 2001, inclusive a x86 (32 *bits*), isto é, a virtualização pode ser realizada em computadores mais simples de configuração ou até mesmo para equipamentos antigos. As vantagens oferecidas pela criação de ambientes virtuais são facilmente identificadas, já que para hospedar uma máquina virtual não é preciso um *hardware* pronto para isto, sendo necessária a instalação de um *software* na máquina física (no *hardware*) chamado Hypervisor, cuja função é emular o *hardware*, permitindo a instalação do SO convidado (CACIATO, 2015).

Em um futuro próximo, as máquinas virtuais chegarão ao seu extremo em questões de desempenho e segurança, com todo o crescimento dessa área nos últimos anos, envolvendo estudos e pesquisas de grandes indústrias e universidades pelo mundo todo. Empresas como Intel e AMD (Advanced Micro Devices) estão investindo pesado em termos de dinheiro e pesquisas, para que a virtualização na plataforma x86 seja aperfeiçoada, permitindo o desenvolvimento de novas tecnologias de virtualização. Outro ponto importante desse aperfeiçoamento são os dispositivos de Entrada e Saída (E/S), cujo objetivo é projetar dispositivos de alto desempenho, capazes de permitir o acesso simultâneo por vários sistemas virtuais (MASSALINO, 2012).

### *2.1.1 Isomorfismo*

O computador é um sistema complexo, composto de *hardware* e *software*. Logo, é necessário enxergar os componentes de forma separada, como sendo um subsistema que oferece serviços de forma independente entre si.

Segundo Carissimi (2009, p.40), “o princípio básico é o de dividir para conquistar, o que significa dividir o sistema em várias camadas funcionais hierárquicas, cada uma com um nível de abstração apropriado”, ou seja, uma

camada pode fornecer serviço à uma outra camada sem enxergar como são os detalhes internos de funcionamento dessa outra camada.

Porém, toda essa organização gera uma dependência entre as camadas, na qual é necessário respeitar a interface para poder utilizar os seus serviços. Para acabar com o problema de dependência gerado pela divisão em camadas, mapeia-se uma interface para outra com a introdução de uma camada intermediária de adaptação (CARISSIMI, 2009).

Segundo Popek e Goldberg (1974 apud Carissimi, 2009, p.41): “esse mapeamento é a base da virtualização e foi introduzido através do conceito de isomorfismo”.

Assim, pode-se definir isomorfismo como sendo a transformação do estado de um sistema X em um estado homólogo em outro sistema Y, onde, uma sequência de operações que modifica o estado do sistema X para outro estado, também executará de forma equivalente no sistema Y. Ou seja, isomorfismo pode ser definido como a abstração de um sistema de forma isolada, onde as instruções executadas em um sistema refletirão diretamente no outro (CARISSIMI, 2009).

### *2.1.2 Multiplexação*

A multiplexação, diretamente ligado ao entendimento de virtualização e fundamental durante a criação de máquinas virtuais, pode ser definida como o compartilhamento de recursos gerenciados pelo processador. Conforme Tanenbaum (2009), o compartilhamento é subdividido de duas maneiras:

- Compartilhamento no tempo (*time sharing*): compartilhamento de um recurso entre vários usuários para a execução de uma ou diferentes tarefas. Tem por objetivo determinar a ordem dos usuários que utilizarão o recurso e o tempo que este será utilizado até o recurso ser cedido ao próximo usuário. Exemplo (ex.): impressora (decidir qual saída será a próxima da fila para ser impressa diante de múltiplas saídas).

- Compartilhamento no espaço: tem por objetivo dividir o recurso desejado para que todos os usuários que pretendem utilizá-lo possam usá-lo ao mesmo tempo, sem a necessidade de espera. Ex.: HD (*hard disk*), pois vários arquivos de usuários podem estar salvos no mesmo disco rígido ao mesmo tempo).

## 2.2 Abstração

Massalino (2012) define a virtualização como a abstração dos recursos de *hardware* oferecidos por um computador em vários ambientes de execução, aplicada através de tecnologias como tempo compartilhado, simulação de máquina e emulação.

A virtualização é capaz de fazer com que uma máquina física se comporte como um ou mais computadores, em que a arquitetura do computador físico é acessada por todos os computadores virtualizados. Os computadores virtualizados podem trabalhar de formas separadas ou interagir uns com os outros.

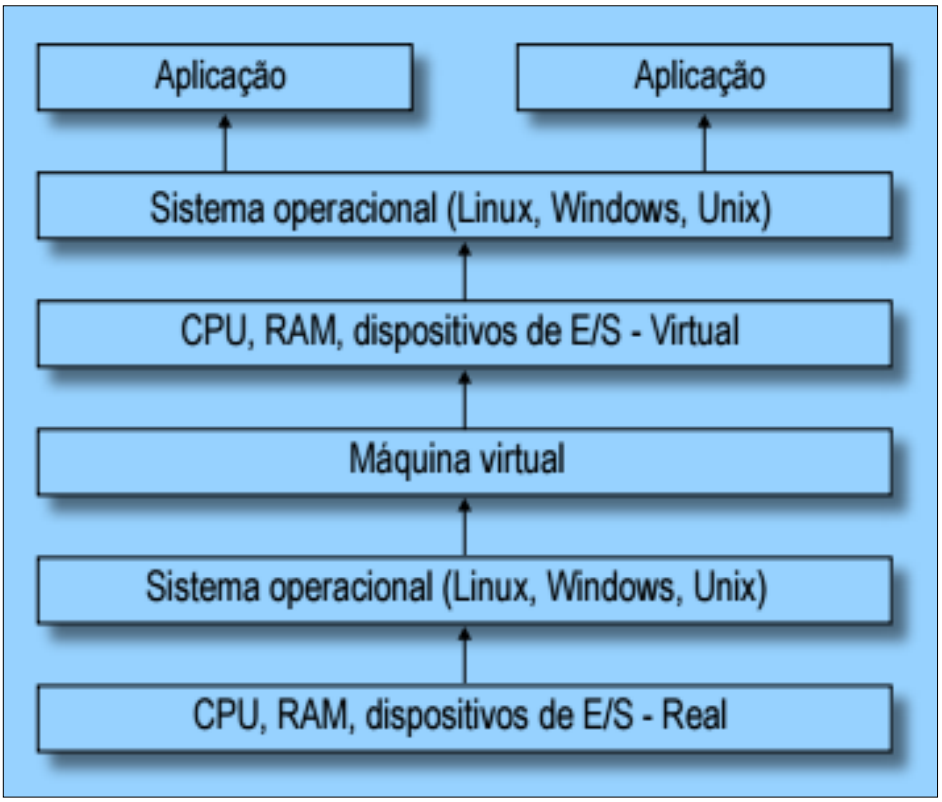
A camada de abstração é a responsável por reproduzir todas as características do *hardware* físico no ambiente virtualizado (MASSALINO, 2012).

Virtualização se disponibiliza também para arquiteturas x86, e existem quatro principais formas de abstração (virtualização): virtualização do *hardware*, virtualização do sistema operacional, virtualização do ISA (Instruction Set Architecture) e virtualização de linguagens de programação.

### *2.2.1 Virtualização do Hardware*

Virtualização de *hardware* foi o primeiro modelo de virtualização criado, desenvolvido pela IBM para a primeira geração de computadores com suporte à arquitetura de virtualização, o /370. Essa tecnologia é ainda utilizada, como por exemplo, o VMware x86 (LAUREANO, 2006). A virtualização do *hardware* é uma tecnologia capaz de dividir todos os recursos de *hardware* em múltiplas partições (*hard partitions*), onde cada partição pode manter uma instância do sistema operacional, conforme a Figura 1.

Figura 1 – Esquema da Virtualização do Hardware.



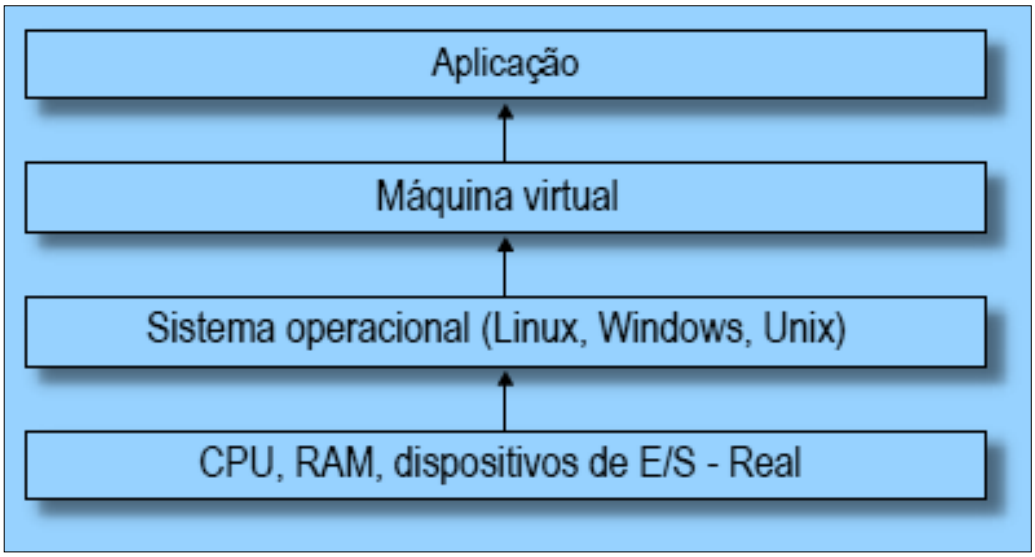

**Fonte:** Laureano, 2006.

A virtualização do *hardware* pode prover várias vantagens, como a maior efetividade no gerenciamento de recursos das múltiplas partições, já que gerenciamento desses recursos é realizado através de componentes de *hardwar*e separados. Outra característica importante é o isolamento elétrico, onde uma partição afetada por uma falha não interfere em outra partição (MASSALINO, 2012).

### *2.2.2 Virtualização de Linguagens de Programação*

A Virtualização de Linguagens de Programação cria uma aplicação no topo do sistema operacional para que a virtualização exporte uma abstração para a execução de programas desenvolvidos em linguagens de programação de alto nível, conforme retratado na Figura 2. Esse tipo de virtualização é desenvolvido apenas para projetos de fins específicos, como por exemplo, a plataforma Java (LAUREANO, 2006).

Figura 2 – Esquema da Virtualização de Linguagens de Programação.




**Fonte**: Laureano, 2006.

### *2.2.3 Abstração do ISA*

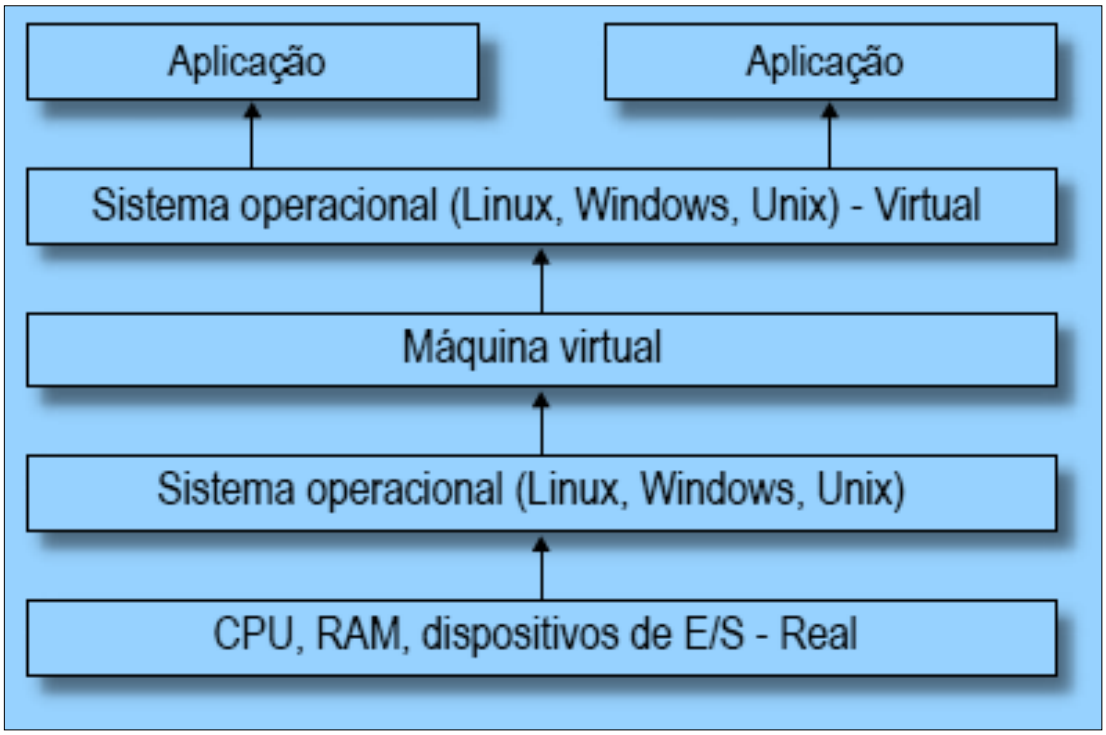
A Abstração do ISA é feita através da emulação completa de todos os conjuntos de instrução da máquina. Nesse caso, a máquina virtual será emulada através da execução das instruções do sistema operacional convidado gerando a tradução das instruções para o sistema nativo (MASSALINO, 2012).

### *2.2.4 Virtualização do Sistema Operacional*

A Virtualização do Sistema Operacional é definida apenas pela abstração do sistema operacional: a máquina física é dividida em múltiplas partições, chamadas de máquinas virtuais, na qual cada partição possui seu próprio sistema operacional de forma isolada[2]. Segundo Massalino (2012), o gerenciamento dos recursos e o isolamento de cada partição são feitos por uma camada de *software* denominada Hypervisor ou VMM (Virtual Machine Monitor), conforme a Figura 3.

[2] **Isolamento**: pode ser definido como a separação das máquinas virtuais que estão em execução da máquina física. Uma máquina virtual não interfere em outra máquina, que também não pode interferir no VMM.

Figura 3 – Esquema da Virtualização do Sistema Operacional.




