Universidade Estadual de Campinas
Instituto de Computação

Giovane de Morais

Virtualização completa, paravirtualização e virtualização leve: comparativo de desempenho de recursos de computação em nuvem virtualizados com VMware ESXi, QEMU/KVM, Drivers VirtIO e Docker

CAMPINAS
2020

Giovane de Morais

Virtualização completa, paravirtualização e virtualização leve: comparativo de desempenho de recursos de computação em nuvem virtualizados com VMware ESXi, QEMU/KVM, Drivers VirtIO e Docker

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Luiz Fernando Bittencourt

Este exemplar corresponde à versão final da Dissertação defendida por Giovane de Morais e orientada pelo Prof. Dr. Luiz Fernando Bittencourt.

CAMPINAS
2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Morais, Giovane de, 1995-
M792v Virtualização completa, paravirtualização e virtualização leve : comparativo de desempenho de recursos de computação em nuvem virtualizados com VMware ESXi, QEMU/KVM, Drivers VirtIO e Docker / Giovane de Morais. – Campinas, SP : [s.n.], 2020.

Orientador: Luiz Fernando Bittencourt.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Computação em nuvem. 2. Virtualização. I. Bittencourt, Luiz Fernando, 1981-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Full virtualization, paravirtualization and light virtualization : performance comparison of virtualized cloud computing resources with VMware ESXi, QEMU/KVM, VirtIO Drivers and Docker
Palavras-chave em inglês:
Cloud computing
Virtualization
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Luiz Fernando Bittencourt [Orientador]
Júlio Cézar Estrella
Edson Borin
Data de defesa: 18-12-2020
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-8391-9797
- Currículo Lattes do autor: http://lattes.cnpq.br/5186071622126830

Universidade Estadual de Campinas
Instituto de Computação

Giovane de Morais

Virtualização completa, paravirtualização e virtualização leve: comparativo de desempenho de recursos de computação em nuvem virtualizados com VMware ESXi, QEMU/KVM, Drivers VirtIO e Docker

Banca Examinadora:

- Prof. Dr. Luiz Fernando Bittencourt
  Instituto de Computação/UNICAMP
- Prof. Dr. Júlio Cézar Estrella
  Instituto de Ciências Matemáticas e de Computação/USP
- Prof. Dr. Edson Borin
  Instituto de Computação/UNICAMP

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 18 de dezembro de 2020

Dedicatória

Dedico esta obra a todo ser humano que atuou incansavelmente para que a luz da ciência vencesse as trevas da ignorância.

O sucesso é ir de fracasso em fracasso sem perder entusiasmo (Sir Winston Churchill)

Agradecimentos

Agradeço ao Criador pelo dom da vida, pela oportunidade concedida de realizar tal obra.

Agradeço à Universidade Estadual de Campinas e especialmente ao Instituto de Computação, com todo seu corpo profissionais por possibilitar a elaboração deste projeto.

Ao professor Dr. Luiz Fernando Bittencourt, por toda a sua paciência, dedicação incansável e profissionalismo de ponta durante a orientação desta obra, além de toda sensibilidade e empatia concedidas a mim durante meus períodos de dificuldades pessoais.

A minha família, especialmente ao meu avô Antonio Morais por todo apoio, amor, suporte psicológico e financeiro para que eu seguisse com esta pesquisa.

A ProDB, que cedeu-me espaço para análise de desempenho em seus equipamentos e, especialmente a seu arquiteto de cloud computing Gesiel Bernardes, por explanar a problemática e me motivar a descobrir a resposta aos questionamentos levantados.

A ADTsys e funcionários Vladimir Junior, Israel Passos, Vitor Silva, Willian Oliveira, Diogo Carvalho, Reginaldo Takita, Alan Cabral, André Munhoz, Carlos Menezes, Edson Fugió, Germano Menezes, Nicholas Prado, Gislaine Barros e Riller Faconi pelo companheirismo e ensinamentos.

A Jéssica Mateus Silva por acreditar em meu sucesso; por seu carinho, dedicação e companheirismo únicos e incondicionais, cuja presença em minha vida foi intensa o suficiente para que, através de seu exemplo de vida, eu ousasse ter coragem de me propor grandes desafios, fosse capaz de sobrepujar os obstáculos que a vida me impôs e me sentir substancialmente apoiado ao longo dos anos para que a presente obra pudesse não apenas ser concluída, mas servir de prova que tudo se alcança quando alguém acredita em nós.

Aos parceiros de UNICAMP que contribuíram para que eu chegasse neste momento: Alexandre Nogueira, Paulo Lange, Rafael Gazzoni, Ricardo Silva, Vanderlei Filho, Daniela Barbetti, Adilson Silva, Vitor Takami, Ivan Silva, Magali Barcelos, Rachel Paschoalino e Roberto Costa.

Aos amigos Ackley Will, Ana Flávia Santos, Carina Rodrigues, Denis Shiguemoto, Diego Froner, Dr. Romis Attux, Emival Eterno Costa, Evandro Bernardi, Fabiana Rodrigues, Felipe Jesus, Geison Ono, Guilherme Dakuzaku, Ivan Dantas, Jefferson Nemeth, Juliana Oliveira, Leandro Xastre, Leonardo Dultra, Lucas Grecco, Luigi Pradela, Luís José Costa, Marcelo Fonseca, Naegelly Vitória, Orivaldo Rodrigues, Paulo Quitério, Quesia Silva, Rafael Curi, Raissa Peralva, Robson Correia, Vitor Lenso e Zilma Moraes por todo o apoio e ajuda, que muito contribuíram para este trabalho e minha formação.

A Andressa Lopes, meu braço forte e mão amiga, por não apenas acreditar em mim, mas de escolher trilhar sua vida ao lado da minha em amizade e companheirismo, cuja luz de sua presença em minha existência dissipou as trevas da tristeza e da solidão.

Aos meus colegas de curso Guilherme Andrino e Matteus Vargas com quem convivi intensamente durante os últimos anos, por toda ajuda acadêmica e pessoal que recebi deles, pelo companheirismo e pela troca de experiências que me permitiram crescer não só como pessoa, mas também como profissional.

Resumo

A computação em nuvem do inglês *cloud computing* se popularizou nas últimas duas décadas no mercado principalmente pelas ideias de redução dos custos com tecnologia, aumento do desempenho na execução de processos e segurança dos dados processados e armazenados. Devido aos inúmeros requisitos de aplicações a serem atendidos, a computação em nuvem foi dividida em três grandes áreas: Infraestrutura como Serviço (IaaS), Plataforma como Serviço (PaaS) e Software como Serviço (SaaS). O advento da computação em nuvem não seria possível sem o conceito de virtualização. Esta, por sua vez, permite que em um mesmo hardware sejam executados paralelamente vários ambientes distintos e isolados. A virtualização é solução de baixo custo que provê escalabilidade, confiabilidade e isolamento e flexibilidade à recursos computacionais, máquinas virtuais e sistemas em geral. A motivação desta obra é voltada principalmente para uma análise do novo cenário de virtualização do ambiente de computação em nuvem surgido em meados de 2013, impulsionada pelos grandes players de cloud computing mundiais: Amazon, Microsoft e Google, quando este primeiro deixou de utilizar virtualização completa por hardware (bare metal) com o Xen Server, adotando a virtualização completa por software (emulação) com o KVM. Além disso, o advento de novas ferramentas e paradigmas como a virtualização leve por contêineres tornam necessária uma reavaliação do cenário de desempenho de elementos de virtualização em ambientes de computação em nuvem. Observando essa necessidade, este trabalho discute, através de análise quantitativa por ferramentas benchmark, os impactos que os diferentes meios de virtualização produzem sobre o desempenho de máquinas virtuais em ambientes de computação em nuvem IaaS. Para tanto, é analisado o desempenho de processamento e armazenamento nos quatro principais tipos de virtualização na atualidade em computação em nuvem: a virtualização completa por hardware, representa pelo estudo de caso do VMware vSphere, a virtualização completa por software, representada pelo QEMU/KVM, paravirtualização, representada pelo KVM com VirtIO Drivers e a virtualização leve representada pelo Docker. Após coleta de dados como tempo de resposta de processamentos com e sem exigência de memória RAM (armazenamento primário), IOPS e latência de disco (armazenamento secundário), é possível observar que a paravirtualização tem um desempenho superior das demais tecnologias de virtualização em todos os cenários de desempenho de disco e em cenários de processamentos que demandem maiores tempos da CPU e/ou memória RAM. O Docker, por sua vez, é a ferramenta de virtualização mais recomendada para processamentos que utilizem a CPU e a RAM por menor tempo.

Abstract

Cloud Computing has become popular over past few decades in the market mainly by introducing ideas to reduce technology speding, performance increase in processes execution and storage, and processed data security. Due to countless application requirements to be attended, Cloud Computing has been divided into three major areas: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The advent of Cloud Computing would not be possible without he concept of Virtualization. This, in turn, allows that, in the same hardware, several distinct and isolated environments are executed in parallel. Virtualization is a low cost solution that provides scalability, reliability, isolation, and flexibility to computational resources, virtual machines, and systems in general. This work aims to analyze Cloud Computing virtualization scenarios emerged around mid 2013, driven by major global Cloud Computing players, namely Amazon, Microsoft and Google, when the former stopped using the first of them stopped using full hardware virtualization (bare metal) as Xen Server, adopting the full virtualization by software (emulation) as KVM. Besides, the arrival of new tools and paradigms, as lightweight virtualization by containers, makes it necessary to reassess virtualization elements performance in Cloud Computing environments. Observing this need, this work discusses, through quantitative analysis made through benchmark tools, the impacts that different ways of virtualization produce over virtual machines performance in IaaS Cloud Computing environments. Therefore, it is analised the storage and processing performance in four main types of Cloud Computing virtualization nowadays: full hardware virtualization, represented by a VMware vSphere case study; full virtualization by software, represented by QEMU/KVM; Paravirtualization represented by KVM with VirtIO Drivers; and lightweight virtualization, represented by Docker. After data collection of processing response time with and without RAM memory load (primary storage), IOPS and disk latency (secondary storage), it is possible to observe the Paravirtualization has a superior performance over other virtualization technologies in all disk performance and larger CPU and/or memory bounded processes. CPU and/or RAM times scenarios. Docker, in turn, is the most recomended virtualization tool for processing shorter CPU/memory bound processes.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Idealizada há meio século e consolidada há pelo menos duas décadas, a virtualização é uma tecnologia que possibilitou a expansão da computação pelo planeta.Através da abstração de hardware nas denominadas "máquinas virtuais", a virtualização é capaz de prover recursos de computação de forma flexível e facilmente gerenciável. Tais características permitem que os recursos e aplicações computacionais em um ambiente virtualizado sejam logicamente isolados, altamente dinâmicos e passíveis de monitoramento através da quantificação do uso efetivo. Graças ao advento e popularização da virtualização, outro paradigma pôde surgir e tornar a computação como serviço uma realidade: a computação em nuvem. Esta surge como uma contra alternativa à aquisição de equipamentos físicos para a implantação e consolidação de recursos computacionais.