**Fonte**: Laureano, 2006.

## 2.3 Máquinas Virtuais

O termo máquina virtual surgiu nos anos 60, com o surgimento dos computadores com suporte a arquitetura /370 da IBM. Pode-se definir o termo máquina virtual (VM) como sendo uma cópia perfeita e isolada de uma máquina física, construída a partir de um Hypervisor (LAUREANO, 2006).

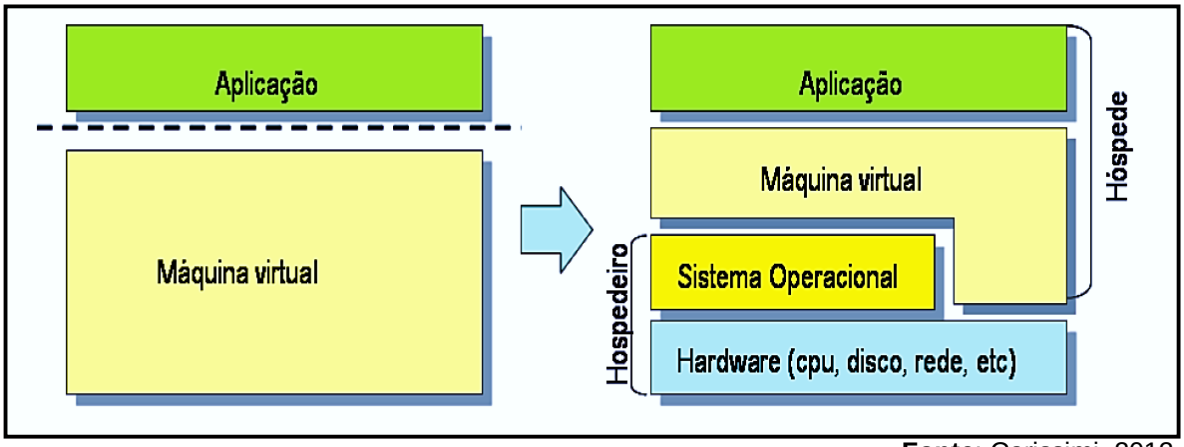
Conforme Carissimi (2012), para conceituar máquina virtual, é preciso dar ênfase a dois conceitos diferentes dentro de um sistema virtualizado, o conceito de processo (máquina virtual de processo) e núcleo do sistema operacional (Hypervisor).

### *2.3.1 Máquina Virtual de Processo*

Máquina Virtual de Processo, também conhecida como *Runtime*, consiste na virtualização de uma aplicação, e se tem a impressão de estar sendo executada sozinha. A camada de abstração é feita pelo conjunto de instruções de máquina (ISA) e pelas chamadas de sistema (*system calls*). A aplicação roda sobre um

programa que executa a máquina virtual de processo, conforme a Figura 4, que é uma entidade temporária, ou seja, só existirá enquanto a aplicação estiver em execução.

Figura 4 – Máquina Virtual de Processo.




**Fonte**: Carissimi, 2012.

Esse tipo de máquina virtual possui um interpretador próprio, pois, muitas vezes, uma aplicação virtualizada possui um código binário diferente do processador no qual a máquina virtual está sendo executada. O interpretador realiza uma procura pelas instruções, decodificações e emulação da instrução necessária para a execução da aplicação.

Outro recurso utilizado é a tradução binária dinâmica, também chamada de Compilação *Just-In-Time* (JIT), cuja função é traduzir os blocos de instruções das aplicações hóspedes em blocos de instruções para o sistema *host*, gerando uma otimização em relação ao interpretador de instrução por instrução (CARISSIMI, 2012).

### *2.3.2 Máquina Virtual de Sistema (Hypervisor)*

O conceito de núcleo do sistema operacional, segundo Carissimi (2012), consiste na definição de máquina virtual como sendo um ambiente completo para execução de vários processos, que podem ser de diferentes usuários. Nesse tipo de máquina, o responsável por gerenciar os recursos de *hardware* utilizados é o núcleo do sistema operacional.

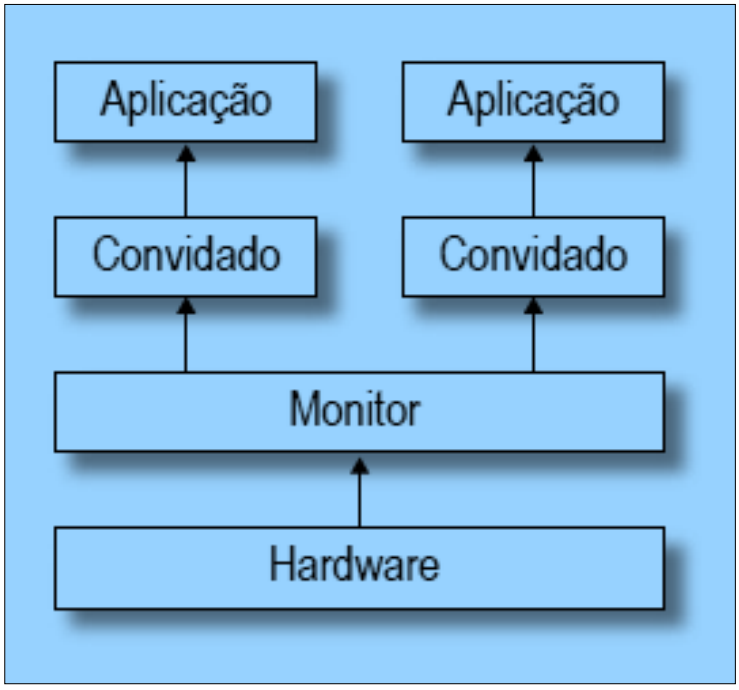
Nesse caso, a abstração também é feita pelo conjunto de instruções de *hardware* (ISA), porém difere da máquina virtual de processo, pois o núcleo do sistema operacional permanece ativo enquanto o computador estiver ligado, e a máquina virtual de processo é temporária, existindo apenas enquanto o processo está em execução.

Vários sistemas operacionais podem ser executados ao mesmo tempo, cada um em sua própria máquina virtual. Há dois tipos de máquina virtual de sistema: Hypervisor tipo I e Hypervisor tipo II (CARISSIMI, 2012).

### ***2.3.2.1* Hypervisor *Tipo I***

Também chamado de nativo, é o responsável por controlar todo o *hardware* da máquina física e das máquinas virtuais no qual ele executa. A sua principal função é compartilhar os serviços de *hardware* entre todas as máquinas virtuais, de modo com que cada máquina haja como se os serviços oferecidos pelo *hardware* fossem dedicados somente à ela, conforme a Figura 5, fazendo com que ela se comporte como uma máquina física completa (CARISSIMI, 2012).

Figura 5 – Hypervisor (monitor) do tipo I.

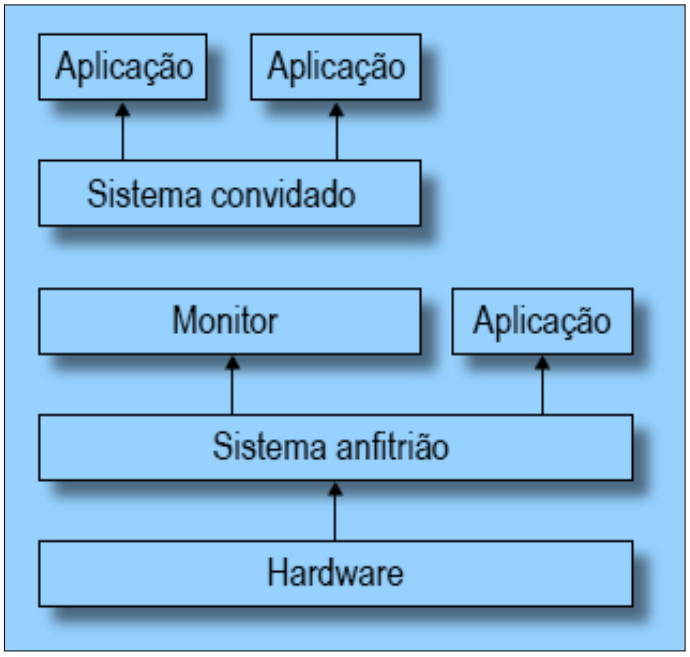



**Fonte**: Laureano, 2006.

### *2.3.2.2* Hypervisor *Tipo II*

Conhecido como Hóspede, sua característica é a execução sobre o sistema operacional anfitrião da máquina, como se fosse um processo em sistema real, conforme a Figura 6. O que o Hypervisor faz é oferecer uma camada de virtualização onde está o sistema operacional hóspede e uma outra camada que virtualiza o *hardware* e todos os seus recursos oferecidos (CARISSIMI, 2012).

Figura 6 – Hypervisor (monitor) tipo II.




**Fonte**: Laureano, 2006.

### *2.3.2.3* Hypervisors *Híbridos*

Além dos 2 tipos relacionados acima, têm-se o Hypervisors Híbridos, que são arquiteturas diferentes dos modelos tipo I e II, pois é uma forma otimizada desses outros dois tipos, já que na maioria das vezes os tipos I e II são pouco utilizados durante as implementações em suas formais originais. Durante as implementações, muitas modificações são feitas, a fim de melhorar o desempenho das aplicações nos sistemas virtualizados. Conforme Laureano (2006), existem quatro tipos de otimização muito utilizados atualmente:

1°- o sistema virtualizado tem permissão para acessar diretamente o *hardware* da máquina física. É feita através da implementação de modificações no núcleo do sistema virtualizado e do Hypervisor.

2°- o *host system* pode ser acessado pelo sistema virtualizado. É feita através da implementação da API do host ao sistema virtualizado.

3°- o sistema virtualizado acessa diretamente o *hardware* do *host* que é implementado através de um driver de dispositivo específico gerenciado pelo Hypervisor e o sistema *host* simultaneamente.

4°- o Hypervisor tem acesso direto ao *hardware*, criando uma interface de baixo nível para acesso ao *hardware*.

### *2.3.3 Propriedade dos* Hypervisors

De acordo com Laureano (2006), elas estão divididas da seguinte forma:

- Isolamento: garante que um sistema em execução em uma máquina virtual não interfira em outro *software* de outra máquina virtual. É uma medida de segurança adotada para que o erro de um *software* não interfira em outro *software*. Outro ponto importante do isolamento é a gerência de recursos, onde o desempenho de uma máquina não afeta a outra.
- Inspeção: o Hypervisor controla todas as informações de estado da máquina. Ex.: CPU, memória RAM, memória *cache*.
- Interposição: o Hypervisor pode controlar as instruções da máquina virtual.
- Eficiência: execução de algumas instruções diretamente no *hardware*.
- Gerenciabilidade: centralização das instâncias das máquinas.
- Compatibilidade de *software*: garantir a compatibilidade de modo que o software escrito para a máquina funcione.
- Encapsulamento: utiliza a camada de virtualização para controlar a execução de *softwares* na máquina virtual.

## 2.4 Técnicas de Virtualização e Emulação

Esta seção abordará as mais utilizadas e modernas técnicas de virtualização existentes: virtualização total, paravirtualização e tradução dinâmica. Além do mais,

especificará o funcionamento das tecnologias para realizar a complexa tarefa de criação dos Hypervisors. O principal quesito é o cuidado de como os recursos de *hardware* da máquina serão compartilhados entre o sistema nativo e o sistema hóspede, de modo que um não interfira no outro.

O componente principal, de maior importância, é o processador, já que a questão chave é o que fazer quando um sistema hóspede realiza uma função privilegiada, exclusiva do sistema nativo.

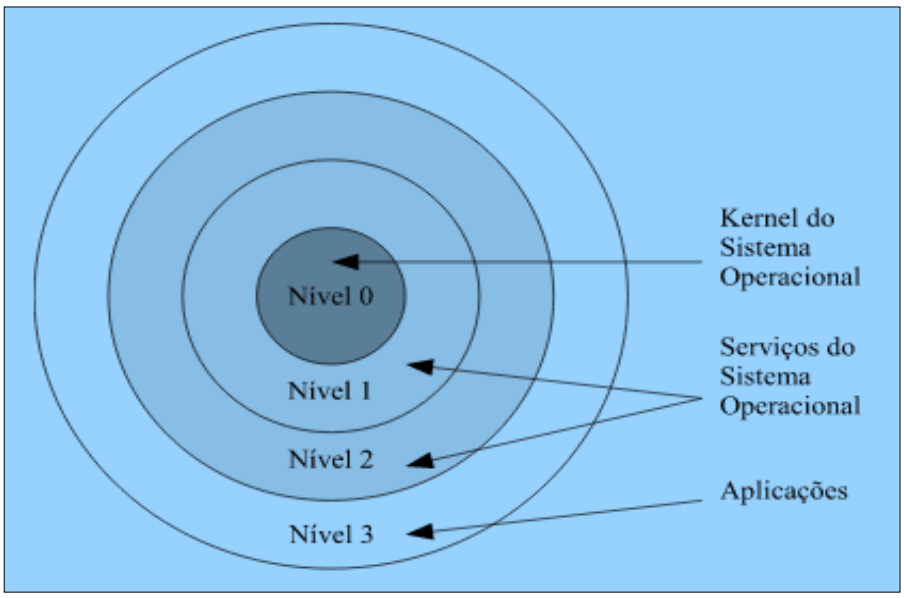
Para entender como esse processo funciona, é preciso conhecer a arquitetura x86, pois a criação de uma máquina virtual não é uma tarefa das mais simples.

O processador, no geral, possui um conjunto de instruções que é classificado em três grupos:

- Privilegiadas: geram exceções quando executadas em modo usuário por um programa.
- Sensíveis de controle: permite que recursos de sistema sejam alterados.
- Sensíveis comportamentais: seu comportamento é gerado através das configurações de recursos.

Os processadores com arquitetura x86 possuem quatro modos de operação, chamados de anéis de proteção, identificados de 0 a 3, conforme a Figura 7.

Figura 7 – Anéis de proteção do processador x86.




**Fonte**: Castro, 2006.

Nos sistemas operacionais tradicionais para essa arquitetura como Windows, Linux e Unix, possuem apenas dois modos: 0 e 3, onde o modo 0 possui um nível mais alto de privilégio de execução, que são os privilégios dados ao SO, e o modo 3 possuí o menor nível de privilégio de execução, que são os privilégios dados ao modo usuário. Nos processadores, em geral, se um processo de usuário realiza uma instrução privilegiada, essa instrução deve ser tratada de modo adequado.