De acordo com Erl, Mahmood e Puttini (2013) a computação em nuvem é um modelo computacional relativamente recente que oferece um modelo de negócios onde pessoas e organizações possam adotar ferramentas tecnológicas sem a necessidade de altos investimentos. Dividida em três modelos (Infraestrutura como Serviço, Plataforma como Serviço e *Software* como Serviço) a computação em nuvem oferece uma forma simples de acessar servidores, estruturas e aplicações para serviços virtuais onde o consumidor somente paga pelo que realmente usa e o único requisito para isto é o cliente estar conectado à *Internet*.

Com a expansão em massa da computação em nuvem, ocorre a necessidade de que esta seja cada vez de melhor desempenho e que os recursos entregues sejam mais eficientes conforme Yarali (2018) descreve. Desta forma, a promoção de alto desempenho por parte das ferramentas de virtualização é um tema amplamente estudado (ERL; MAHMOOD; PUTTINI, 2013).

O problema é investigado pela perspectiva dos métodos de virtualização mais conhecidos: virtualização tradicional baseada em *hardware*, virtualização tradicional baseada em *software*, paravirtualização e virtualização leve. Através de pesquisa analítica com coleta de dados como tempo de resposta em processamentos não paralelizáveis, taxa de escrita e leitura em dispositivo de armazenamento, esta obra tem como objetivo comparar o desempenho dos métodos de virtualização citados anteriormente entre si e com um *hardware* dedicado observando os impactos e limitações que cada modelo de virtualização possui em uma estrutura de nuvem computacional. Desta forma é possível definir para quais paradigmas de virtualização têm desempenho mais adequado para uma computação em nuvem IaaS.

É valido ressaltar que ao longo do tempo, principalmente nos últimos cinco anos, o cenário de computação em nuvem sofreu diversos revezes (advindos principalmente dos principais provedores desta tecnologia do planeta) em que estudos sobre ambientes virtualizados mais eficiente são cada vez mais necessários.

1.1 Organização do Texto

Este trabalho está dividido em 9 capítulos:

- Capitulo 1: Introdução e considerações iniciais;
- Capitulo 2: Referencial conceitual onde assuntos relevantes ao tema da obra são levantados;
- Capitulo 3: Objetivos gerais e específicos do trabalho; justificativa e delimitação dos componentes deste trabalho observando o cenário atual do mercado de computação em nuvem a nível mundial e suas principais mudanças;
- Capitulo 4: Trabalhos relacionados onde serão explanados os principais trabalhos alinhados ao tema desta obra;
- Capitulo 5: Levantamento metodológico da obra, discutindo ferramentas e benchmarks de análise, além da explanação da infraestrutura desenvolvida para tais testes;
- Capitulo 6: Testes de controle com a infraestrutura sem elementos de virtualização como referencial para demais testes;
- Capitulo 7: Aplicação de testes em ambientes virtualizados com respectivas análises de resultados de processamento (CPU e CPU acrescida de memória RAM) e armazenamento secundário (discos HDD e SSD);
- Capitulo 8: Levantamento de horas trabalhadas para cada paradigma de virtualização;
- Capitulo 9: Considerações finais e seção de trabalhos futuros;

Capítulo 2

Referencial Conceitual

Nesta seção são apresentados os conceitos básicos e referencial teórico para esta obra.

2.1 Virtualização

De acordo com Portnoy (2016) o início do processamento em larga escala no período entre 1975-1995 teve como protagonistas os chamados *mainframes*. Tais máquinas eram responsáveis pelo processamento dos grandes volumes de dados que corporações, setores governamentais, universidades, dentre outros, produziam. E, para isso, os *mainframes* precisavam ser grandes, robustos e, consequentemente, eram altamente custosos em questão de capital e mão de obra especializada. Assim sendo, apenas grandes empresas de tecnologia poderiam ter capital e mão de obra especializada o suficiente para adquirir e manter um equipamento deste porte em plena operação.

Portnoy (2016) prossegue afirmando que um grave problema advindo desta tecnologia foi o engessamento do *hardware* e dos *softwares* executados pois os mainframes somente poderiam ser operados com sistemas operacionais e aplicações desenvolvidas estritamente para o uso de um *hardware* específico. Ou seja, em uma situação de *hardware* defasado e com a necessidade de substituição por um equipamento mais moderno todo *software* (sistemas operacionais, aplicações etc.) que se executavam nele deveria ser igualmente substituído e refeito do zero para a nova arquitetura de *hardware* que seria configurada. Este tipo de tecnologia ficou vigente durante aproximadamente duas décadas até que surgiu um novo conceito que prometia ampliar o acesso à tecnologia de processamento de armazenamento de dados em alta escala com custo mais acessível e sem a necessidade de uma equipe grande e profissionais altamente qualificados. Este conceito ficou conhecido como virtualização.

A virtualização surgiu como uma forma de flexibilizar a relação entre *hardware* e *software* retirando forte a dependência que os mainframes possuíam com seus programas. A proposta do ambiente virtualizado era que os softwares a serem executados em um determinado *hardware* não mais teriam acesso direto ao equipamento físico senão à uma visão abstraída do equipamento de forma que as características de estruturais de do equipamento não influenciassem no funcionamento. A virtualização permite emular uma imensa variedade de sistemas operacionais em um mesmo *hardware*, além de garantir flexibilidade

e portabilidade das aplicações de alto nível.

A virtualização é responsável por abstrair e controlar, através de um *software*, todos os dispositivos de um ou mais equipamentos físicos (*hardwares*) como processador, memória, armazenamento, placas de vídeo, placas de rede e outros periféricos Babu et al. (2014). A virtualização concede a um ou mais usuários a ilusão do controle total sobre um ou mais *hardwares.* Dessa forma o *hardware* virtualizado é totalmente independente das aplicações executadas, isolado em relação ao acesso dos usuários, seguro em relação aos dados processados e armazenados devido ao encapsulamento de comandos e altamente escalável comparado à *hardwares* físicos (POTHERI, 2018).

2.1.1 Monitor de Máquina Virtual (*Hypervisor*)

De acordo com Morabito, Kjallman e Komu (2015) o Monitor de Máquina Virtual (sigla em inglês VMM), também conhecido como *hypervisor*, é a camada de *software* alocada entre o *hardware* e o sistema operacional. O VMM é a ferramenta de virtualização cuja função se estende desde a abstração dos recursos físicos, controle de acesso aos dispositivos de *hardware* à estruturação de sistemas operacionais completos com suas respectivas aplicações.

O *hypervisor* é o responsável por transmitir as instruções advindas do sistema operacional ao *hardware* e, igualmente, receber as instruções de "baixo nível"e traduzi-las para o sistema operacional de forma transparente e ajustável às características do sistema operacional. Entretanto, para compreensão da atuação de um VMM sob um sistema operacional, é necessário compreender como um sistema não virtualizado atua.

Em arquitetura de computadores atuais, sem quaisquer ferramentas de virtualização, o sistema operacional e o *hardware* trabalham em conjunto restringindo as atividades de um programa ou aplicação à nível do usuário sob o sistema como um todo. Desta forma, os aplicativos de usuário têm privilégios muito limitados, onde determinadas tarefas só podem ser executadas pelo código do sistema operacional e do *kernel*.

2.1.2 Virtualização Tradicional (Completa)

Conforme Altaher (2017) a virtualização tradicional (completa, segundo Morabito, Kjallman e Komu (2015), Portnoy (2016)) o *hypervisor* tem controle total e amplo ao *hardware* de modo a criar uma réplica virtual completa da(s) máquina(s) física(s) e, por meio desta replica abstrata, realizar alocação de recursos para cada sistema operacional que for incluído de acordo com parâmetros que podem ser previamente definidos ou alocados de forma dinâmica. Neste tipo de virtualização cada sistema operacional é distinto, completo em seus recursos (*kernel, drivers*, aplicações, etc.) e deve ser alocado em uma VM que é responsável pelo isolamento e segurança dos dados. Neste modelo o sistema operacional emite comandos para o que considera ser hardware real, mas, na verdade, os dispositivos de hardware são apenas simulados pelo VMM.

Ainda conforme Altaher (2017) é válido ressaltar que as máquinas virtuais não possuem acesso umas às outras apesar de compartilharem teoricamente os mesmos recursos físicos. Esta característica é denominada encapsulamento e/ou isolamento, essencial em

qualquer ambiente virtualizado para preservar a segurança dos dados processados e armazenados neste tipo de ambiente. Neste tipo de virtualização tanto o equipamento físico quanto o sistema operacional e os programas a serem executados não necessitam de nenhuma alteração de estrutura para realizar o processo de virtualização, cabendo única e exclusivamente ao VMM flexibilizar a comunicação entre *hardware* e *software*(Dong et al., 2014).

Estrutura e Funcionamento

A virtualização completa se adapta aos diferentes sistemas operacionais executados em suas VMs. Para tanto, os *hypervisors* de virtualização completa se utilizam de comandos de *kernel* genéricos, que abrangem a maioria dos sistemas operacionais utilizados. Entretanto, esse uso genérico de recursos compromete o desempenho do VMM.

Conforme explanado anteriormente, em um paradigma de virtualização total o sistema operacional visitante não tem conhecimento de que está sendo executado sobre uma camada de virtualização (Zhang; Dong, 2008). Desta forma, então, todas as instruções executadas pelo sistema operacional na VM devem ser previamente testadas pelo *hypervisor* para que possam ser executadas à nível físico e vice-versa. Essa operação é conhecida como tradução binária, que é uma espécie de recompilação binária na qual sequências de parâmetros do *hardware* para o sistema operacional ou vice-versa são traduzidas a partir de um conjunto de instruções de origem para o conjunto de instruções de destino (CHIERICI; VERALDI, 2010).

Na virtualização tradicional, conforme figura 2.2, o VMM ocupa o anel 0, sendo ele o elemento de maior prioridade no envio e recebimentos de comandos do sistema operacional. Este, por sua vez, recebe as instruções vindas direto do sistema operacional, realizando a tradução binária de todos os comandos e requisições dos demais anéis para o nível físico e vice-versa. Tal procedimento gera um enorme consumo de tempo de resposta devido à estrutura de virtualização receber os comandos de todas as camadas acima para a tradução binária e repasse ao *hardware*.

Categorias

A virtualização tradicional pode ser segregada em dois tipos: as baseadas em *hardware* e as baseadas em *software* (CHIERICI; VERALDI, 2010), que serão abordadas no tópico subsequente.

Virtualização Tradicional Baseada Em Hardware - Bare Metal

Altaher (2017) afirma que a virtualização tradicional baseada em *hardware* (popularmente conhecida virtualização bare metal, nativa ou *unhosted*) ocorre quando o *hypervisor* é executado diretamente sobre o *hardware* a ser virtualizado. Neste tipo de estrutura o VMM possui acesso completo a todas as instâncias do equipamento físico em questão, conforme figura 2.3.

De acordo com Altaher (2017) e Fayyad, Perneel e Timmerman (2013) os *hypervisors* bare metal mais populares são: VMware ESX (VMware, Inc.), Xen e Xen Server (Citrix

Systems), Hyper-V e Hyper-V Server (Microsoft Corporation).

Virtualização Tradicional Baseada Em Software - Hosted

Já na virtualização tradicional baseada em *software* (popularmente conhecida virtualização *hosted*), conforme figura 2.4, o VMM precisa ser executado sobre um sistema operacional hospedeiro para conseguir acesso ao *hardware*. Neste tipo de virtualização todos os recursos computacionais físicos também são abstraídos, porém com desempenho um pouco menor do que a virtualização tradicional por *hardware*, uma vez que a existência de um sistema operacional para a execução do *hypervisor* causa um atraso na comunicação entre o VMM e os recursos físicos além de exigir processamento para o sistema operacional que está hospedando o VMM (ALTAHER, 2017).

Conforme Altaher (2017) os *hypervisors* hosted mais populares são: VirtualBox (Oracle Corporation),VMware Workstation (VMware, Inc.), QEMU (Fabrice Bellard), oVirt (Red Hat, Inc.) e Kernel-based Virtual Machine (KVM)

2.1.3 Taxonomia de Virtualização

Existe uma ampla gama de Máquinas Virtuais (VM, do inglês *Virtual Machine*), com vários propósitos e implementações. Como forma de organizar as características e questões comuns de elaboração, os autores Smith e Nair (2005) apresentam uma taxonomia de VMs. Essa taxonomia deve lidar com a descrição, identificação e classificação dos métodos de virtualização, individualmente ou em grupo.

As VMs são divididas em dois grandes tipos: VMs de processo e de sistema. No primeiro tipo, a VM oferece suporte a uma ABI—*Application Binary Interface*— que inclui recursos do conjunto de instruções de hardware e do sistema operacional; no segundo tipo, a VM oferece suporte a um ISA—*Instruction Set Architecture*—completo. O ISA descreve o projeto de um processador em termos das operações básicas que ele deve suportar (TANENBAUM, 2016). O ISA não se preocupa com os detalhes específicos de implementação de um processador. O ISA preocupa-se apenas com o conjunto de operações básicas que o processador deve suportar. Por exemplo, processadores x86 e ARM têm implementações totalmente diferentes, mas suportam o mesmo conjunto de operações básicas.

As subdivisões da taxonomia são baseadas em se o convidado e o host usam o mesmo ISA, tanto para as VMs de processo quanto as de sistema. Nas VMs de processo, estão as VMs onde os conjuntos de instruções do host e do convidado são iguais. Há dois exemplos dessa situação. O primeiro são os sistemas multiprogramados onde um programa pode ser lido na memória principal, executado por algum tempo, gravado no disco e, em seguida, lido de volta na memória principal para ser executado novamente (TANENBAUM, 2016). O segundo são os otimizadores binários dinâmicos do mesmo ISA, que transformam as instruções do convidado apenas otimizando-as e, em seguida, as executam nativamente (SMITH; NAIR, 2005).

Ainda nas VMs de processo, há dois exemplos onde os ISA's de convidado e host são diferentes. Estes são os tradutores dinâmicos e VMs HLL (High-Level Language). VM HLL são aquelas que se concentram em minimizar recursos específicos do hardware e do

sistema operacional, pois são independentes de plataforma como, por exemplo, o Java Virtual Machine (JVM) (SMITH; NAIR, 2005). Já a tradução dinâmica é usada para acelerar a execução de programas de código de bytes. A VM compila seus bytecodes em código de máquina (hardware) e o código traduzido também é colocado em um cache, para que da próxima vez o código de máquina dessa unidade possa ser executado imediatamente, sem repetir a tradução (DEUTSCH; SCHIFFMAN, 1984).

Sobre as VMs do sistema, caso o convidado e o host usem o mesmo ISA, então os exemplos são as VMs de sistema tradicional e VMs hospedadas. Em ambas VMs, o objetivo é fornecer ambientes de sistema replicados e isolados. A principal diferença entre VMs clássicas e hospedadas é a implementação do VMM, em vez da função fornecida ao usuário. Exemplos de VMs de sistema em que os ISA's de convidado e host são diferentes incluem VMs de todo o sistema e VMs codificadas.

VMs de todo o sistema são aquelas onde um sistema de software completo, tanto sistema operacional quanto aplicativos, são suportados em um sistema host com suporte a um ISA e sistema operacional diferente (SMITH; NAIR, 2005). Já as VMs codificadas são projetadas para permitir ISA's inovadores e/ou implementações de hardware (SMITH; NAIR, 2005). Com VMs de sistema completo, o desempenho costuma ser de importância secundária em comparação com a funcionalidade precisa, enquanto com VMs codificadas, desempenho (e eficiência de energia) costumam ser os principais objetivos.

2.1.4 Interpretação e Tradução

De acordo com os autores Smith e Nair (2005), antes de definir interpretação e tradução, é necessário definir emulação. Os próprios autores definem emulação como o processo de implementação da interface e funcionalidade de um sistema ou subsistema em um sistema ou subsistema com interface e funcionalidade diferentes. A emulação do conjunto de instruções é um aspecto chave de muitas implementações de máquina virtual porque a máquina virtual deve suportar um programa binário compilado para um conjunto de instruções diferente daquele implementado pelo(s) processador(es) host. A emulação permite que uma máquina implementando um conjunto de instruções, o conjunto de instruções alvo, reproduza o comportamento de software compilado para outro conjunto de instruções, o conjunto de instruções de origem.

As técnicas de emulação de instrução podem ser aplicadas a cada tipo de VM. Uma dessas técnicas é a otimização binária dinâmica do mesmo ISA, em que um binário de programa é otimizado em tempo de execução na mesma plataforma para a qual foi originalmente compilado. Outro exemplo são as VMs de sistema em que o VMM deve controlar e gerenciar a execução de certas instruções de sistema operacional convidado privilegiado, e técnicas semelhantes a emulação são usadas para esse propósito.

A emulação do conjunto de instruções pode ser realizada usando uma variedade de métodos que requerem diferentes quantidades de recursos de computação e oferecem diferentes características de desempenho e portabilidade. Em uma extremidade do espectro está a técnica direta de interpretação, enquanto na outra está a tradução binária.

A interpretação envolve um ciclo de busca de uma instrução de origem, analisando-a, realizando a operação necessária e, em seguida, buscando a próxima instrução de origem—

tudo em software. Tradução binária, por outro lado, tenta amortizar os custos de busca e análise, traduzindo um bloco de instruções de origem em um bloco de instruções de destino e salvando o código traduzido para uso repetido (JOURDAN et al., 1990). Em contraste com a interpretação, a tradução binária tem um custo de tradução inicial maior, mas um custo de execução menor.

A escolha de um ou outro depende do número de vezes que se espera que um bloco de código-fonte seja executado pelo software convidado. Previsivelmente, existem técnicas que se encontram entre esses extremos. Por exemplo, a *Direct Threaded* elimina o *loop* do interpretador correspondente ao ciclo mencionado anteriormente, e a eficiência pode ser aumentada ainda mais pela pré-codificação das instruções de origem em uma forma intermediária interpretável de forma mais eficiente.

Em suma, dentre as formas de interpretação, temos:

1. *Decode-and-dispatch*: o mais simples. Ele opera percorrendo o programa de origem, instrução por instrução, lendo e modificando o estado de origem de acordo com a instrução;

2. *Indirect Threaded*: usa ponteiros para locais que, por sua vez, apontam para o código de máquina. O *Indirect Threaded* pode ser seguido por operandos que são armazenados no "bloco" indireto em vez de armazená-los repetidamente na *thread*. Portanto, o código indireto geralmente é mais compacto do que o código encadeado direto. A indireção normalmente o torna mais lento, embora geralmente ainda mais rápido do que a interpretação *Decode-and-dispatch*;

3. *Direct Threaded with Pre-Decoding*: Inicialmente, a pré-codificação é feita. Isso envolve analisar as instruções e as coloca em campos facilmente acessíveis. Posteriormente, o endereço da rotina do interpretador é carregado de um campo no código intermediário, e um salto indireto do registrador vai diretamente para a rotina;

2.1.5 Paravirtualização

Lingeswaran (2017) afirma que a paravirtualização é um paradigma distinto à virtualização tradicional, uma vez que a mesma não utiliza o conceito de abstração completa dos recursos do *hardware*. Neste tipo de virtualização o *hypervisor* modifica a forma e acesso à recursos específicos através de *drivers* aumentando, assim, a velocidade de acesso aos recursos (Walters et al., 2008).