Porém, há uma exceção na arquitetura x86, onde esta possui mais dezessete instruções sensíveis não privilegiadas, ou seja, processos de usuários podem ser executados sem ser tratados, sem gerar exceções, o que gera vários problemas no momento de se criar uma máquina virtual, sendo a grande perda de desempenho o principal problema gerado. Para que a virtualização possa ocorrer na arquitetura x86, foram desenvolvidas as técnicas de virtualização, que são técnicas desenvolvidas para tratar de forma adequada todas as instruções dos processadores com arquitetura x86 (CARISSIMI, 2012).

### *2.4.1 Virtualização Total*

Esta técnica, como o próprio nome sugere, é responsável pela virtualização total da camada de *hardware* do sistema hospedeiro, incluindo as instruções do processador. Assim, o sistema hóspede pode ter uma interface de *hardware* igual à do sistema hospedeiro. Para que isso seja possível, cada uma das instruções executadas pelo sistema hóspede devem ser traduzidas em instruções equivalentes no sistema hospedeiro (MAZIERO, 2013).

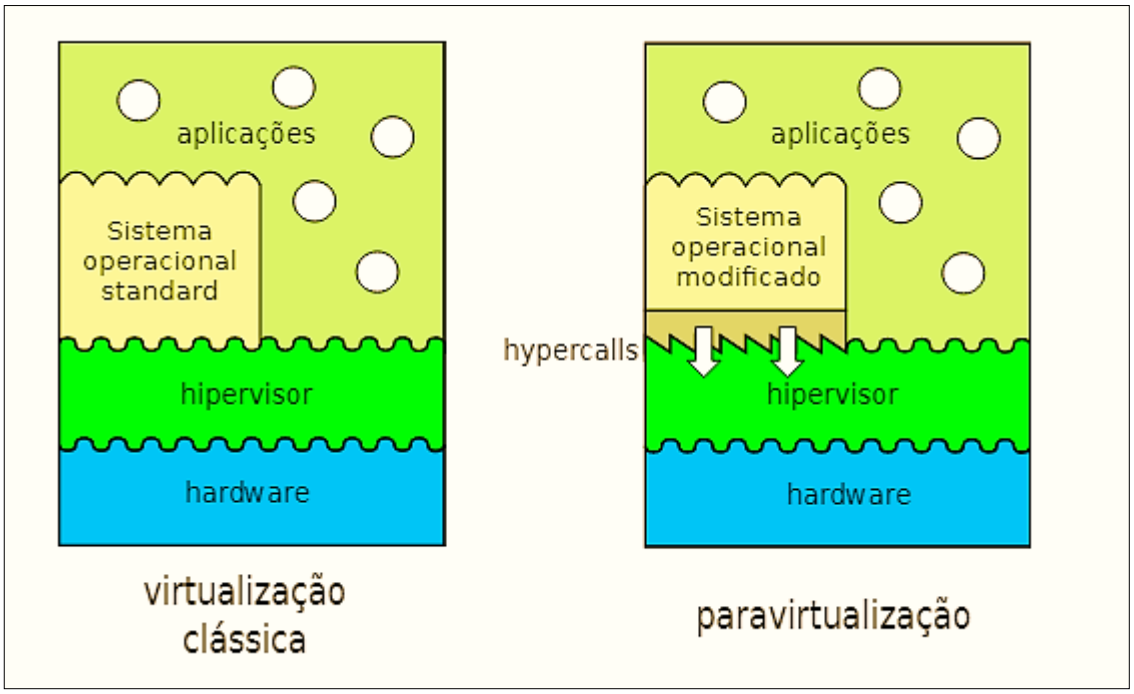
Mas para isso, o sistema virtualizado passa a ser executado de forma mais lenta, pois o Hypervisor deve efetuar alguns procedimentos para determinar quais instruções são sensíveis ou não, o que acaba gerando uma grande perda de desempenho. Mas, com isso, é possível executar sistemas operacionais em plataformas diferentes daquela para a qual foi desenvolvida (LAUREANO, 2006).

### *2.4.2 Paravirtualização*

Pode ser considerada um ‘aprimoramento’ da virtualização total. Como a arquitetura x86 é complexa para a realização da virtualização por causa de suas instruções sensíveis, na paravirtualização, essas instruções são tratadas pelo Hypervisor diretamente na memória.

Para isso, a interface entre o Hypervisor e o sistema hóspede é modificada de forma que o *hardware* virtualizado seja semelhante, mas não igual ao *hardware* real. Com isso, o Hypervisor e o sistema hóspede são acoplados de uma melhor forma, conforme a Figura 8, gerando um ganho de desempenho significativo. Para que isso ocorra, cada sistema convidado recebe uma API do Hypervisor chamada de *hypercall* que é utilizada toda vez que a máquina virtual precisa acessar a interface de *hardware* virtual (MAZIERO, 2013).

Figura 8 – Comparação entre Virtualização Total e Paravirtualização.




**Fonte**: Maziero, 2013.

### *2.4.3 Tradução Dinâmica*

A tradução dinâmica é a compilação dos códigos da máquina hóspede. A máquina hóspede emite instruções que são analisadas e traduzidas pelo Hypervisor à medida que a máquina virtual executa seus processos.

O objetivo da tradução dinâmica é adequar as instruções geradas pela máquina virtual à interface ISA do sistema real e também tratar das instruções sensíveis de baixo nível de privilégio. Tudo isso, otimizando as sequências de instruções geradas pela máquina virtual a fim de ganhar desempenho (MAZIERO, 2013).

De acordo com Ung e Cifuentes (2006, apud Maziero, 2013, p.319), a tradução dinâmica é composta pelos elementos descritos no Quadro 1.

Quadro 1 – Sequência de instruções da Tradução Dinâmica.

| | |
|---|---|
| Desmontagem | A decomposição em blocos de instrução do fluxo de *bits* de código da *guest system*. |
| Geração de código intermediário | Representação independente de máquina para cada bloco de instrução. |
| Otimização | Análise dos blocos para aplicação de eventuais otimizações |
| Codificação | Tradução do bloco de instruções da máquina virtual para a máquina física. |
| *Caching* | Blocos com execução frequentes são armazenados em *cache* para evitar que sejam recompilados novamente. |
| Execução | O bloco de instruções compilado é executado no próprio processador da máquina real. |

**Fonte**: Adaptado pelo autor, 2017.

## 2.5 Ferramentas de Virtualização

O objetivo desse capítulo é abordar as ferramentas que serão utilizadas no *benchmark* deste projeto, a fim de descobrir qual possui o melhor desempenho durante a execução das máquinas virtuais. As ferramentas utilizadas serão o VirtualBox, VMware Workstation Player e o Qemu-KVM. Essas ferramentas foram escolhidas porque são as mais utilizadas no dia a dia de pequenas empresas e usuários domésticos como solução para uso e aplicação de máquinas virtuais. Será

especificado detalhadamente o que são e como funcionam essas ferramentas, as suas formas de aplicação e uso.

### *2.5.1 VirtualBox*

O VirtualBox é um *software open source* sob a licença GNU/GPL, podendo ser copiado, estudado, modificado e usado sem restrições, na qual o código-fonte é de livre acesso para todos. Foi desenvolvido em 2007, na Alemanha, por uma empresa chamada Innotek GmBH. Em 2009, o VirtualBox foi adquirido pela Oracle através da aquisição da SunMicrosystems, que na época era a atual proprietária do VirtualBox.

O VirtualBox é um *software* Hypervisor do tipo II que utiliza a técnica de virtualização total, que virtualiza os principais componentes de *hardware* para sua execução. Os códigos são executados diretamente no processador, e para que as instruções privilegiadas sejam executadas, o VirtualBox move o sistema do nível 0 para o nível 1. O recompilador do VirtualBox, baseado no Qemu, executa nativamente a maior parte do tempo, a fim de gerar um melhor desempenho. É executado em um grande número de sistemas operacionais hospedeiros. Sistemas operacionais hospedeiros em que o VirtualBox está disponível, de acordo com a Oracle, como pode ser observado no Quadro 2.

Quadro 2 – Sequência de instruções da Tradução Dinâmica.

| | |
|---|---|
| Windows | Windows Vista SP1, Windows Server 2008, Windows Server 2008, Windows 7, Windows 8, Windows 8.1, Windows 10, Windows Server 2012, Windows Server 2012 R2. |
| Mac OS X | 10.8 (Mountain Lion), 10.9 (Mavericks), 10.10 (Yosemite), 10.11 (El Capitan). |
| Linux | Debian 6.0 e 8.0, Fedora 6 até 22, Gentoo, Mandriva 2011, Ubuntu 10.04 ao 15.04, Oracle Enterprise Linux 5, Oracle Linux 6 e 7, Redhat Enterprise Linux 5, 6 e 7, OpenSUSE 11.4, 12.1, 12.2, 13.1. |
| Solaris | Solaris 10, Solaris 11. |

**Fonte**: Adaptado pelo autor, 2017.

O VirtualBox apresenta muitas características, dentre elas a mais importante é a tecnologia de multiprocessamento de convidado, onde, cada máquina virtual

pode possuir até 32 processadores virtuais, independentemente do processador do *sistema host*.

O VirtualBox virtualiza praticamente todo o *hardware* do sistema hospedeiro. Uma das principais interfaces virtualizadas são as interfaces de controladores de *hardware* ATA e SATA e os adaptadores SCSI. Essas interfaces são virtualizadas porque muitos sistemas operacionais são bastante exigentes com os dispositivos de inicialização, não aceitando o uso de dispositivos genéricos. Um dos poucos componentes não virtualizados são os dispositivos gráficos. O VirtualBox cria um dispositivo genérico simples com o padrão VGA com um conjunto de registro que são utilizados pela BIOS (ORACLE, 2016).

### *2.5.2 VMware Workstation Player*

Assim como o VirtualBox, o VMware Workstation Player é ideal para a virtualização em servidores *desktop*, ou seja, destinado para usuários comuns e pequenas empresas. É um Hypervisor do tipo II que, também, aplica a técnica de virtualização total. A primeira versão foi lançada em 1999 para ambientes *desktop*.

O VMWare utiliza o sistema operacional nativo para suporte dos dispositivos de entrada e saída que são gerenciados por uma aplicação chamada VMApp. Quando o VMApp é executado, o VMDriver, que é um driver utilizado pelo VMApp, é instalado e carregado diretamente no núcleo do sistema operacional nativo para estabelecer os privilégios da máquina virtual sobre o *hardware* da máquina física. Com isso, a VM (Virtual Machine) e o S.O. passam a compartilhar o tempo de processamento de CPU. Quando uma tarefa de entrada e saída é realizada, o Hypervisor as intercepta e encaminha a tarefa ao VMApp, que realiza as instruções de entrada e saída através de chamadas de sistema do S.O. nativo. Com isso, todo o suporte de dispositivos de entrada e saída nativos ficarão disponíveis para o Hypervisor, ao contrário do VirtualBox, que virtualiza os dispositivos de entrada e saída.

Assim como o VirtualBox, o VMware também possui a tecnologia de multiprocessamento, onde, uma máquina virtual pode possuir até 32 processadores virtuais. Em arquiteturas 64 *bits*, cada máquina virtual suporta até 64GB de RAM. Já em arquiteturas 32 *bits* o máximo de memória RAM por máquina virtual não passa de 8GB (VMWARE, 2016).

O VMware tem suporte para vários sistemas operacionais disponíveis no mercado hoje. No Quadro 3, seguem alguns sistemas operacionais hospedeiros que suportam o VMware.

Quadro 3 – Sistemas operacionais suportados pelo VMware.

| | |
|---|---|
| Windows | Windows 7, Windows 8, Windows 8.1, Windows 10, Windows Server 2008 e 2012. |
| Linux | Ubuntu 8.04 ou superior, Red Hat Enterprise Linux 5 ou superior, CentOS 5.0 ou superior, openSUSE 10.2 ou superior e SUSE Linux 10 ou superior. |

**Fonte**: Adaptado pelo autor, 2017.

### *2.5.3 Qemu-KVM*

O Qemu utiliza a técnica de tradução dinâmica, ou seja, não é preciso fazer nenhuma alteração no *host system*, ao contrário das outras ferramentas já citadas. O Qemu oferece dois modos de operação: Emulação total do sistema e emulação no modo usuário.

A emulação total emula um sistema completo, que pode ser com um ou mais processadores, e a maioria de seus periféricos. Já a emulação no modo usuário, executada somente em Linux, pode executar processos Linux em diferentes plataformas, como por exemplo, executar um programa desenvolvido para a arquitetura x86 em um Power PC (MAZIERO, 2013).

O Qemu está disponível para as arquiteturas x86, PowerPC, MIPS e Sparc 32 e 64. Os principais dispositivos emulados são os dispositivos de vídeo (VGA), portas seriais, *mouse* e teclado, disco rígido e adaptadores de rede. O Qemu possui suporte a auto emulação, onde é possível executar o Qemu dentro de outro Qemu (OLIVEIRA, 2007).

Como o processo de tradução dinâmica afeta o desempenho da máquina virtual, existe uma ferramenta chamada KVM, onde o Hypervisor consegue otimizar o seu desempenho. Com a utilização dessa ferramenta, o Qemu executa chamadas de sistemas da máquina virtual diretamente sobre *host system*, ao invés de interpretar. Os processos do sistema hóspede passam a ser executados diretamente no modo usuário do sistema hospedeiro. O modo núcleo das máquinas virtuais é utilizado apenas para virtualizar os processadores (MAZIERO, 2013).

## 2.6 Sistemas Operacionais – Hospedeiro e Hóspede

Neste capitulo será abordado os sistemas operacionais utilizados no *benchmark*, que será apresentado como resultado no final deste projeto. Serão abordadas algumas das principais características principais desses sistemas, para entendermos melhor os sistemas que serão utilizados. Os sistemas operacionais utilizados serão o GNU/Linux Ubuntu 15.10 e GNU/Linux Debian 8.0, onde, o Ubuntu será o sistema operacional hospedeiro e o Debian será o sistema operacional hóspede. Ambos os sistemas operacionais são *softwares* livres.