Estrutura e Funcionamento

Neste paradigma, o sistema operacional precisa ser modificado para o *kernel* realizar as chamadas do VMM apenas quando executar uma instrução que possa alterar o estado do sistema. Fora esta exceção, o *hypervisor* de paravirtualização não testa todas as instruções, o que gera uma diminuição nos tempos de resposta dos comandos emitidos e, consequentemente, um aumento no desempenho Babu et al. (2014). As demais instruções não são lidas pelo VMM, mas sim interpretadas e enviadas ao nível físico diretamente

via *drivers* previamente instalados na própria VM com modificação do *kernel* do sistema operacional para aceitação destes (Walters et al., 2008).

A paravirtualização não modifica o modelo de priorização de processos de um sistema operacional (FAYYAD; PERNEEL; TIMMERMAN, 2013). O *kernel* do sistema operacional continua na posição zero, enquanto o *hypervisor* é executado fora deste modelo, em modo de super usuário, para recebimento e conversão das instruções sensíveis (aquelas que são passíveis de modificação do sistema operacional) (LINGESWARAN, 2017). O restante das instruções são enviadas pelos *drivers* de paravirtualização diretamente ao *hardware* sem a necessidade do processo de tradução binária.

Projetada para aprimorar o desempenho do sistema e minimizar a sobrecarga, mitigando o processo de subutilização de VMs através da eliminação do uso de *drivers* genéricos que inibem o uso da capacidade total do dispositivo em paradigmas como da virtualização completa, a paravirtualização é considerada como o método de virtualização mais eficiente em relação à desempenho pois não necessita de um *hardware* virtual abstraído para comunicação de um recurso específico, porém menos flexível que os métodos citados anteriormente devido as alterações que o sistema operacional precisa sofrer. (ALTAHER, 2017).

Outro ponto importante a ser considerado é a desvantagem de custo de manutenção do sistema operacional devido às modificações inseridas para a paravirtualização (Zhang; Dong, 2008). Desta forma, a paravirtualização é melhor caracterizada como um recurso adicional a um virtualizador emulado ou *bare metal* geralmente na forma de *drivers* ou de uma ferramenta adicional. Assim sendo, não é possível encontrar um virtualizador inteiramente paravirtualizado sem que o mesmo seja ancorado em uma estrutura de virtualização tradicional (LINGESWARAN, 2017) (ALTAHER, 2017).

Os maiores representantes da paravirtualização no mercado até o momento de escrita desta obra são:

- VirtIO Drivers: embutido diretamente no *kernel* do KVM, essa ferramenta é a responsável por possibilitar que o virtualizador realize a paravirtualização de recursos que tenham *inputs* e *outputs* de dados como placas de rede, armazenamento, entradas seriais, USB e periféricos em geral de uma VM (ALTAHER, 2017).

- Os drivers nativos de paravirtualização do Xen e Xen Server (Citrix Systems), porém somente ativos quando configurados (ALTAHER, 2017).

- Alguns componentes nativos das Ferramentas de Convidado da VMware ESX (VMware, Inc.), porém somente ativos quando configurados (ALTAHER, 2017).

2.1.6 Virtualização Leve

A denominada virtualização leve, ou virtualização baseada em contêineres, fornece um nível diferente de abstração se comparada à tradicional. Essa categoria de virtualização pode construir uma réplica virtual completa das máquinas físicas, abstraindo todos os recursos disponíveis, no entanto a separação das aplicações não ocorre a nível de sistema operacional como no processo de virtualização tradicional, mas sim a nível de aplicação

e/ou programas a serem executados. A virtualização leve executa um ou mais processos totalmente isolados entre si sobre o mesmo *kernel* do sistema operacional hospedeiro virtualizado em um servidor físico. O exemplo mais comum desta tecnologia é o contêiner.

Contêiner pode ser considerado um ambiente virtual geralmente pequeno, porém totalmente encapsulado e isolado, mesmo inserido juntamente com outros em um mesmo sistema operacional, no qual inclui um conjunto de dependências específicas necessárias para executar determinada aplicação.

O Docker surgiu com base no conceito de virtualização, originalmente pensado por Solomon Hykes em 2013. Teve como intuito primário ofertar suporte ao *dotCloud* em seu gerenciamento de serviços, atuando como facilitador de *deploys* das aplicações, onde seu processamento ocorreria em um container Linux, sem a necessidade de VMs que demandassem alto volume de armazenamento .

Logo após sua ascensão, tornou-se uma ferramenta de código aberto para que vários desenvolvedores pudessem auxiliar o projeto. Há uma gama de procedimentos necessários para se colocar em prática até que um software esteja apto ao ambiente de produção. O Docker emerge justamente em função de prover aplicações de uma maneira mais rápida que o usual, utilizando a técnica de encapsulamento que engloba a operação de criar um objeto subdividido que atenda aos níveis de quaisquer ambientes, racionalize a quantidade de serviço e a incumbência da corporação que cuida de softwares ágeis.

O escopo do Docker apresenta vários benefícios tais como: uso sútil de especificações, se porta de maneira diferente realizando o isolamento de níveis processuais utilizando o *kernel* da máquina, ao invés de todo sistema operacional virtualizado. A portabilidade permite que os pacotes sejam executados em todos os *hosts dockers*. O *host* permanece alheio a tudo que ocorre no contêiner devido a previsibilidade das interfaces que seguem um padrão e mantém uma comunicação previsível. Disponibilidade de uma *cloud* pública que pode ser utilizada para criação dos ambientes de acordo com a necessidade pessoal, tudo isso seguindo um padrão de configurações (Raho et al., 2015).

Uma característica de seu funcionamento é o empacotamento de aplicações em contêineres, este pacote se transforma em imagem Docker que pode ser executada em plataformas distintas, se a reprodução funcionar em um ambiente menor como um simples computador, logo poderá ser executada em um servidor sem nenhum empecilho. Os Linux Contêineres (LXC) auxiliam na adição de serviços disponíveis e simplificação de seu uso, atuam para isolar procedimentos e programas, é semelhante a um local virtualizado, entretanto é bem leve e interligado ao *host* (Raho et al., 2015).

Os LXC isolam os contêineres virtualizados em níveis de conexão, processamentos e armazenamentos.Este modelo distribuído disponibiliza flexibilização para diversos ambientes estarem juntos ao *host* sem nenhuma falha. Uma característica positiva deste cenário é a separação de processos por contêiner, onde cada um possui uma ação específica e contém todos os itens imprescindíveis de configuração, assim podem ser executados separadamente.

Os moldes para isolar no ambiente Docker consistem em virtualizar o sistema operacional, nesta virtualização é concedido a permissão da execução de vários processos simultâneos sendo processados de forma isolada a um único *host*. Além disso, o LXC possibilita a criação de *namespaces* para as partes fundamentais e quando são isoladas

com o cgroups responsável por subdividir o processamento das requisições.

Nas infraestruturas virtualizadas, as VMs efetuam duas réplicas de maneira integral de *hardware* e *software* do *host*, uma do local físico e a outra para o ambiente que será virtualizado. Em função disso, as máquinas virtuais perdem um pouco em desempenho se comparado aos contêineres. O Docker atua para encaixar as configurações primordiais de um sistema operacional, para que ele possa ser executado sem a necessidade da criação de outro servidor distinto, funciona mediante a três características: contêineres, registros e imagens.

2.2 Computação em Nuvem

No ano de 1961 o pesquisador John McCarthy propôs uma alternativa ao modelo de implantação de soluções físicas para atender as necessidades de processamento e armazenamento tanto de empresas de grande porte quanto de usuários comuns. McCarthy imaginava que a computação poderia ser oferecida como um serviço a todos de forma democrática e não como um produto fruto da compra de grandes e caros equipamentos, onde o usuário apenas pagasse pelo o que se usa de fato, surgindo neste momento o conceito que mais tarde seria cunhado de *computation-as-a-service* (CaaS), em português computação como serviço (YARALI, 2018).

Conforme Yarali (2018) McCarthy também abordou temas como o uso compartilhado do computador e de seus recursos, de forma simultânea, por um ou mais usuários interconectados. Desta forma a nuvem computacional poderia trazer flexibilidade, segurança e robustez no desempenho do *hardware* e *software* (Armstrong; Djemame, 2011). Apesar de ser um tema muito presente na atualidade, a implantação da CaaS dependia de tecnologias que não eram amadurecidos no tempo de McCarthy como a virtualização de *hardware* e interconexão de dispositivos em uma rede de alta escala. Foi apenas em meados do novo milênio que a tecnologia proposta por McCarthy começou a ser utilizada em larga escala. Em 2006 a ideia da CaaS foi trazida novamente à tona quando o termo computação em nuvem foi estabelecido pelo palestrante Eric Schmidt, da Google, onde explanava como a empresa gerenciava os recursos de seus *datacenters*.

Utilizando diretamente a virtualização de *hardware*, citada na seção anterior, como ferramenta de base para sua infraestrutura, a computação em nuvem tem como objetivo prover um ambiente robusto para realizar atividades de processamento, armazenamento e compartilhamento de dados que de forma convencional não ocorreriam (Dong et al., 2014). Pode-se afirmar que a computação em nuvem transfere a responsabilidade de processamento e armazenamento dos dados para servidores maiores e mais robustos hospedados em *datacenters* interconectados via internet. Desta forma a contratante não tem custos de manutenção e provisionamento de máquinas e ambiente de computação. E, conforme explanado anteriormente, se tratando de CaaS, o pagamento varia conforme a utilização, o que auxilia na redução dos custos com operação (ERL; MAHMOOD; PUTTINI, 2013) (YARALI, 2018).

Em uma nuvem computacional os dispositivos finais não possuem nenhum alto poder de processamento ou armazenamento, bastando estarem conectados à nuvem para en-

viar e receber os dados que precisam ser processados. Portanto nuvem computacional é um modelo que possibilita acesso, de modo conveniente e sob demanda, a um conglomerado de recursos computacionais configuráveis como: redes, servidores, armazenamento, aplicações e serviços, que podem ser rapidamente adquiridos e liberados com mínimo esforço gerencial ou interação com o provedor de serviços, diminuindo custos e provendo escalabilidade (YARALI, 2018).