### *2.6.1 O Sistema GNU/Linux*

O SO GNU/Linux é um ambiente repleto de aplicativos e utilitários (ferramentas do projeto GNU), e sua parte mais importante é o *kernel*; pois isso a denominação Linux.

#### *2.6.1.1 O Projeto GNU*

“O sistema operacional de Richard Stallman recebeu o nome de Projeto GNU ou sistema operacional GNU. A palavra GNU, originalmente, refere-se a um mamífero ruminante, semelhante a um búfalo, com chifres espiralados, que vive no continente africano” (MOTA FILHO, 2012, p. 51).

A palavra GNU foi usada como trocadilho por Stallman, que quer dizer: ‘GNU's Not Unix’. Com essas palavras, Stallman quis dizer que o seu sistema é completamente diferente do Unix, no sentido de ser um *software* livre, que com o passar do tempo o UNIX deixou de ser um sistema livre.

O GNU é um sistema constituído de aplicativos livres, desenvolvido com o intuito de ser um sistema livre (MOTA FILHO, 2012).

Ainda no início do Projeto GNU, em 1985, Richard Stallman estabeleceu o ‘Manifesto GNU’, que nada mais era um pedido de participação e ajuda de outros desenvolvedores. O projeto GNU foi uma forma, principalmente, de resgatar o espírito de compartilhamento de conhecimento entre programadores, como acontecia nos grupos de compartilhamento do qual Richard Stallman e outras

centenas de programadores e universitários faziam parte, antes das empresas começarem a proteger os seus códigos-fontes.

O GNU iniciou-se pelo seu compilador, o GCC (GNU C Compiler). Em 1984, a partir do GCC, Stallman desenvolveu seu primeiro aplicativo, o GNU Emacs, um editor de texto, que até hoje, é muito utilizado. No ano seguinte, o Emacs já podia ser utilizado. O editor de texto permanece disponível nos servidores do MIT até hoje (MOTA FILHO, 2007).

#### *2.6.1.2 Kernel Linux*

O Linux surgiu em 1991 na Finlândia, desenvolvido por um estudante de Ciências da Computação chamado Linus Torvalds, que na época de seu desenvolvimento, não se passava de um projeto pessoal. No começo, o projeto era apenas baseado no Minix (NEMETH; SNIDER; HEIN; 2007).

O Minix foi um sistema operacional criado por um professor de Ciências da Computação, Andrew Stuart Tanenbaum em 1986, para fins de estudo, já que na época, utilizava o Unix para realizar os seus estudos, mas este acabou sendo privatizado, tendo o seu código fechado apenas para a equipe de desenvolvimento gerenciada pela AT&T na época, o que acabou inviabilizando os estudos de cientistas e estudantes em todo o mundo como Tanenbaum (MOTA FILHO, 2007).

#### ***2.6.1.3 Software Livre***

O conceito de *software* livre define que todo o código-fonte de cada *software* desenvolvido será distribuído junto ao programa. O *software* em questão pode ser pago ou gratuito. O código fonte do *software* pode ser alterado e redistribuído por qualquer um que tenha acesso ao código-fonte (MOTA FILHO, 2007).

Com o crescimento do projeto GNU, Richard Stallman necessitava de investimentos financeiros para continuar com o seu projeto. Para resolver essa situação, Stallman criou a Free Software Foundation (FSF), que tinha por objetivo arrecadar dinheiro para dar continuidade ao Projeto GNU. A FSF possui várias pessoas, ainda nos dias de hoje, que desenvolvem e mantêm o projeto GNU funcionando.

A respeito da Licença do GNU, foi criada a GNU GPL (General Public License) desenvolvida pela Free Software Foundation, com o objetivo de especificar se um *software* é livre (código-fonte aberto) ou não (código-fonte fechado). A GNU GPL é baseada em executar, estudar, modificar e redistribuir suas versões modificadas (MOTA FILHO, 2007).

### *2.6.2 Linux Ubuntu – Sistema Hospedeiro*

O Linux Ubuntu surgiu em 2004, desenvolvido por Mark Shuttleworth e um grupo de desenvolvedores criado por ele. Foi desenvolvido utilizando as primeiras versões do Debian com base. Em pouco mais de três anos após seu desenvolvimento, o Ubuntu já possuía mais de doze mil membros e oito milhões de usuários. Atualmente, o Ubuntu é a distribuição Linux mais utilizada no mundo (CANONICAL, 2016).

Uma nova versão do Ubuntu é lançada a cada seis meses, sendo que, cada versão possui suporte para 18 meses. Também existem várias derivações do Ubuntu como o Edubuntu, Kubuntu e Xubuntu, por exemplo.

A estrutura de funcionamento do Ubuntu é bem parecida com o Debian. O Ubuntu destaca-se mais por causa de sua interface bem trabalhada, é conhecido por ter a interface mais bonita entre todas as distribuições, mas no geral, como é baseado no Debian, o seu funcionamento é similar (CANONICAL, 2016).

### *2.6.3 Linux Debian – Sistema Hóspede*

O projeto Debian foi fundado em 1993 por Ian Murdock. O Debian é uma combinação das ferramentas GNU, desenvolvidas por Richard Stallman, com o *Kernel* Linux. Formando assim, a distribuição chamada Debian GNU/Linux, que é um *software* livre (DEBIAN, 2016).

O Debian é composto por pacotes de *software*, onde, cada pacote contém executáveis, *scripts*, documentação, informações. O Debian é uma das distribuições Linux mais popular no mundo, devido à sua excelência técnica e funcionalidades. O que fez o Debian diferenciar das outras distribuições Linux foi o sistema de gestão de pacotes, sendo a primeira distribuição a utilizar este recurso e a primeira a atualizar para uma versão mais nova do sistema sem a necessidade de reinstalação.

O gerenciador de pacotes é uma ferramenta de administrador do sistema que tem o controle completo dos pacotes instalados no sistema, sendo possível atualizar um único pacote ou o sistema operacional inteiro.

Atualmente o Debian está disponível em sua última versão, o Debian 8.0, sob o codinome de Jessie. O Debian não impõe nenhum requisito de *hardware* para que seja executado, sendo possível executá-lo em qualquer plataforma disponível (DEBIAN, 2016).

## 2.7 Phoronix Test Suite

O PTS é uma plataforma de *benchmarking* disponível para Linux, Solaris, Mac OS X e sistemas operacionais BSD. Foi lançada em 2008 pela Phoronix Media e é uma plataforma *open source* sob a licença GNU/GPL. Os testes realizados pelo Phorononix são realizados de forma automatizada, desde a instalação até a divulgação dos resultados. O PTS pode ser executado em plataformas que vão desde *smartphones* e *desktop* a infraestruturas de computação em nuvem (PHORONIX, 2016).

A plataforma do PTS possui uma estrutura abrangente para que novos testes possam ser incorporados a ele. O PTS é utilizado por organizações para propósitos internos de garantia de qualidade, validação de *hardware*, entre outros, realizando *benchmarks* qualitativos e quantitativos de forma limpa e fácil.

O PTS, além de realizar a execução de testes, possui um conjunto de finalidades que o torna uma das principais ferramentas de *benchmark* existentes hoje, como por exemplo, a abundância de opções de análises dos resultados dos testes, permitindo a junção de vários conjuntos de testes, validação de testes, entre outros. Depois de gerados, os testes são disponibilizados no *site*[3], que é um repositório de testes, suítes e dados gerados pelo *benchmark* (PHORONIX, 2016).

Depois de instalada a plataforma, os testes podem ser adicionados um por um, conforme a necessidade do usuário. O usuário escolhe os testes a serem utilizados e a plataforma baixa os arquivos necessários para a execução dos testes. O PTS possui uma infinidade de testes que levam o computador ao extremo. A seguir, serão discutidos os testes que serão usados neste projeto. O Phoronix Test

[3] ***Openbenchmarking site:*** *http://openbenchmarking.org*

Suite está disponível para *download* no *site*[4]. A versão utilizada no projeto será a v6.4.0 sob o codinome de Hasvik.

[4] **PTS *site:*** http://www.phoronix-test-suite.com/?k=downloads

## 3 PROCEDIMENTOS METODOLÓGICOS

Este projeto teve como base durante o seu referencial teórico pesquisas bibliográficas de diversos autores baseados em livros, projetos de conclusão de curso de pós-graduação, dissertações e teses de mestrado e doutorado de diversas faculdades, universidades e institutos do Brasil, e diversos artigos científicos publicados nacionalmente e internacionalmente. Foram levantadas informações através de pesquisas referentes a vários estudos relacionados à virtualização, entre eles os principais foram os estudos de ambientes de virtualização, os benefícios que a realização da virtualização em ambientes de trabalho pode oferecer, o que são e como funcionam as máquinas virtuais, como funcionam as técnicas de virtualização, sistemas operacionais nativos e hospedeiros. Esses são os principais tópicos abordados na primeira parte do projeto.

Para a segunda parte do projeto, será feito um estudo de caso através de um *benchmark* para a comparação de desempenho entre três ferramentas de virtualização muito utilizadas em ambientes domésticos e pequenas empresas. As ferramentas utilizadas serão o VirtualBox versão 4.3.36, o Qemu versão 2.6.0 e o VMWare Workstation Player 12. Essas ferramentas foram definidas a partir das pesquisas feitas durante o referencial teórico, na qual também se definiu a ferramenta de *benchmark* que será utilizada. As ferramentas de virtualização serão instaladas em um computador que foi preparado e montado para os testes. O computador foi preparado com a instalação dos sistemas operacionais mencionados no trabalho, no qual o Linux Ubuntu 15.04 será o sistema hospedeiro e o Debian

8.5.0 será o sistema operacional convidado. Depois de concluído todo o processo de montagem e execução dos testes, serão analisados os resultados para se obter uma resposta de qual virtualizador possui o maior desempenho realizando os testes de forma mais rápida.

Para a realização deste projeto foi utilizado o seguinte computador para a execução do *benchmark*:

*Hardware*:

- Processador: Intel Core i5-3210M @ 3.10GHz (4 Cores)
- Placa-mãe: ASUS K45A v1.0
- Chipset: Intel 3rd Gen Core DRAM
- Memória RAM: 6144MB
- Disco Rígido: 500GB TOSHIBA MQ01ABD0
- Gráfico: Intel HD 4000 (1100MHz),
- Áudio: Intel 7 /C210
- Rede: Realtek RTL8111/8168/8411 + Qualcomm Atheros AR9485 Wireless

*Software:*

- Sistema Operacional Nativo: Ubuntu 15.04
- Kernel: 3.19.0-15-generic (x86_64)
- Ambiente Gráfico: Unity 7.3.2
- Display Server: X Server 1.17.1
- Display Driver: Intel 2.99.917
- Compilador: GCC 4.9.2
- Sistema de Arquivos: ext4
- Resolução: 1366x768

Plataforma de *Benchmark*:

- Phoronix Test Suite 6.4.0 codinome Hasvik

Testes realizados:

- Testes de processador
- Testes de memória
- Testes de disco

# 4 RESULTADOS

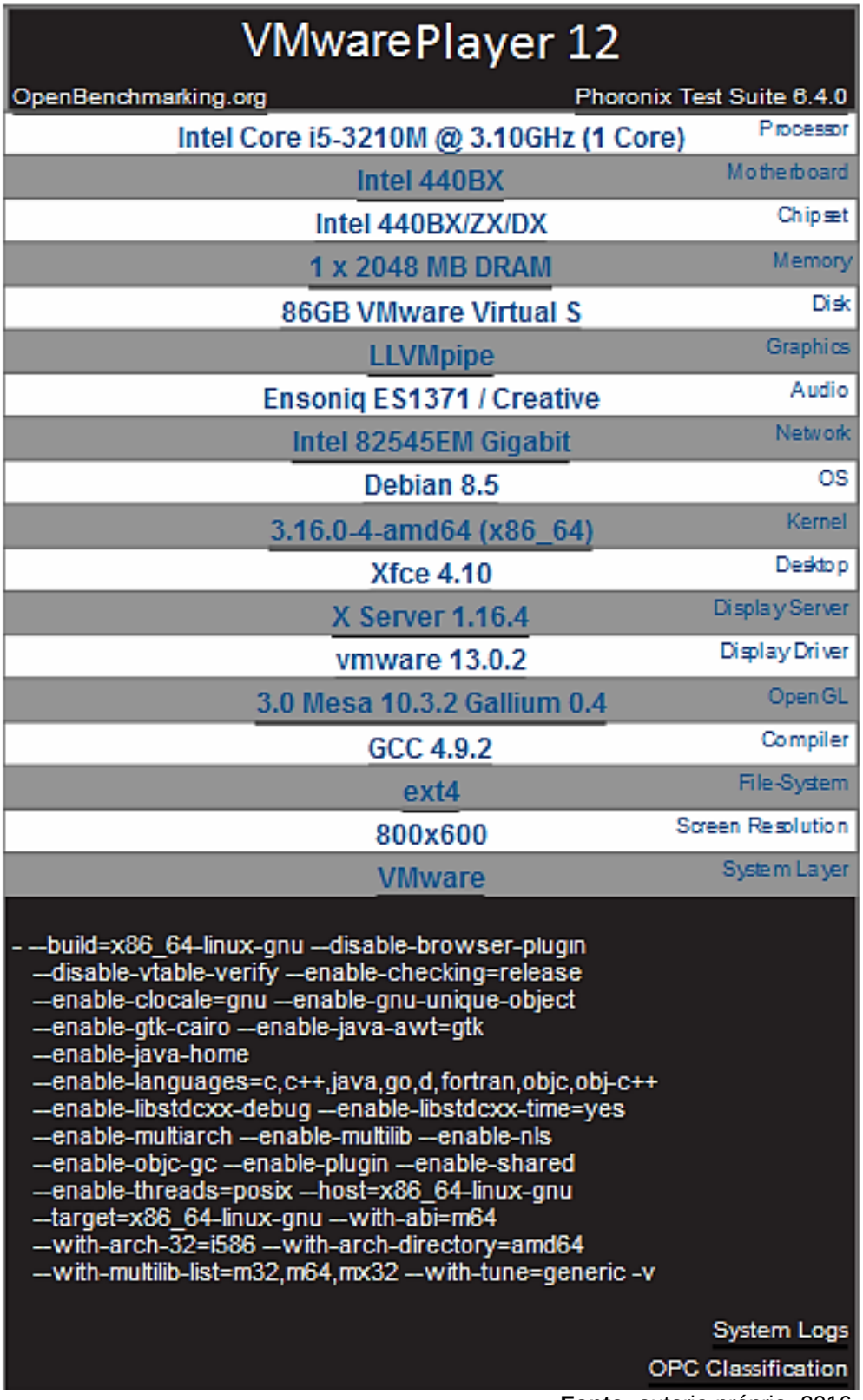
Neste capítulo serão realizados os testes através das máquinas virtuais instaladas no computador preparado especialmente para este *benchmark*; todas as máquinas virtuais foram preparadas da mesma forma, em ambas as ferramentas, com as mesmas configurações de *hardware* para que estas estejam em igualdade durante a realização dos testes. O sistema operacional utilizado na virtualização é o Linux Debian 8.5.0 64 *bits* sob o codinome de Jessie.