2.2.1 Características Essenciais

Para que um ambiente de dados seja classificado como computação em nuvem, o mesmo deve atender à algumas especificações e possuir certas funcionalidades como requisitos básicos de sua estrutura e funcionamento (ERL; MAHMOOD; PUTTINI, 2013) (YARALI, 2018), conforme as características a seguir.

Self-service sob demanda

A primeira funcionalidade essencial de uma nuvem computacional diz a respeito da possibilidade de o cliente adquirir os serviços, sem a necessidade de interação humana com os provedores. Essa funcionalidade é denominada pelo NIST como sendo uma espécie de self-service sob demanda e, por meio dela, o cliente pode obter uma quantidade considerada infinita de recursos computacionais, tais como poder de processamento, capacidade de armazenamento na medida em que necessite (ERL; MAHMOOD; PUTTINI, 2013).

Na estrutura interna de uma nuvem, o *hardware* e o *software* podem ser reconfigurados de forma automática, personalizada e transparente para cada perfil de usuário. Assim cada indivíduo que utiliza uma fração da nuvem pode personalizar as características de seu ambiente computacional de forma única, como por exemplo, a declaração de determinados privilégios de redes, instalação de algum *software* ou definição do sistema operacional a ser utilizado (ERL; MAHMOOD; PUTTINI, 2013).

Amplo acesso à rede

Erl, Mahmood e Puttini (2013) Afirma que a nuvem computacional deve ser capaz de disponibilizar recursos através da rede e possibilitar que estes sejam acessados de qualquer lugar, hora e por quaisquer tipos de plataformas, como por exemplo: *smartphones, notebooks, tablets, Chromebook, netbooks, smart tvs* etc. Segundo mesmo autor o tipo de interface de acesso à nuvem não deve ser um impeditivo aos usuários. A única exigência quanto a este ponto é que o dispositivo tenha conexão com a Internet (YARALI, 2018).

Conjunto de recursos

Os recursos de computação de uma nuvem são agrupados logicamente para atender diversos tipos de usuários com diferentes recursos físicos e virtuais atribuídos dinamicamente conforme a demanda do consumidor (Armstrong; Djemame, 2011). O cliente/usuário não exerce nenhum controle ou conhecimento sobre a localização exata dos recursos disponibilizados (YARALI, 2018).

Geralmente os recursos de uma nuvem, dependendo da sua amplitude, podem estar espalhados fisicamente ao redor do globo, interligados apenas pela rede, porém à nível virtual, os usuários enxergam toda a gama de recursos em um nível mais alto de abstração, como se todo o conjunto de recursos estivesse alocado no mesmo lugar. Os exemplos de recursos podem incluir: armazenamento, processamento, memória, largura de banda de rede, número de máquinas virtuais etc. (ERL; MAHMOOD; PUTTINI, 2013).

Elasticidade rápida

Um outro atributo importante da nuvem é sua capacidade de adaptação as necessidades dos seus usuários. Recursos podem ser alocados ou desalojados de forma rápida e elástica, em alguns casos até automaticamente de acordo com a demanda. Para os usuários, os recursos disponíveis na nuvem aparentam ser infinitos, podendo ser adquiridos em qualquer quantidade e a qualquer momento (YARALI, 2018).

A elasticidade em nuvem só é possível graças a virtualização, que é capaz de criar várias instâncias de recursos requisitados pelo cliente utilizando um mesmo recurso físico. Virtualização é a criação de ambientes virtuais com o propósito de abstrair características físicas do *hardware*, podendo emular vários sistemas operacionais em uma única plataforma computacional (ERL; MAHMOOD; PUTTINI, 2013).

Serviço medido

Todo sistema de nuvem computacional, em algum nível de abstração, realiza quantificação dos recursos desprendidos para o tipo de serviço fornecidos pela mesma (armazenamento, processamento, largura de banda, contas de usuários etc.). O uso de recursos pode ser monitorado, controlado e relatado para os fornecedores e os consumidores do serviço utilizado pela nuvem, garantindo assim transparência e qualidade para o cliente e o provedor (ERL; MAHMOOD; PUTTINI, 2013).

2.2.2 Modelos de Implantação

Os modelos de implantação da computação em nuvem podem ser divididos em: privado (restrito à apenas uma pessoa ou grupo específico), público (aberto a qualquer indivíduo que queira acessar a nuvem), comunidade (conjunto de uma ou mais nuvens computacionais que trocam informações entre si) e híbrido (nuvens com estrutura parcialmente públicas e privadas) (ERL; MAHMOOD; PUTTINI, 2013).

2.2.3 Modelos de Serviço

Conforme observado anteriormente, a computação em nuvem distribui os recursos na forma de serviços aos consumidores. Baseado nisso é possível dividir a computação em nuvem em três modelos específicos com base nos serviços oferecidos: o *Software* como Serviço (SaaS), Plataforma como Serviço (PaaS) e Infraestrutura como Serviço (IaaS). Estes três modelos de serviços de nuvem computacional são protagonizados respectivamente por três agentes: o provedor (atua em IaaS), o desenvolvedor (atua em PaaS) e o usuário final

(atua em SaaS). Para efeitos de explicação, define-se "provedor" como aquele que disponibiliza, gerencia e supervisiona toda a infraestrutura de computação em nuvem (ERL; MAHMOOD; PUTTINI, 2013) (YARALI, 2018).

Software as a Service (SaaS)

Segundo Kavis (2014) o *Software* como Serviço (SaaS) representa os serviços de mais alto nível ofertado por uma nuvem. O SaaS permite aos usuários conectar-se e utilizar aplicativos completos baseados em computação em nuvem. Estas aplicações são acessíveis de qualquer dispositivo do cliente através de uma relação do *thin client* (computador cliente de uma rede que possui uma personificação cliente-servidor) como, por exemplo, um navegador Web. Os prestadores de serviços disponibilizam o SaaS sempre na camada de aplicação, o que leva a executar os serviços inteiramente na nuvem podendo ser uma alternativa a executar um programa em uma máquina local. Desta forma o SaaS traz a redução de custos, dispensando a aquisição de licença de *softwares*. Pode se citar exemplos de SaaS como os sistemas de banco de dados, e-mail, calendário, lojas de aplicativos online, processadores de textos etc.

Platform as a Service (PaaS)

Conforme Kavis (2014) a Plataforma como Serviço (PaaS) se objetiva em servir como base para o desenvolvimento de aplicações destinadas aos usuários de uma nuvem, criando uma plataforma que agiliza esse processo e possibilita a manutenção da infraestrutura que mantém essas aplicações. O PaaS oferece uma infraestrutura intermediária (operando entre o *hardware* e o *software* de alto nível) para implementar e testar aplicações na nuvem. No PaaS o consumidor não controla a infraestrutura da nuvem, entretanto pode controlar os aplicativos utilizados no sistema e até mesmo hospedagem de aplicativos e configurações de ambiente. Pode-se citar alguns exemplos de PaaS como o Windows Azure Cloud, IBM Bluemix, Amazon Web Services (AWS), dentre outros.

Infraestructure as a Service (IaaS)

De acordo com Kavis (2014) a Infraestrutura como Serviço (IaaS) traz os serviços oferecidos diretamente na camada de infraestrutura para o usuário final, nestes serviços se pode citar servidores, roteadores, rede, *firewalls*, *storages*, núcleos de processamento e outros recursos de computação virtualizados pela nuvem. Baseado em técnicas de virtualização de recursos de computação conforme citados anteriormente, o IaaS é a base de toda infraestrutura necessária de suporte para a SaaS e o PaaS. Esse modelo de serviço se objetiva em prover uma interface única para administração da infraestrutura, a aplicação API (*Application Programming Interface*) para interação com *hosts, switches*, roteadores e o suporte para a adicionar novos equipamentos de forma simples e transparente e barata (pois não será necessário a aquisição de novos servidores e dispositivos de rede para a expansão de serviços). Pode-se citar como exemplos de IaaS: Amazon EC2, Hostgator, Xen Cloud Platform etc.

O modelo IaaS para computação em nuvem é a base para o PaaS e o SaaS, uma vez que sem este modelo, os dois anteriores não podem existir. Desta forma, para obter a gerência e o controle de recursos em ambiente IaaS é necessário o uso de um orquestrador. A orquestração de computação em nuvem utiliza ferramentas de automatização, coordenação, monitoramento e gerenciamento da entrega da infraestrutura e dos recursos computacionais. Alguns exemplos de orquestradores IaaS são: OpenQRM, OpenNebula, OpenStack, CloudStack, CloudHit etc (KAVIS, 2014).

2.2.4 IaaS, PaaS e SaaS: Uma Visão Holística

Apesar de serem apresentados de formas distintas, os três modelos de serviço de computação em nuvem são interligados entre si. Em uma visão *top-down*, a abstração mais complexa é o SaaS que, por sua vez, se ancora do PaaS que, por sua vez é ancorado no IaaS. Assim, é impossível desvinculá-los entre si. Cada modelo de serviço de computação em nuvem recebe todas as características intrínsecas ao modelo que está ancorado. Desta forma é possível observar a seguinte dinâmica: o modelo SaaS recebe as características estruturais como desempenho e acesso à recursos do PaaS e do IaaS. Por sua vez, o PaaS recebe as características do IaaS. Desta forma, é possível observar que o desempenho em um ambiente IaaS vai impactar fortemente o desempenho do PaaS e do SaaS (WATTS; RAZA, 2019).

É valido ressaltar que as ofertas de serviços de IaaS criam uma base para as ofertas serviços em PaaS e, consequentemente, as ofertas de PaaS fazem similar para as ofertas de SaaS. Entretanto, para manter um passo à frente da concorrência, é comum observar que provedores utilizam partes de cada um dos três modelos, cruzando as fronteiras delimitadas pela convenção dos conceitos IaaS, PaaS e SaaS.

Capítulo 3

Objetivos

Nesta seção são apresentados os propósitos e metas para esta obra.

3.1 Objetivo Geral

Segundo Gordon et al. (2012), McDougall e Anderson (2010), Xavier et al. (2013) o processamento e dispositivos de armazenamento são fortemente responsáveis pelo desempenho de uma VM (virtual machine - sigla em inglês para máquina virtual), o presente trabalho consiste em análise quantitativa de dados obtidos através de *benchmarks* que representem uma carga típica de uma estrutura de nuvem computacional IaaS com foco em analisar isoladamente o uso de recursos como processamento (CPU e RAM) e armazenamento secundário (disco) entre os principais paradigmas de virtualização da atualidade.

3.2 Objetivos Específicos

Esta obra tem como objetivos específicos:

- Atualizar o estado da arte dos dispositivos físicos de alocação de recursos virtualizados utilizando *hardwares* voltados à computação em nuvem de alto desempenho;
- Atualização do cenário de virtualização em computação em nuvem através da criação de infraestruturas de alto desempenho com os principais paradigmas de virtualização e seus respectivos virtualizadores do mercado atual: completa baseada em *hardware* - bare metal (VMware), completa baseada em software - emulação (QEMU/KVM), paravirtualização (KVM com VirtIO Drivers) e virtualização leve - contêiner (Docker);
- Conforme pontuado por Xavier et al. (2013), aperfeiçoar a segregação de recursos computacionais virtualizados em testes com o objetivo de melhorar sua análise;
- Através dos testes de desempenho de cargas internas e pontuais, observar cenários nos quais cada modelo de virtualização seria adequado para ser utilizado em nuvens IaaS;

3.3 Justificativa e Delimitação

Segundo Yarali (2018) um dos problemas do escopo de computação em nuvem na atualidade é o desempenho dos recursos entregues por VMs em comparação com máquinas físicas. Já é conhecido que o desempenho entregue por VMs é menor que o desempenho do que o mesmo servidor físico entregaria diretamente. Desta forma, a cada ano, as provedoras adotam tecnologias de virtualização de melhor desempenho diante da crescente demanda mundial.

Conforme observado no capítulo anterior, o cenário de computação em nuvem mundial a nível de virtualização passou por evoluções ao longo do tempo motivada, principalmente, por dois grandes fatores: a adoção de métodos (emulação e paravirtualização) e virtualizadores (KVM) mais eficientes em detrimento do modo tradicional de virtualização (bare metal) e virtualizadores mais consolidados (Xen) pela AWS e o advento da utilização de um modelo denominado virtualização leve através de contêineres (FORBES BRASIL, 2019). Tais fatores foram de extrema importância para a modificação do modo como os grandes provedores de nuvem computacional IaaS entregam seus serviços ao público.

Diante deste novo cenário, do advento da virtualização por contêineres, da evolução de *hardware* que naturalmente ocorre ao longo do tempo e da redução de obras redigidas diante da problemática citada dos últimos três anos, abre-se uma janela de oportunidade de refatoração dos testes e *benchmarks* citados no penúltimo capítulo com o objetivo de atualização de dados de desempenho de nuvem computacional.

Esta obra delimita-se a uma análise quantitativa de dados obtidos pelos *benchmarks* de desempenho realizados em uma mesma estrutura de computação em nuvem, diferenciados apenas pelos paradigmas de virtualização citados anteriormente, compará-los entre si e com o mesmo *hardware* dedicado. Não pertence ao escopo desta obra comparar orquestradores de computação em nuvem ou arquiteturas de equipamentos físicos distintos.

3.3.1 Cenários do Mercado de Computação em Nuvem

A utilização de computação em nuvem em escala comercial se deu principalmente por influência de três grandes provedores à nível mundial: Amazon Web Services (AWS), Microsoft Azure e Google Cloud Platform (GCP). Segue um breve histórico sobre estes.

Segundo Narula, Jain e Prachi (2015) a Amazon Web Services (AWS) possui um terço do mercado de nuvem computacional mundial da atualidade, a AWS disponibilizou serviços em nuvem IaaS em 2006, evoluindo para o Elastic Cloud Computing (EC2), ofertando máquinas virtuais baseada no virtualizador Xen Server.

(Kotas; Naughton; Imam, 2018) afirma que, com pouco mais de um décimo do mercado de nuvem computacional mundial da atualidade, a Microsoft Azure disponibilizou serviços em nuvem IaaS em 2010, possuindo o serviço de IaaS mais simples dos três. Com uma baixa variabilidade de tipos de instâncias que podem ser criadas, a Azure tem como foco sistemas operacionais Windows Server virtualizados. Desde sua concepção até os dias atuais utiliza como principal virtualizador o Hyper-V Gusev et al. (2014).

Hunter e Porter (2018) explica que com pouco menos de um décimo do mercado de nuvem computacional mundial da atualidade, a Google Cloud Platform GCP disponibili-

zou serviços em nuvem IaaS em 2013, possuindo o serviço de IaaS menos popular dos três. Apesar da margem pequena de clientes em comparação dos demais provedores, os números de utilização do provedor da Google aumentaram exponencialmente nos últimos anos. Pouco se sabe dos virtualizadores que existem no grupo de recursos da GCP. Conforme Mitchell e Zunnurhain (2019), o principal é o KVM.

Comparativo de Mercado

FORBES BRASIL (2019) indica a fatia de mercado proporcional a cada provedor com dados que variam de 2015 a 2018, conforme figura 3.1.

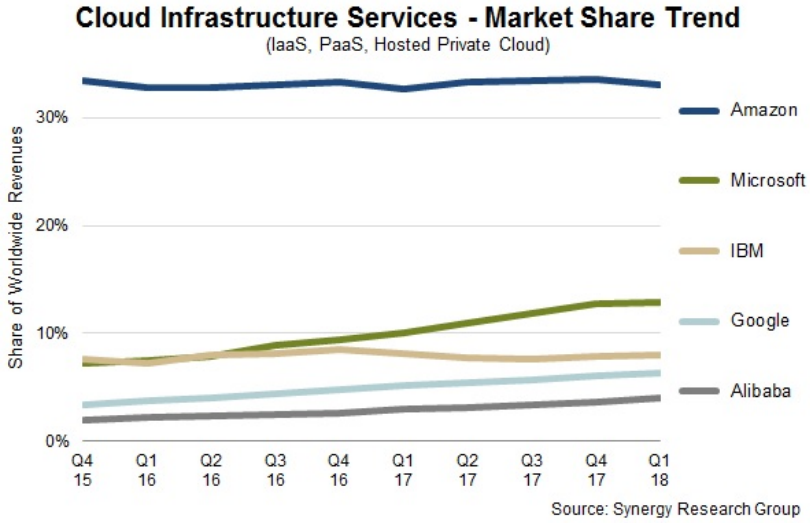

Figura 3.1: Comparativo dos principais provedores de nuvem computacional IaaS
Fonte: (FORBES BRASIL, 2019)

Evolução do Cenário

Esse modelo de utilização de recursos virtualizados de infraestrutura como serviço foi (e é) reproduzido por provedores de computação em nuvem ao redor do globo. Consequentemente, ao longo das duas últimas décadas, uma profusão de estudos decorrente de virtualizadores como Xen e Hyper-V foram realizados, além de exaustivos comparativos de desempenho entre estes, conforme retratado pelo capítulo anterior.

A AWS avaliou o Xen como uma ferramenta com diversas limitações, dentre elas de desempenho em relação ao *hardware*. Em 2016 o provedor criado por Jeff Bezos compra uma startup israelense chamada Annapurna Labs e, em 2017, esta anunciou a nova família de instâncias IaaS com uma modificação drástica: substituição completa virtualizador, onde o Xen fora substituído pelo KVM modificado voltado para otimização de desempenho. Desde então o Xen deixou de ser o virtualizador mais utilizado pela AWS. Atualmente o provedor continua atuando fortemente com o KVM de forma emulada e paravirtualizada em versões de uso exclusivo, além do VMware vSphere, Hyper-V e do Xen apenas em sistemas legados (FORBES BRASIL, 2019).

Outro fator que corrobora para uma nova abertura de cenário é a diminuição da linha de tendência de estudos relacionados ao tema desta obra ao longo dos anos. Conforme verificado no capítulo anterior o maior índice de profusão de artigos, *papers*, obras e estudos relacionados ao desempenho de ambientes virtualizados em *cloud computing* se deu até o ano de 2013. Conforme verificado anteriormente, a partir do ano de 2017 existiram mudanças drásticas nos modelos de virtualização e os estudos acadêmicos não acompanharam essas modificações na mesma velocidade.

Conforme explanado em capítulos anteriores, este novo cenário possibilitou novas oportunidades de novos estudos sobre ambientes virtualizados, seus métodos e paradigmas. Consequentemente, a análise de desempenho destes virtualizadores, outrora não observados, volta à discussão na academia.

Capítulo 4

Trabalhos Relacionados

Esta seção aborda os trabalhos relacionados ao tema dos últimos anos, observando pontos importantes de cada obra citada, além de desafios e limitações explanados. É importante salientar que esta obra tem um forte motivacional teórico baseado nas informações colhidas das obras citadas: a refatoração de alguns destes testes com o objetivo de atualização do estado da arte devido evolução das tecnologias envolvidas.

4.1 Container-based Operating System Virtualization (...) (2007)

Soltesz et al. (2007), descrevem uma abordagem de virtualização projetada para impor um alto grau de isolamento entre VMs, mantendo o uso eficiente dos recursos do sistema. A abordagem sintetiza ideias de trabalhos anteriores sobre contêineres de recursos e contêineres de segurança e aplica-as a sistemas operacionais de uso geral e tempo compartilhado. De fato, variantes de tais sistemas operacionais baseados em contêiner estão em uso de produção hoje - por exemplo, Solaris 10, Virtuozzo e Linux-VServer. Foram comparadas a arquitetura do VServer com uma geração recente do Xen.

Em termos de desempenho, as duas soluções são iguais para cargas de trabalho vinculadas à CPU, enquanto para cargas de trabalho centradas em I/O, o VServer faz um uso mais eficiente dos recursos do sistema e, portanto, obtém um melhor desempenho geral. Em termos de escalabilidade, o VServer ultrapassa de longe o Xen em cenários de uso em que o overbooking de recursos do sistema é necessário (por exemplo, PlanetLab, hospedagem gerenciada na Web, etc), enquanto para cenários de uso baseados em reserva envolvendo um pequeno número de VMs, o VServer retém uma vantagem pois, inerentemente, evita a duplicação do estado do sistema operacional.