Para cada ferramenta de virtualização foi preparada uma máquina virtual, onde ambas possuem as configurações presentes nas Figuras 9, 10 e 11.

Figura 9 - Configuração da máquina virtual no VirtualBox 4.3.36.

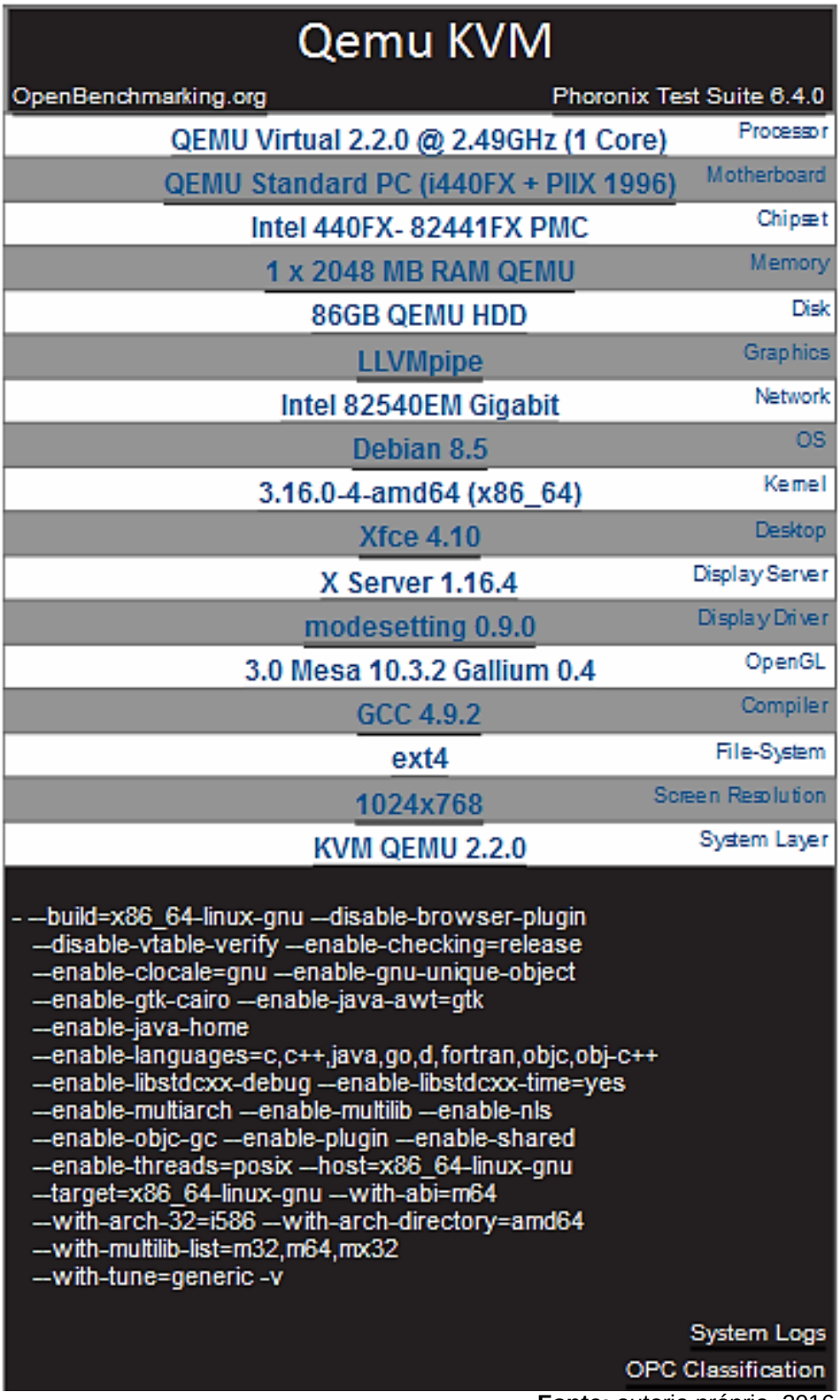

**Fonte:** autoria própria, 2016.

Figura 10 - Configuração da máquina virtual no VMware Workstation Player 12.




**Fonte:** autoria própria, 2016.

Figura 11 - Configuração da máquina virtual no Qemu-KVM.




**Fonte:** autoria própria, 2016.

## 4.1 Testes de Processador

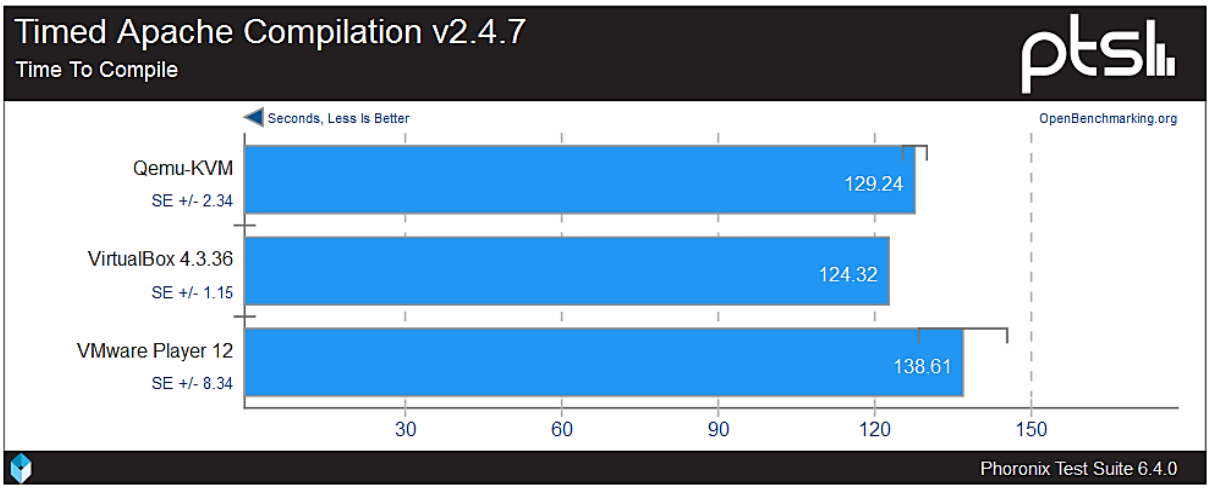
Os seguintes testes foram realizados a fim de se medir a velocidade dos processadores utilizados por cada uma das máquinas virtuais com o objetivo de concluir qual máquina realiza as operações de processador em um menor tempo.

### *4.1.1 Build-Apache*

Este teste consiste em um conjunto de ferramentas que mede a quantidade de tempo necessário para construir pacotes comuns de *software* de código aberto. Este teste utilizará o compilador padrão (GCC) e irá ajustar o número de compilação de postos de trabalho com base no número de núcleos de CPU detectados no sistema. Esta ferramenta é utilizada para testar o processador e o desempenho geral do sistema. O Build-Apache testa quanto tempo leva para construir um Servidor Apache HTTP. O resultado deste teste é dado por segundo, sendo o menor tempo o melhor (OPENBENCHMARKING.ORG, 2016).

De acordo com os testes realizados, conforme as Figuras 12 e 13, pode-se observar que houve um equilíbrio de desempenho entre as três ferramentas durante o teste, o VirtualBox e o Qemu-KVM obtiveram um desempenho semelhante, mas o VirtualBox obteve um resultado melhor, conseguindo realizar as funções com um tempo de 124,32 segundos, enquanto o Qemu-KVM conseguiu em 129,24. Já o VMware Player ficou um pouco atrás, com um tempo de 138,61.

Figura 12 - Tempo de execução Build-Apache.




**Fonte:** autoria própria, 2016.

Figura 13 - Resultados Build-Apache.

Build-Apache

| | Qemu-KVM | VirtualBox 4.3.36 | VMware Player 12 |
|---|---|---|---|
| build-apache: Time To Compile | 129.24 | 124.32 | 138.61 |
| Difference | 1.07x | 1.10x | 1.00x |
| Standard Error | 2.34 | 1.15 | 8.34 |
| Standard Deviation | 3.14% | 1.60% | 14.74% |

OpenBenchmarking.org

**Fonte:** autoria própria, 2016.

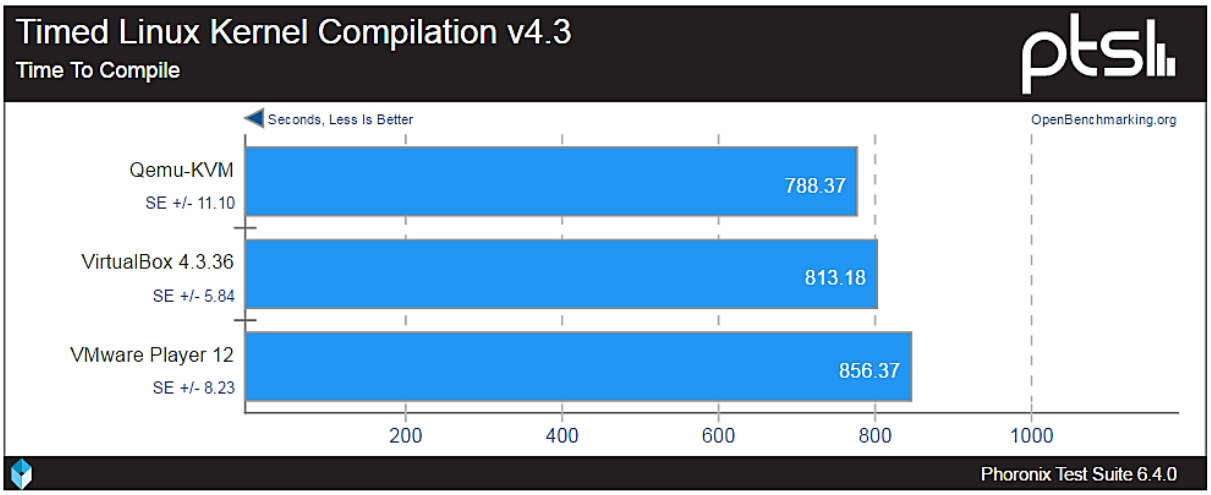
### *4.1.2 Build-Linux-Kernel*

Este conjunto de testes é parecido com o Build-Apache, pois consiste em um teste que mede a quantidade de tempo necessário para se construir pacotes comuns de *software* de código aberto. Este teste também utilizará o compilador padrão GCC e ajustará o número de compilação de postos de trabalho com base no número de núcleos de CPU detectados no sistema. Esta ferramenta, assim como o Build-Apache, também é utilizada para testar o processador e o desempenho geral do sistema. O objetivo dessa ferramenta é testar quanto tempo se leva para construir o *Kernel* do Linux. O resultado deste teste é dado por segundo, sendo o menor tempo o melhor (OPENBENCHMARKING.ORG, 2016).

De acordo com os testes realizados com o Build-Linux-Kernel, os resultados se mostraram melhor com o Qemu-KVM obtendo o melhor tempo, conseguindo compilar os pacotes de código-fonte com uma média de 788,37 segundos, conforme

as Figuras 14 e 15. Enquanto a segunda melhor ferramenta foi o VirtualBox, executando a compilação com uma média de 813,18 segundos, com apenas 24,81 segundos mais lento que o Qemu-KVM, mostrando mais uma vez um desempenho parecido entre as duas ferramentas. Já o VMware Player conseguiu um tempo de 856,37 segundos, ficando com o pior desempenho neste teste.

Figura 14 - Tempo de execução Build-Linux-Kernel.




**Fonte:** autoria própria, 2016.

Figura 15 - Resultado Build-Linux-Kernel medido em segundos.

Build-Linux-Kernel

| | Qemu-KVM | VirtualBox 4.3.36 | VMware Player 12 |
|---|---|---|---|
| build-linux-kernel: Time To Compile | 788.37 | 813.18 | 856.37 |
| Difference | 1.08x | 1.05x | 1.00x |
| Standard Error | 11.10 | 5.84 | 8.23 |
| Standard Deviation | 2.44% | 1.24% | 1.66% |

OpenBenchmarking.org

**Fonte:** autoria própria, 2016.

### *4.1.3 Pbzip2-Compress*

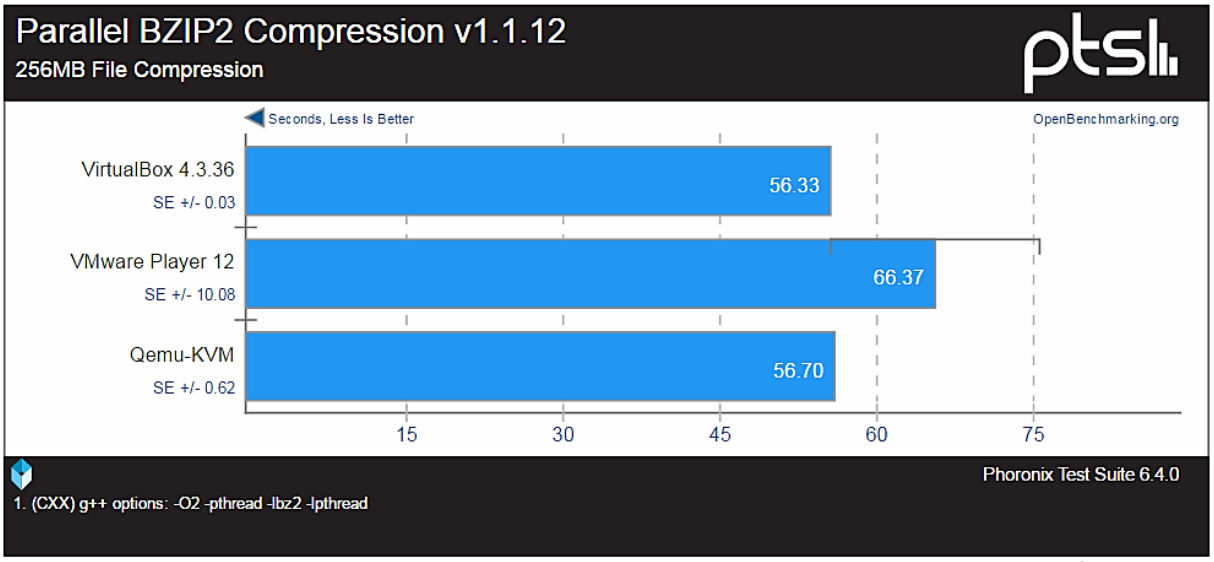
Este teste mede o tempo necessário para compactar um arquivo para o formato .tar, formato padrão de arquivos compactado do Linux utilizando a ferramenta de compressão Bzip2 (OPENBENCHMARKING.ORG, 2016).

O Bzip2, neste teste, comprimirá os dados em blocos de tamanho entre 100 KB e 900 KB e utilizará um algoritmo chamado de Burrows-Wheeler para converter

sequências de caracteres com frequência recorrentes em blocos idênticos. A versão atual deste teste utiliza arquivos de teste de 256 MB de compressão. O resultado deste teste também é dado por segundo, sendo o menor tempo o melhor.

Conforme as Figuras 16, 17 e 18, mais uma vez, houve um resultado bem semelhante entre o Qemu-KVM e o VirtualBox. As duas ferramentas obtiveram uma velocidade de compactação na casa dos 56 segundos, mas com uma diferença de apenas 0,67 milésimos o VirtualBox obteve o melhor resultado com 56,33, enquanto o Qemu-KVM conseguiu 56.70 segundos. O VMware Player, novamente ficou com o pior resultados nos testes de processador, com uma velocidade de 66.37 segundos.

Figura 16 - Tempo de execução Pbzip2-Compress.




**Fonte:** autoria própria, 2016.

Figura 17 - Resultado Pbzip2-Compression.

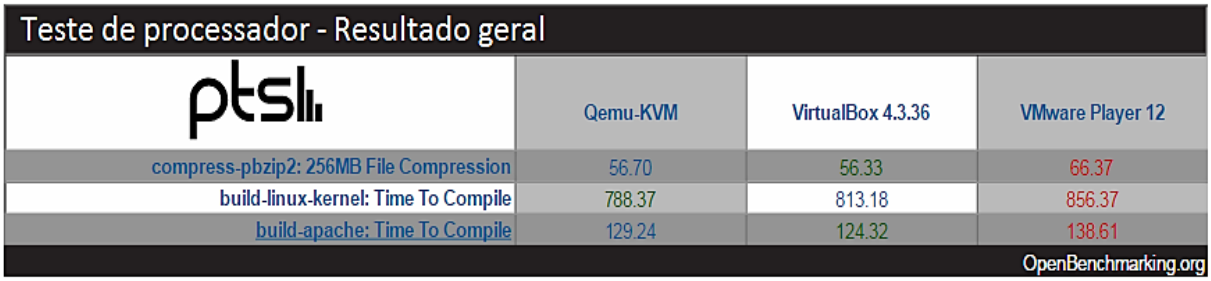
Pbzip2-Compress

| | VirtualBox 4.3.36 | VMware Player 12 | Qemu-KVM |
|---|---|---|---|
| compress-pbzip2: 256MB File Compression | 56.33 | 66.37 | 56.70 |
| Difference | 1.15x | 1.00x | 1.15x |
| Standard Error | 0.03 | 10.08 | 0.62 |
| Standard Deviation | 0.09% | 37.21% | 1.91% |

OpenBenchmarking.org

**Fonte:** autoria própria, 2016.

Figura 18 - Resultado geral dos Testes de Processador.




**Fonte:** autoria própria, 2016.

## 4.2 Testes de memória

Os testes a seguir foram realizados a fim de se medir a velocidade de memória utilizadas por cada uma das máquinas virtuais com o objetivo de concluir qual máquina possui o melhor desempenho de memória realizando os testes em um menor tempo.

### *4.2.1 Ramspeed*

Ferramenta de teste de memória do sistema que consiste em sub testes destinados a testar o desempenho da memória RAM do computador. O Ramspeed mede a largura de banda efetiva de ambos os subsistemas de *cache* e memória. O principal objetivo dessa ferramenta é realizar o maior número de tarefas no menor tempo possível (OPENBENCHMARKING.ORG, 2016).

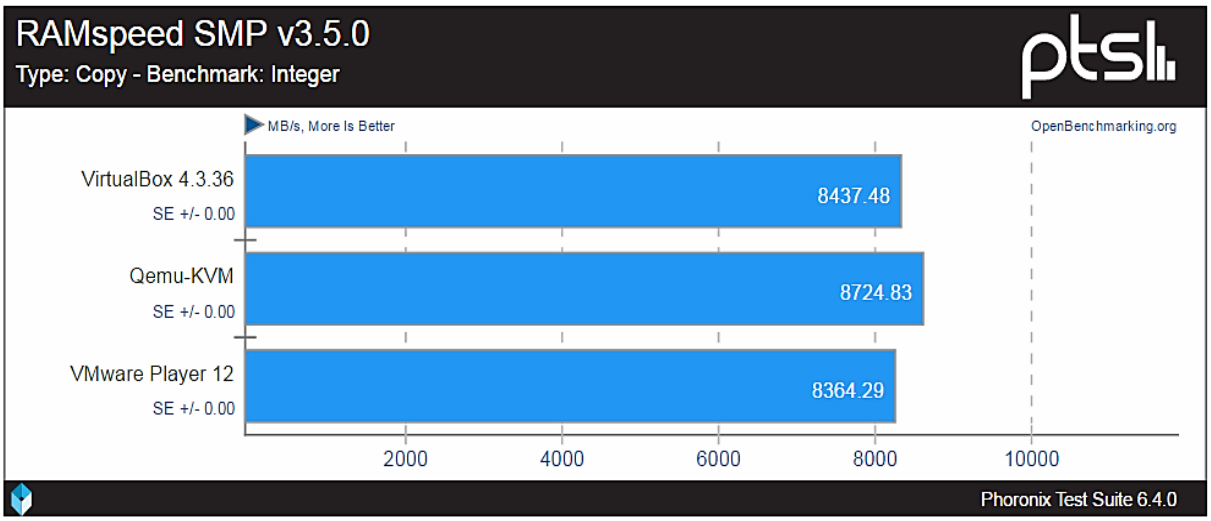
A simulação do Ramspeed foi dividida em quatro sub testes para medir os diferentes aspectos de desempenho da memória. Os dois primeiros sub testes são os testes de cópia, que consistem na transferência de um local da memória para outro utilizando valores inteiros e pontos flutuantes. Já o terceiro e o quarto sub teste é o de soma, que consiste na leitura dos dados a partir da primeira posição da memória que, em seguida, lê a segunda posição da memória, depois soma os valores da primeira e da segunda posição e escreve o resultado dessa soma em uma terceira posição da memória. Para o terceiro e quarto sub testes serão utilizados valores inteiros e pontos flutuantes. O resultado destes testes são dados em MB/s, sendo o maior valor o melhor resultado.

#### *4.2.1.1 Cópia*

4.2.1.1.1 Valores inteiros

O primeiro teste de memória realizado foi o sub teste de cópia com valores inteiros, onde, conforme as Figuras 19 e 20, os resultados favoreceram a ferramenta de emulação Qemu-KVM, que conseguiu operar os testes a uma velocidade de 8724,83 MB/s, ficando um pouco à frente das outras ferramentas. Já o VirtualBox obteve o segundo melhor desempenho, realizando os testes com uma velocidade de 8437,48 MB/s, enquanto o VMware Player executou em apenas 8364,29 MB/s.

Figura 19 - Tempo de execução Ramspeed Copy Integer.




**Fonte:** autoria própria, 2016.

Figura 20 - Resultado Ramspeed Copy Integer.

RAMspeed Copy Integer

| | VirtualBox 4.3.36 | Qemu-KVM | VMware Player 12 |
|---|---|---|---|
| ramspeed: Type: Copy - Benchmark: Integer | 8437.48 | 8724.83 | 8364.29 |
| Difference | 1.01x | 1.04x | 1.00x |

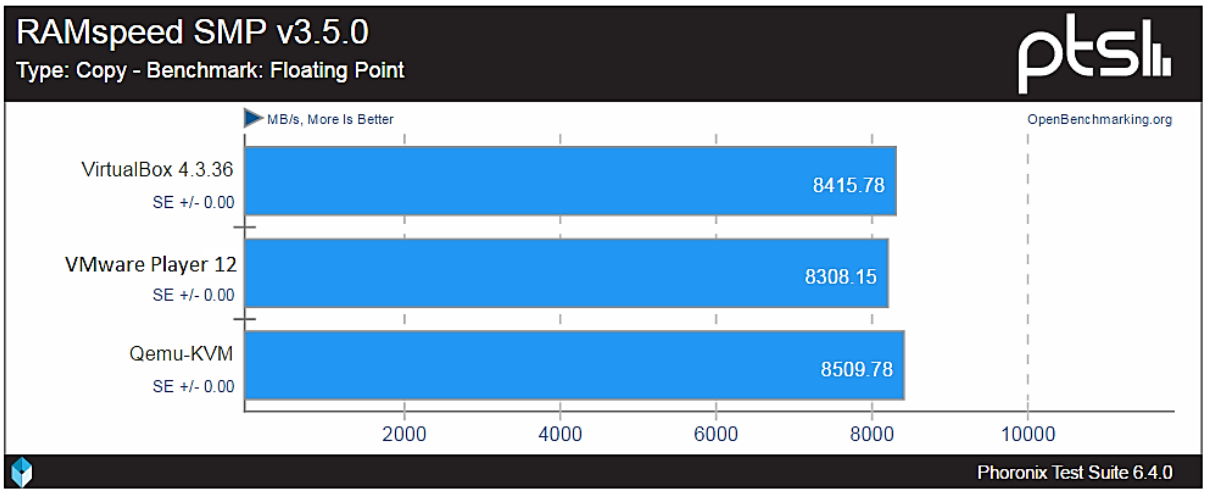
OpenBenchmarking.org

**Fonte:** autoria própria, 2016.

#### 4.2.1.1.2 Ponto Flutuante

Depois de realizados os testes de cópias com valores inteiros, o próximo sub teste a ser apresentado é o sub teste de cópia com pontos flutuantes. Como pode ser observado na Figura 21, novamente, o Qemu-KVM obteve o melhor resultado, apesar do VirtualBox ter conseguido um resultado semelhante com um bom desempenho. O Qemu-KVM realizou as operações com 8509.78 MB/s de velocidade contra 8415,78 MB/s do VirtualBox. O VMware Player, mais uma vez, com o pior resultado com apenas 8308,15 MB/s.

Figura 21 - Tempo de execução Ramspeed Copy Floating.




**Fonte:** autoria própria, 2016.

Figura 22 - Resultado Ramspeed Copy Floating.

RAMspeed Copy Floating

| | VirtualBox 4.3.36 | VMware Player 12 | Qemu-KVM |
|---|---|---|---|
| ramspeed: Type: Copy - Benchmark: Floating Point | 8415.78 | 8308.15 | 8509.78 |
| Difference | 1.01x | 1.00x | 1.02x |

OpenBenchmarking.org

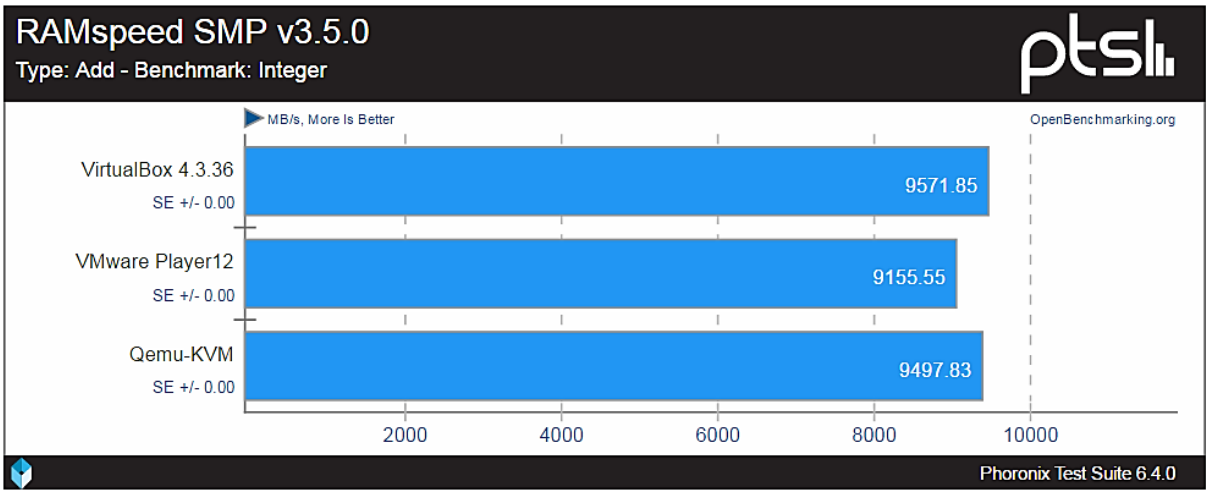
**Fonte:** autoria própria, 2016.

### ***4.2.1.2 Soma***

4.2.1.2.1 Valores Inteiros

O terceiro sub teste de memória a ser a realizado é o sub teste de soma com valores inteiros. Desta vez, conforme as Figuras 23 e 24, os resultados favoreceram a ferramenta VirtualBox. Novamente, o resultado mostrou uma igualdade muito grande no desempenho entre as máquinas VirtualBox e Qemu-KVM, onde, com uma pequena diferença, o VirtualBox conseguiu obter um melhor resultado com uma velocidade de 9571,85 MB/s de velocidade contra 9497,83 MB/s do Qemu-KVM. O VMware ficou em último, com uma velocidade de 9155,55 MB/s.