As duas abordagens compartilham características em sua organização de alto nível, porém alguns recursos são exclusivos da ferramenta de virtualização, em específico. O Xen, por exemplo, é capaz de suportar vários *kernels*, enquanto o VServer não pode. O Xen também tem maior suporte para virtualização da pilha de rede e permite a possibilidade de migração de VM, um recurso que é possível para um design COS, mas ainda não disponível no VServer. O VServer, por sua vez, mantém uma pequena pegada de *kernel* e executa igualmente com *kernels* Linux nativos na maioria dos casos.

Infelizmente, não existe uma solução única. Como os testes mostraram, *benchmarks* relacionados a I/O têm pior desempenho no Xen quando comparados ao VServer. Este é um artefato de virtualizar dispositivos de I/O por meio de uma VM de *host proxy*. A VMware soluciona parcialmente esse problema incorporando *drivers* de dispositivo para um pequeno conjunto de dispositivos de I/O de alto desempenho com suporte diretamente em sua linha de produtos de *hypervisor* ESX. Em contraste, esse problema não é relevante para sistemas baseados em COS, onde toda I/O opera em velocidades nativas. Os experimentos indicam que os sistemas baseados em contêiner fornecem até 2x o desempenho dos sistemas baseados em *hypervisor* para cargas de trabalho do tipo servidor e escalam ainda mais, preservando o desempenho. Espera-se que os sistemas baseados em contêineres, como o VServer, incorporem mais do conjunto de recursos que atrai os usuários aos *hypervisors* (por exemplo, virtualização de rede completa, migração, etc.).

4.2 Quantifying the Performance Isolation Properties (...) (2007)

Matthews et al. (2007) apresentam o design de um *benchmark* de isolamento de desempenho e o usa para examinar três ambientes de virtualização - um exemplo de virtualização completa (VMware Workstation), um exemplo de paravirtualização (Xen) e dois exemplos de virtualização de nível de sistema operacional (Solaris contêineres e OpenVZ).

Para quantificar o isolamento de desempenho de um sistema de virtualização, os autores projetaram um conjunto de testes incluindo seis testes de estresse diferentes: *loops* alocando e tocando constantemente a memória sem liberá-la (intensivo de memória), *loops* criando novos processos filho (*fork bomb*), loops contendo operações aritméticas inteiras (CPU intensivo), execução de 10 *threads* de Iozone, cada uma executando um padrão alternado de leitura e gravação (iozone –i0 –i1 –r4 -s 100M -t 10 -Rb) (Disco Intensivo), 4 *threads* com cada uma enviando pacotes de tamanho 60K constantemente sobre UDP para receptores externos (rede intensivo - transmissão) e 4 *threads* em que cada um lê constantemente pacotes de 60K via UDP de remetentes externos (rede intensivo - recepção).

Para realizar cada teste, iniciaram-se um conjunto de máquinas virtuais de servidor web com bom comportamento na mesma máquina física. Os autores quantificaram a degradação do desempenho em VMs como comportamento inadequado e bem-comportado. Nos testes, foi usado o tempo de resposta relatado pelo *benchmark* SPECweb como a métrica de desempenho de interesse. No entanto, o pacote de teste de estresse pode ser usado em qualquer ambiente de produção para avaliar o impacto em outras métricas de interesse para os serviços em execução nas máquinas virtuais dos sistemas.

Como resultados, a virtualização completa protegeu completamente as VMs bem comportadas em todos os testes de estresse. A paravirtualização também oferece excelente isolamento de recursos. Nos testes Xen, as VMs bem comportadas sofreram no máximo uma degradação de 1,7% para o teste intensivo de disco com muitos outros testes mostrando apenas degradação leve, mas repetível. Com a virtualização no nível do sistema operacional, a necessidade de controles de recursos, seja como padrão ou por meio da

configuração adequada, era clara. Sem eles, cargas de trabalho bem e mal comportadas sofreram. O forte isolamento de recursos pode ser claramente adicionado à virtualização no nível do sistema operacional. Como no caso do Solaris e OpenVZ, o sistema operacional pode ser modificado para implementar novos algoritmos de agendamento de recursos que reforçam o isolamento de recursos nas VMs.

Quando os controles de recursos estavam disponíveis e eram usados adequadamente, apenas uma ligeira degradação ocorria, que não significativa o suficiente para causar mudanças perceptíveis no tempo de resposta ou usabilidade, foi observada. Os resultados destacam a diferença entre essas classes de sistemas de virtualização, bem como a importância de considerar várias categorias de consumo de recursos ao avaliar as propriedades de isolamento de desempenho de um sistema de virtualização.

4.3 Virtualization Performance: Perspectives and Challenges Ahead (2010)

McDougall e Anderson (2010) avaliam a solução vSphere sob os aspectos de *throughput, overhead*, latência e consolidação de servidores. A arquitetura vSphere foi projetada para ser uma plataforma de virtualização de alto desempenho para a empresa. Congruente com o caso de uso para consolidação de servidor, a plataforma de virtualização vSphere precisa ter sobrecarga muito baixa e fornecer a escalabilidade necessária para executar muitas máquinas virtuais simultaneamente. As prioridades de desempenho da plataforma vSphere podem ser resumidas como sobrecarga de virtualização de baixa CPU, escalabilidade vSMP, escalabilidade pSMP, densidade de embalagem, garantias de tempo de resposta, previsibilidade, taxa de transferência de IO, latência de IO e extrema observabilidade.

O desempenho da virtualização evoluiu bastante - de uma sobrecarga substancial para uma sobrecarga mínima para a maioria das cargas de trabalho. A plataforma vSPhere evoluiu a cada lançamento para fornecer melhor desempenho e suportar configurações maiores. Além disso, observa-se que o rendimento da plataforma virtualizada tem aumentado continuamente por meio de uma combinação de melhorias na pilha de *software* de virtualização e melhorias nas arquiteturas de *hardware*. Nos últimos cinco anos, a taxa de transferência de um *host* virtualizado, medida com VMmark, dobrou a cada ano. A combinação de eficiência aprimorada e *throughput* permitiu que a virtualização fosse aplicada a muitos tipos de cargas de trabalho e permitiu que a virtualização passasse de casos de uso de teste/desenvolvimento para virtualização de carga de trabalho de produção principal no data center.

4.4 A Synthetical Performance Evaluation of OpenVZ, Xen and KVM (...) (2010)

Che et al. (2010) avaliam a eficácia de três monitores de máquina virtual de código aberto disponíveis para a plataforma x86 - OpenVZ, Xen e KVM. Foram selecionados como dele-

gados da virtualização no nível do sistema operacional, paravirtualização e virtualização completa, respectivamente para medir seu desempenho do subsistema de processador, subsistema de arquivo, subsistema de rede e fornecer uma análise quantitativa e qualitativa comparação de três monitores de máquina virtual. Os dados de teste oferecem algumas informações úteis para os usuários escolherem o monitor de máquina virtual mais adequado para seus aplicativos. Em segundo lugar, foram medidos como uma caixa preta para obter seus dados de macro e micro desempenho, e os análises foram feitas como uma caixa branca para observar suas características de desempenho. Ao investigar seu princípio de *design* e prós e contras, alguns gargalos de desempenho potenciais são descobertos.

Para avaliar o desempenho do OpenVZ, Xen e KVM, uma série de experimentos foram conduzidos testando ambiente nativo e máquina virtual em OpenVZ, Xen e KVM com SPECCPU v2006, LINPACK v10.0.2, RAMSPEED v3.4.4, LMbench v3. 0, IOzone v3.287, Bonnie ++ v1.03, NetIO v126, WebBench v1.5, SysBench v0.4.8 e SPECJBB v2005. Todos os experimentos foram executados em uma máquina empresarial Dell OPTIPLEX 755 com um processador Intel Core2 DUO 2,33 GHz E6550, 2 GB DDRII 667 RAM, disco Seagate 250 GB 7200 RPM SATA II. Foi adotado o Gentoo Linux 2008.0 AMD64 com kernel 2.6.18 como sistema operacional *host* (ou seja, Gentoo nativo) e sistema operacional convidado (ou seja, Gentoo virtualizado), e OpenVZ no *kernel* atualizado 2.6.18 028stab056.1, Xen 3.3 e KVM-75 como monitor de máquina virtual. A memória alocada para cada máquina virtual foi definida para 1,5 GB.

Na avaliação do Macro-Desempenho, foi medida a sobrecarga de virtualização do subsistema de processador, subsistema de arquivo, subsistema de rede, servidor web, servidor de banco de dados e servidor Java em três monitores de máquina virtual. Já o Micro-Desempenho significa principalmente o bom desempenho de algumas operações ou instruções específicas. Foi medida a sobrecarga de virtualização de operações de sistema comuns e troca de contexto em quatro ambientes com LMbench.

Medindo e analisando OpenVZ, Xen e KVM com SPEC CPU2006, LINPACK, compilação de Kernel, RAMSPEED, LMbench, IOzone, Bonnie ++, NetIO, WebBench, SysBench e SPEC JBB2005, descobriu-se que o OpenVZ tem o melhor desempenho e o Xen segue o OpenVZ com uma leve degradação na maioria dos experimentos, enquanto KVM tem desempenho aparentemente inferior do que OpenVZ e Xen. Além disso, isso indica que a virtualização e paravirtualização no nível do sistema operacional têm alguma vantagem aparente em aplicativos de dados intensivos, como I/O de disco, I/O de rede, servidor web, servidor de banco de dados e servidor Java. No entanto, a virtualização e paravirtualização no nível do sistema operacional têm suas próprias desvantagens: a virtualização no nível do sistema operacional só pode virtualizar o sistema operacional convidado se o *kernel* é o mesmo que o sistema operacional *host* e a paravirtualização requer mudanças no *kernel* do sistema operacional convidado sistema. Ao contrário, a virtualização completa apresenta aos usuários o uso mais conveniente, sem qualquer modificação no sistema operacional convidado.