Figura 23 - Tempo de execução Ramspeed Add Integer.




**Fonte:** autoria própria, 2016.

Figura 24 - Resultado Ramspeed Add Integer.

RAMspeed Add Integer

| | VirtualBox 4.3.36 | VMware Player12 | Qemu-KVM |
|---|---|---|---|
| ramspeed: Type: Add - Benchmark: Integer | 9571.85 | 9155.55 | 9497.83 |
| Difference | 1.05x | 1.00x | 1.04x |

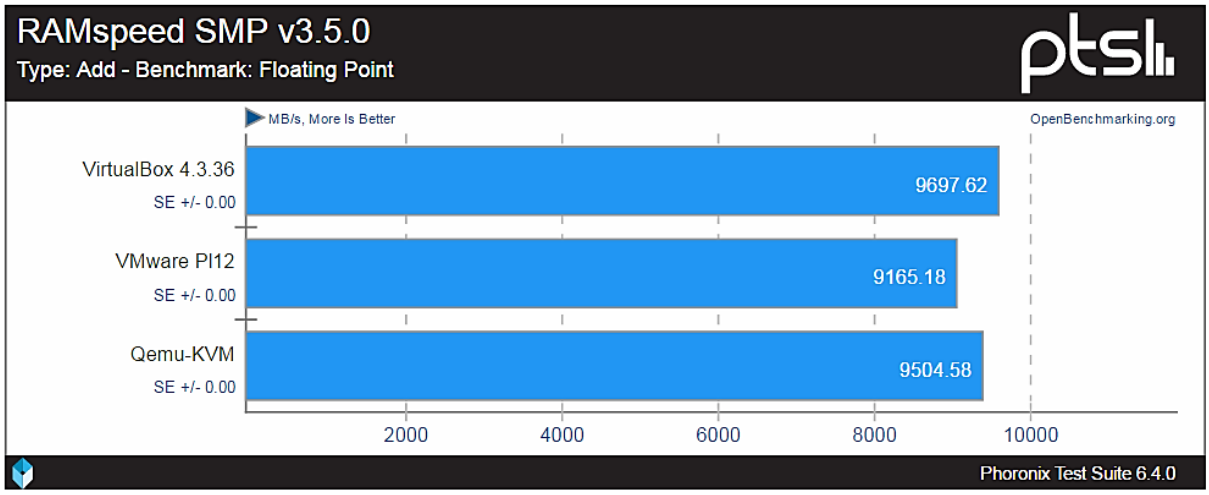
OpenBenchmarking.org

**Fonte:** autoria própria, 2016.

4.2.1.2.2 Ponto Flutuante

O quarto sub teste de memória a ser a realizado é o sub teste de soma com pontos flutuantes. Nesse teste, os resultados se mostraram favoráveis, mais uma vez, ao VirtualBox, que conseguiu realizar todas as operações deste teste com uma velocidade de 9697.92 MB/s, conforme as figuras 25, 26 e 27. O Qemu-KVM vem logo atrás com uma pequena diferença, realizando todas as operações a uma velocidade de 9504.58 MB/s. O VMware, novamente, ficou com o pior resultado, provando ser a pior ferramenta durante os testes de memória, com apenas 9165.18 MB/s.

Figura 25 - Tempo de execução Ramspeed Add Floating.




**Fonte:** autoria própria, 2016.

Figura 26 - Resultado Ramspeed Add Floating.

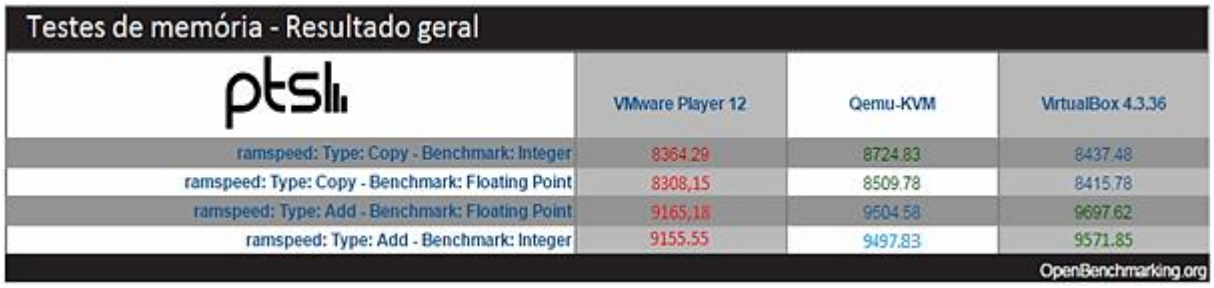
RAMspeed Add Floating

| | VirtualBox 4.3.36 | VMware Player 12 | Qemu-KVM |
|---|---|---|---|
| ramspeed: Type: Add - Benchmark: Floating Point | 9697.62 | 9165.18 | 9504.58 |
| Difference | 1.06x | 1.00x | 1.04x |

OpenBenchmarking.org

**Fonte:** autoria própria, 2016.

Figura 27 - Resultado geral dos Testes de Memória.




**Fonte:** autoria própria, 2016.

## 4.3 Testes de Disco

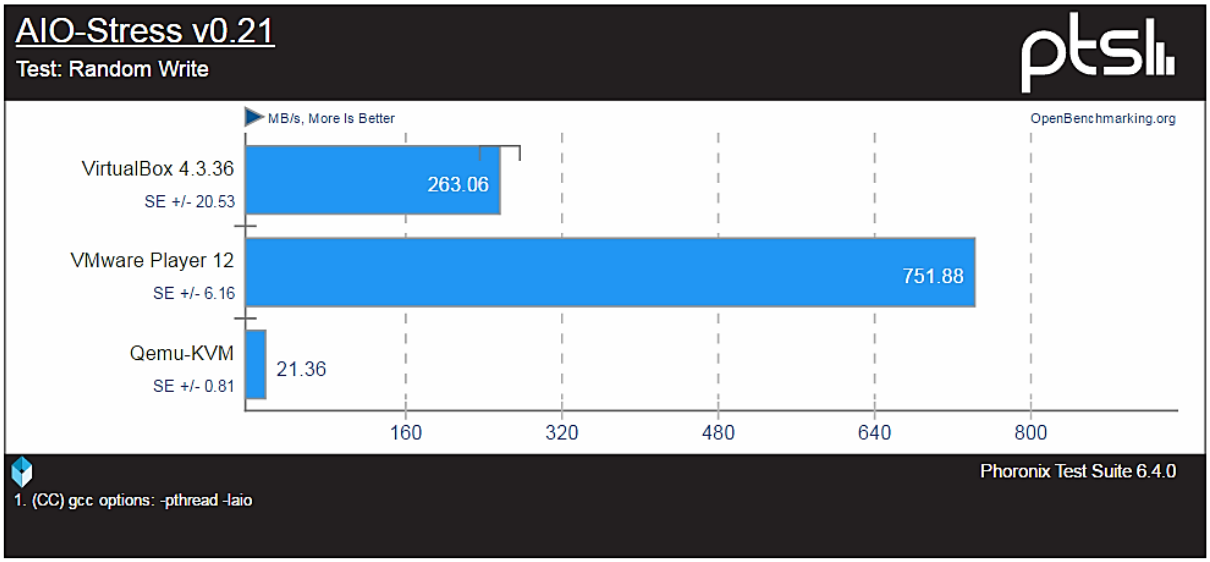
Os testes de disco a seguir, foram realizados a fim de se medir a velocidade dos discos virtuais utilizadas por cada uma das máquinas virtuais com o objetivo de concluir qual máquina possui o melhor disco virtual realizando os testes em um menor tempo.

### *4.3.1 AIO-Stress*

O primeiro teste de disco a ser realizado foi o AIO-Stress. Este teste foi projetado para conter os testes de disco e arquivo de sistemas do mundo real. O AIO-Stress é um teste de processo assíncrono de E/S que envia múltiplos dados de leitura aleatoriamente pelos diversos setores do disco rígido. A versão atual deste perfil utiliza um arquivo de teste de 2048 MB e um registro do tamanho de 64KB. O resultado deste teste é dado em MB/s, sendo o maior tempo o melhor (OPENBENCHMARKING.ORG, 2016).

Ao analisar os resultados desse teste, conforme as figuras 28 e 29, observa-se que os resultados favoreceram o VMware, que conseguiu realizar todas os processos de entrada e saída durante o teste a uma velocidade de 751.88 MB/s. Esse foi o teste que apresentou a maior diferença de desempenho entre as ferramentas, onde o VMware conseguiu executar todos os processos a uma velocidade 35 vezes mais rápida do que a ferramenta que obteve o pior resultado, no caso, o Qemu-KVM com 21.36 MB/s. Já o VirtualBox, ficou com o segundo melhor desempenho com 263.06 MB/s de velocidade, também apresentando uma execução bem inferior ao VMware.

Figura 28 - Tempo de execução AIO-Stress.




**Fonte:** autoria própria, 2016.

Figura 29 - Resultado AIO-Stress.

AIO-Stress

| | VirtualBox 4.3.36 | VMware Player 12 | Qemu-KVM |
|---|---|---|---|
| aio-stress: Test: Random Write | 263.06 | 751.88 | 21.36 |
| Difference | 12.32x | 35.20x | 1.00x |
| Standard Error | 20.53 | 6.16 | 0.81 |
| Standard Deviation | 19.12% | 1.42% | 9.32% |

OpenBenchmarking.org

**Fonte:** autoria própria, 2016.

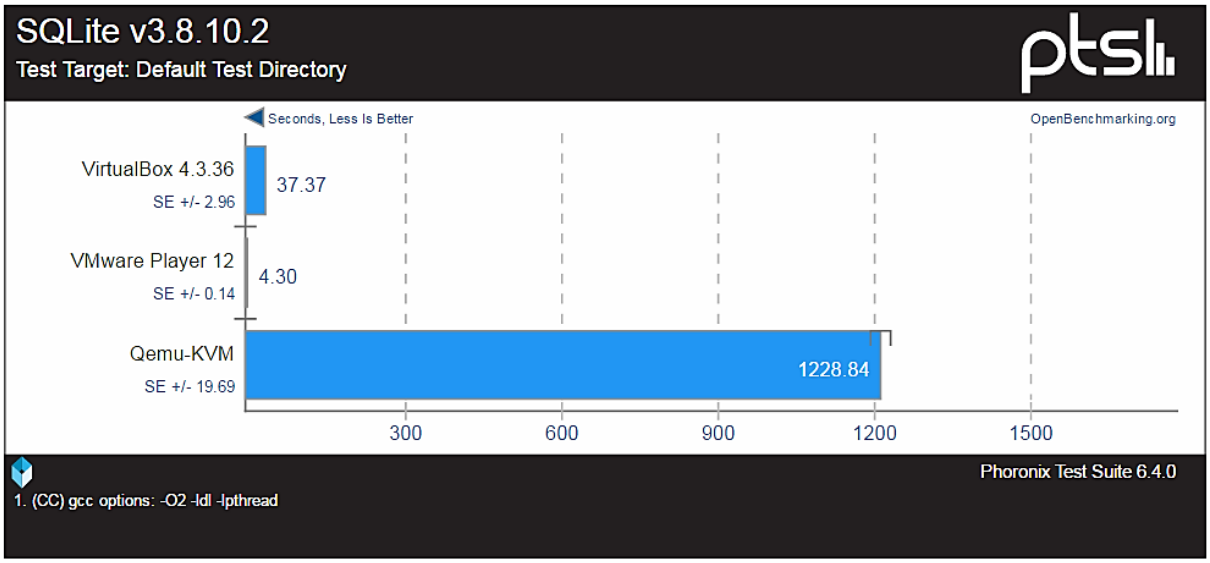
### *4.3.2 SQLite*

O SQLite é um banco de dados muito utilizado no mundo real. Esse teste utiliza o SQLite para medir o tempo para se executar um número predefinido de inserções em uma base de dados indexada. O resultado deste teste também é dado por segundo, sendo o menor tempo o melhor (OPENBENCHMARKING.ORG, 2016).

Mais uma vez, conforme as figuras 30 e 31, o VMware Player prova ser superior durante as execuções dos testes de disco. Com um tempo de 4.30 segundos, o VMware conseguiu executar os processos desse teste 285 vezes mais rápido do que a ferramenta com o pior desempenho, mais uma vez, o Qemu-KVM,

que executou os processos com um tempo de 1228.84 segundos. O VirtualBox obteve o segundo melhor tempo, mas com um resultado bem inferior ao VMware Player, 37.37 segundos.

Figura 30 - Tempo de execução SQLite.




**Fonte:** autoria própria, 2016.

Figura 31 - Resultado SQLite.

SQLite

| | VirtualBox 4.3.36 | VMware Player 12 | Qemu-KVM |
|---|---|---|---|
| sqlite: Test Target: Default Test Directory | 37.37 | 4.30 | 1228.84 |
| Difference | 1.97x | 2.00x | 1.00x |
| Standard Error | 2.96 | 0.14 | 19.69 |
| Standard Deviation | 19.37% | 8.25% | 3.20% |

OpenBenchmarking.org

**Fonte:** autoria própria, 2016.

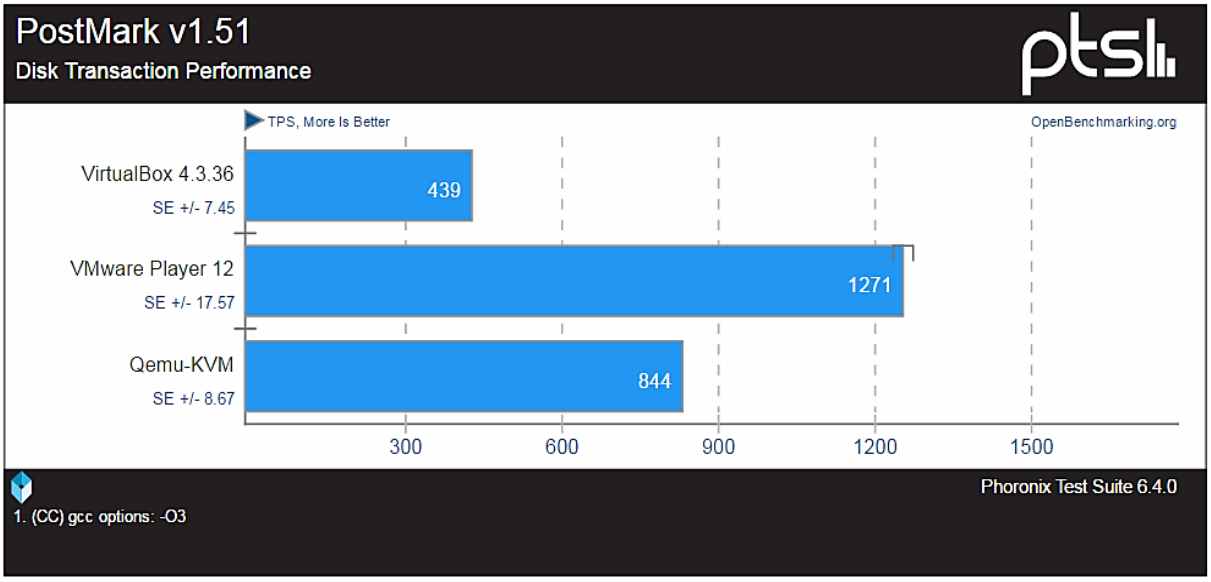
### *4.3.3 PostMark*

O PostMark é um teste desenvolvido pela NetApp projetado para simular testes em pequenos arquivos semelhante às tarefas enfrentadas por servidores *web* e de *e-mail*, como serviços de envio de *e-mails*, incluindo processamento de *spam* e estatísticas detalhadas. Para a realização dos testes, o PostMark é configurado para realizar 25.000 operações com 500 arquivos simultaneamente com arquivos que

variam entre 5 e 512 *kilobytes* de tamanho. O resultado deste teste também é dado por TPS (Transação Por Segundo), sendo o maior tempo o melhor (OPENBENCHMARKING.ORG, 2016).

De acordo com os resultados apresentados nas Figuras 32, 33 e 34, o VMware Player, mais uma vez, conseguiu o melhor resultado, sendo a ferramenta que obteve os melhores resultados durante os testes de disco. Com uma velocidade 1271 operações por segundo, o VMware se saiu melhor do que o Qemu-KVM com 844 transações por segundo, e o VirtualBox, pior desempenho, com uma velocidade de 439 transações por segundo. Este foi o teste de disco mais equilibrado entre as ferramentas, mesmo assim, com o VMware apresentando um processamento muito superior à ferramenta com o pior desempenho.

Figura 32 - Tempo de execução PostMark.




**Fonte:** autoria própria, 2016.

Figura 33 - Resultado PostMark.

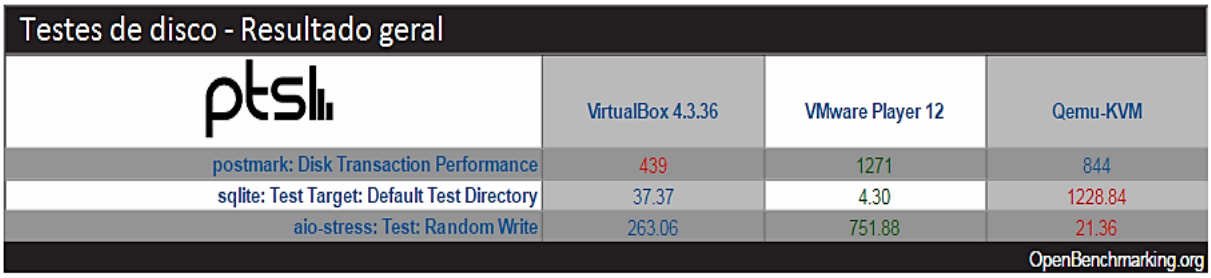
PostMark

| | VirtualBox 4.3.36 | VMware Player 12 | Qemu-KVM |
|---|---|---|---|
| postmark: Disk Transaction Performance | 439 | 1271 | 844 |
| Difference | 1.00x | 2.90x | 1.92x |
| Standard Error | 7.45 | 17.57 | 8.67 |
| Standard Deviation | 2.94% | 2.39% | 1.78% |

OpenBenchmarking.org

**Fonte:** autoria própria, 2016.

Figura 34 - Resultado geral dos Testes de Disco.



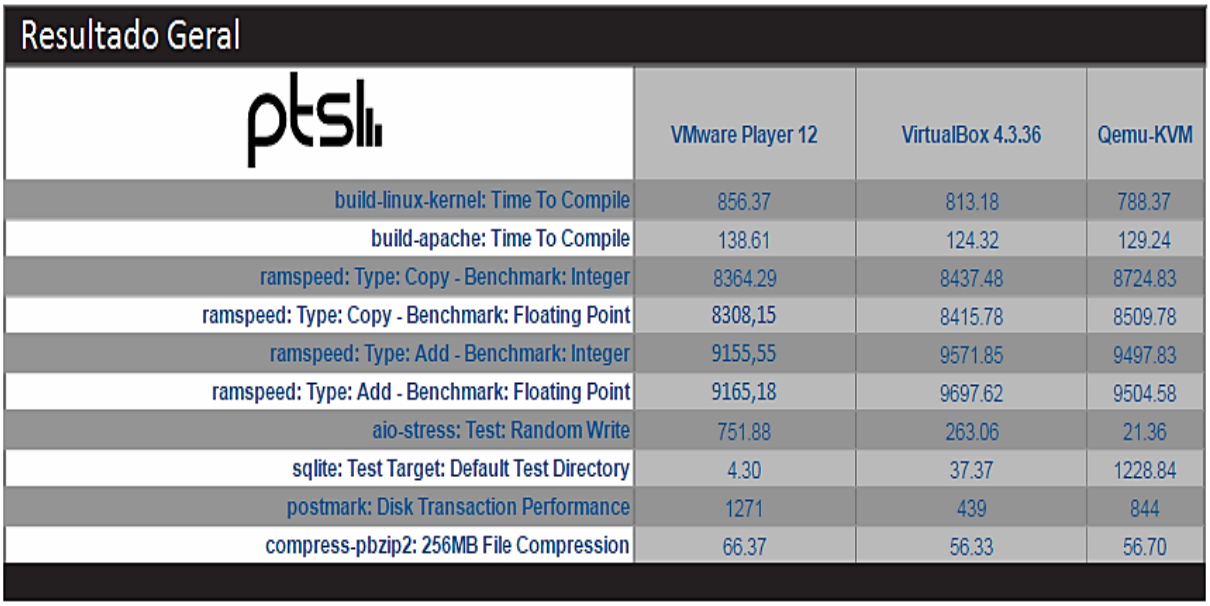
Testes de disco - Resultado geral

| | VirtualBox 4.3.36 | VMware Player 12 | Qemu-KVM |
|---|---|---|---|
| postmark: Disk Transaction Performance | 439 | 1271 | 844 |
| sqlite: Test Target: Default Test Directory | 37.37 | 4.30 | 1228.84 |
| aio-stress: Test: Random Write | 263.06 | 751.88 | 21.36 |

OpenBenchmarking.org

**Fonte:** autoria própria, 2016.

Figura 35 - Resultado geral de todos os testes realizados no Phoronix Test Suite.



Resultado Geral

| | VMware Player 12 | VirtualBox 4.3.36 | Qemu-KVM |
|---|---|---|---|
| build-linux-kernel: Time To Compile | 856.37 | 813.18 | 788.37 |
| build-apache: Time To Compile | 138.61 | 124.32 | 129.24 |
| ramspeed: Type: Copy - Benchmark: Integer | 8364.29 | 8437.48 | 8724.83 |
| ramspeed: Type: Copy - Benchmark: Floating Point | 8308,15 | 8415.78 | 8509.78 |
| ramspeed: Type: Add - Benchmark: Integer | 9155,55 | 9571.85 | 9497.83 |
| ramspeed: Type: Add - Benchmark: Floating Point | 9165,18 | 9697.62 | 9504.58 |
| aio-stress: Test: Random Write | 751.88 | 263.06 | 21.36 |
| sqlite: Test Target: Default Test Directory | 4.30 | 37.37 | 1228.84 |
| postmark: Disk Transaction Performance | 1271 | 439 | 844 |
| compress-pbzip2: 256MB File Compression | 66.37 | 56.33 | 56.70 |

**Fonte:** autoria própria, 2016.

# 5 CONCLUSÃO

Desde o seu surgimento nos anos 50 no MIT, a virtualização passou por um grande processo de desenvolvimento e aperfeiçoamento durante todo esse tempo. Várias técnicas e métodos foram desenvolvidos a fim de aperfeiçoar o seu desempenho durante o processo de criação e execução de máquinas virtuais. Durante todo esse tempo, várias ferramentas foram desenvolvidas e utilizadas ao longo dos anos, cada ferramenta com um propósito específico de virtualização, como virtualização de servidores, virtualização de *desktops*, virtualização de máquinas de computação em nuvem, entre outros.

O objetivo específico desse projeto foi adotar as ferramentas de virtualização de *desktop* com o objetivo de se obter o resultado de qual a melhor ferramenta existente hoje para usuários comuns.

O resultado esperado era obter uma ferramenta que se sobressaísse em relação às outras em termos de desempenho, e ao analisar o resultado, pode-se concluir que o objetivo foi alcançado, mesmo as ferramentas apresentando um resultado de desempenho muito semelhante entre si durante os testes. O VirtualBox 4.3.36 acabou se sobressaindo na maioria dos testes, obteve os melhores resultados durante os testes de processador e nos testes de memória, trabalhando com soma de dados. Já o Qemu-KVM 2.2.0, obteve o melhor desempenho durante os testes de memória trabalhando com soma de dados junto com o VirtualBox 4.3.36 que obteve o melhor resultado nos testes de memória com o sub teste de cópia de dados. O VMware Workstation Player 12 foi a melhor ferramenta durante os testes

de disco, obtendo os melhores resultados, muito superior aos resultados das outras ferramentas.

Portanto, ao analisar os resultados desse *benchmark* (Figura 35), chega-se à conclusão de que a ferramenta VirtualBox 4.3.36 obteve os melhores resultados, se sobressaindo na maioria dos testes. Assim, pode-se dizer que o objetivo do trabalho foi alcançado, mostrando que o VirtualBox 4.3.36 é a melhor ferramenta para se trabalhar em ambientes *desktop,* seguido pelo Qemu-KVM 2.2.0, e pelo VMware Workstation Player 12, sendo este que obteve os piores resultados.

# REFERÊNCIAS


BARHAM, P. et al. **Xen and the art of Virtualization**. Proceedings of the 19th ACM Symposium on Operating Systems Principles – SOSP 2003, p. 164-177.

CACIATO, L. E. **Alta disponibilidade em serviços essenciais utilizando virtualização**. Dissertação de Mestrado. Universidade Estadual de Campinas. Campinas, 2015.

CANONICAL. **O que é Ubuntu**. 2016. Disponível em: <https://ubuntu-pt.org/content/o-que-%C3%A9-o-ubuntu>. Acesso em: 09 jun. 2016.

CARISSIMI, A. **Virtualização**: Princípios básicos e aplicações. Caxias do Sul, 2009. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/erad/2009/004.pdf>. Acesso em: 17 maio 2016.

CASTRO, A. B. **Máquinas virtuais em ambientes seguros**. Dissertação de Mestrado. Universidade Estadual de Campinas. Campinas, 2006.
Castro, Arthur Bispo de. Máquinas Virtuais em Ambientes Seguros. Dissertação de Mestrado. Universidade Estadual de Campinas. Campinas, 2006.

DEBIAN. **Uma breve história do Debian**. 2016. Disponível em: <https://www.debian.org/doc/manuals/project-history/index.pt.html#contents>. Acesso em: 29 maio 2016.

LAUREANO, M. **Máquinas virtuais e emuladores**: conceitos, técnicas e aplicações. São Paulo: Novatec, 2006.

MASSALINO, F. **Virtualização de servidores e suas principais ferramentas**. Escola superior do Brasil. Vila Velha, 2012.

MATTOS, D. M. F. **Virtualização: VMWare e Xen**. s.d. Disponível em: <http://www.gta.ufrj.br/grad/08_1/virtual/index.html>. Acesso em: 07 maio 2016.

MAZIERO, C. A. **Sistemas Operacionais**: Conceitos e Mecanismos. Universidade Tecnológica Federal do Paraná. Paraná, 2013.

MOTA FILHO, J. E. **Descobrindo o Linux**: entenda o sistema operacional GNU/Linux. 3. ed. São Paulo: Novatec, 2012.

NEMETH, E.; SNYDER, G.; HEIN, T. R. **Manual Completo do Linux**. Tradução de Edson Furmankiewicz, Carlos Chafransky. São Paulo: Person Prentice Hall, 2007.

OLIVEIRA, F. B. **Virtualização de Sistemas Operacionais**. Monografia de Pós-Graduação. Instituto Superior de Tecnologia em Ciências da Computação. Petrópolis, 2007.

OPENBENCHMARKING.ORG. **Phoronix Test Suite Tests**. 2016. Disponível em: <https://openbenchmarking.org/tests/pts>. Acesso em 14 novembro 2016.

ORACLE. **Oracle VM VirtualBox User Manual**. c2016. Disponível em: <http://download.virtualbox.org/virtualbox/5.0.20/UserManual.pdf>. Acesso em 25 maio 2016.

PHORONIX. **Phoronix Test Suite v6.4.0 (Hasvik) User Manual**. 2016. Disponível em: <http://www.phoronix-test-suite.com/?k=downloads>. Acesso em: 10 jun. 2016.

POPEK, G.; GOLDBERG, R. **Formal requirements for virtualizable 3rd generation architectures.** Communications of the ACM, 1974. In: Carissimi, Alexandre. Virtualização: Princípios básicos e aplicações. Caxias do Sul, 2009. p.40.

SIQUEIRA, L. A. **Máquinas Virtuais com VirtualBox**. São Paulo: Linux New Media do Brasil Editora, 2010.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 3 ed. São Paulo: Pearson Prentice Hall, 2009.

VMWARE. **VMWare Workstation 12 Documentation.** 2016. Disponível em: <http://pubs.vmware.com/workstation-12/index.jsp#com.vmware.ICbase/PDF/ic_pdf.html>. Acesso em: 27 maio 2016.