Consequentemente, geralmente há uma compensação entre perda de desempenho e facilidade de uso. Além disso, de acordo com os resultados dos testes, a virtualização do processador não é o gargalo de desempenho de um monitor de máquina virtual maduro,

enquanto a maioria dos monitores de máquina virtual perde muito desempenho em três aspectos: falha na tabela de página de hardware, solicitação de interrupção e I/O. Portanto, os focos devem ser fixados na otimização desses gargalos potenciais para melhorar o desempenho dos monitores da máquina virtual.

4.5 Performance Evaluation of Container-based Virtualization (...) (2013)

Xavier et al. (2013) conduziram uma série de experimentos a fim de realizar uma avaliação aprofundada do desempenho da virtualização baseada em contêiner para HPC. Também avaliaram a compensação entre desempenho e isolamento em sistemas de virtualização baseados em contêiner e os compararam com o Xen, que é um representante dos sistemas de virtualização baseados em *hypervisors* tradicionais usados hoje.

Foram feitos vários experimentos com as implementações atuais de virtualização baseada em contêiner Linux: Linux VServer, OpenVZ e LXC. O Xen foi escolhido como representante da virtualização baseada em *hypervisor*, por ser considerada uma das implementações mais maduras e eficientes desse tipo de virtualização. A configuração experimental consiste em quatro Dell PowerEdge R610 idênticos com dois processadores Intel Xeon E5520 de 2,27 GHz (com 8 núcleos cada), 8 MB de cache L3 por núcleo, 16 GB de RAM e um adaptador NetXtreme II BCM5709 Gigabit Ethernet. Todos os nós são interconectados por um switch Ethernet Dell PowerConnect 5548. A distribuição Linux Ubuntu 10.04 LTS (Lucid Lynx) foi instalada em todas as máquinas *host* e as configurações padrão foram mantidas, exceto para o *kernel* e pacotes que foram compilados a fim de satisfazer os requisitos dos sistemas de virtualização.

Diferentes versões do *kernel* podem introduzir ganhos ou perdas de desempenho que influenciam os resultados dos experimentos. Portanto, todos os sistemas foram compilados com a mesma versão do *kernel*, no caso a 2.6.32-28, porque ela tem suporte para todos os patches e configurações do sistema. Portanto, para OpenVZ, foi corrigido o *kernel* (2.6.32-feoktistov) e instalado o pacote vzctl (3.0.23-8), que é necessário para gerenciar os contêineres OpenVZ. O kernel OpenVZ foi compilado com o arquivo de configuração oficial (.config) sugerido pela equipe de desenvolvimento do OpenVZ, a fim de garantir que todas as opções do *kernel* OpenVZ foram habilitadas. Para Linux-VServer, também foi corrigido o *kernel* (2.3.0.36.29.4) e instalado o pacote util-vserver (0.30.216 r2842-2) para controlar os contêineres Linux-VServer. O LXC já possui uma implementação principal no código-fonte oficial do *kernel*. Dessa forma, foi instalado o kit de ferramentas LXC (0.6.5-1) e garantido que todos os requisitos presentes pela ferramenta lxc-checkconfig foram atendidos. Finalmente, para o Xen, foi compilado e instalado o kernel (xen4.1.2) e as ferramentas fornecidas pelo pacote Xen. Foram avaliados, portanto, a performance de computação, memória, disco, rede, *overhead* em aplicações HPC e isolamento.

Os *clusters* HPC são normalmente compartilhados entre muitos usuários ou institutos, que podem ter requisitos diferentes em termos de pacotes de *software* e configurações. Conforme apresentado, essas plataformas podem se beneficiar do uso de tecnologias de virtualização para fornecer melhor compartilhamento de recursos e ambientes personali-

zados. No entanto, o HPC só poderá tirar proveito dos sistemas de virtualização se a sobrecarga de desempenho fundamental (como CPU, memória, disco e rede) for reduzida. Nesse sentido, descobriu-se que todos os sistemas baseados em contêiner têm um desempenho quase nativo de CPU, memória, disco e rede. As principais diferenças entre eles estão na implementação do gerenciamento de recursos, resultando em isolamento e segurança insuficientes. Enquanto o LXC controla seus recursos apenas por cgroups, o Linux-VServer e o OpenVZ implementam seus próprios recursos, introduzindo ainda mais limites de recursos, como o número de processos, que descobriu-se ser uma contribuição importante para dar mais segurança a todo o sistema. Supõe-se que esse recurso será introduzido em cgroups em um futuro próximo.

O exame cuidadoso dos resultados de isolamento revela que todos os sistemas baseados em contêiner ainda não estão maduros. O único recurso que pôde ser isolado com sucesso foi a CPU. Todos os três sistemas apresentaram isolamento de desempenho insatisfatório para memória, disco e rede. No entanto, para ambientes HPC, que normalmente não requerem a alocação compartilhada de uma partição de *cluster* para vários usuários, esse tipo de virtualização pode ser muito atraente devido à sobrecarga de desempenho mínima. Como os aplicativos HPC foram testados, até agora, o LXC demonstra ser o mais adequado dos sistemas baseados em contêineres para HPC. Apesar do LXC não apresentar o melhor desempenho do NPB na avaliação de vários nós, seus problemas de desempenho são compensados pela facilidade de gerenciamento e gerenciamento. No entanto, algumas técnicas de virtualização usuais que são úteis em ambientes HPC, como migração ao vivo, pontos de verificação e currículo, ainda precisam ser implementadas pela equipe de desenvolvedor do *kernel*.

4.6 A Component-Based Performance Comparison (...) (2013)

Hwang et al. (2013) realizaram uma ampla comparação de desempenho de quatro plataformas de virtualização populares: Hyper-V, KVM, vSphere e Xen. Foi usada uma abordagem baseada em componentes que isola o desempenho por recurso, como CPU, memória, disco e rede. Também estudaram o nível de isolamento de desempenho fornecido por cada *hypervisor* para medir como as VMs concorrentes podem interferir umas nas outras. Os resultados sugerem que o desempenho pode variar entre 3% e 140% dependendo do tipo de recurso estressado por um aplicativo, e que o nível de interferência causado por VMs concorrentes pode variar de 2X a 10X, dependendo do *hypervisor*.

A metodologia para a comparação de desempenho de *hypervisors* é detalhar cada componente de recurso um por um com uma carga de trabalho de *benchmark* específica. Os componentes incluem CPU, memória, I/O de disco e I/O de rede. Cada componente tem diferentes requisitos de virtualização que precisam ser testados com diferentes cargas de trabalho. Isso foi seguido com um conjunto de cargas de trabalho mais gerais representativas de aplicações de nível superior.

Estudaram-se casos em que as VMs são atribuídas a uma única VCPU ou quatro VCPUs (o máximo em nosso sistema de teste), uma variedade de tipos e tamanhos de

I/O, bem como o comportamento de I/O de nível superior, como o de um e-mail, arquivo ou servidor da web e as características de rede de baixo nível e o desempenho do servidor web Além dos testes de nível de componente descritos acima, foram executados vários *benchmarks* de nível de aplicação. Esses *benchmarks* ilustram como a sobrecarga de diferentes componentes interage para determinar o desempenho geral. Por fim, também foram testados cenários em que várias VMs interferentes são executadas simultaneamente. O isolamento do desempenho é uma meta importante para a camada de virtualização, especialmente quando usada em ambientes de nuvem. Se o isolamento de desempenho falhar, os clientes podem reclamar que o desempenho da VM varia dependendo do padrão de uso de outros locatários. Nos experimentos de *benchmarking* compostos, foram explorados quão bem o *hypervisor* programa ou gerencia os recursos físicos compartilhados pelas VMs.

Os resultados mostram que não há *hypervisor* perfeito que seja sempre a melhor escolha; aplicações diferentes se beneficiarão de *hypervisors* diferentes, dependendo de suas necessidades de desempenho e dos recursos precisos de que necessitam. No geral, o vSphere tem o melhor desempenho em nos testes. No entanto, todos os outros três *hypervisors* têm um desempenho respeitável e cada um dos *hypervisors* testados tem pelo menos um *benchmark* com desempenho superior a todos os outros. Em geral, as tarefas relacionadas à CPU e à memória apresentam os níveis mais baixos de sobrecarga, embora o KVM experimente sobrecargas de memória mais altas quando todos os núcleos do sistema estão ativos.

O desempenho diverge mais fortemente para atividades de I/O, em que o Xen exibe altas sobrecargas ao executar pequenas operações de disco. O Hyper-V também experimenta uma desaceleração dramática quando vários núcleos são dedicados à execução de pequenas leituras e gravações sequenciais. O Xen também sofre com o rendimento da rede. O Xen foi testado usando virtualização completa assistida por *hardware*, enquanto o *hypervisor* foi originalmente desenvolvido para paravirtualização. Na prática, nuvens públicas como Amazon EC2 usam Xen em modo paravirtualizado para todos, exceto seus tipos de instância de ponta. Os testes de nível de aplicação correspondem a esses resultados, com diferentes *hypervisors* exibindo diferentes sobrecargas dependendo da aplicação e do número de núcleos atribuídos a eles. Toda essa variação sugere que combinar adequadamente uma aplicação com o *hypervisor* certo é difícil, mas pode valer a pena o esforço, pois a variação de desempenho pode chegar a 140%.

4.7 Performance Overhead Among Three Hypervisors (...) (2013)

Li et al. (2013) fornecem alguns casos úteis para ajudar a decidir quando usar determinado *hypervisor* para uma carga de trabalho específica. De forma geral, este estudo mostra que as tecnologias de virtualização são relativamente imaturas e significativamente mais análises experimentais serão necessárias para que se tornem realmente adequadas para aplicações de missão crítica. Além disso, é importante compreender as características de uma aplicação (por exemplo, é intensivo em CPU ou I/O) para determinar melhor a

plataforma de virtualização certa a ser usada.

Foram executados seis *benchmarks MapReduce Hadoop* diferentes nos experimentos. Três dos *benchmarks* (TestDFSIO *Write / Read* e *TeraSort*) podem ser encontrados nas últimas distribuições do *Hadoop*. Os últimos três *benchmarks* (*Word Count, KMeans Clustering* e *Hivebench*) foram executados a partir do conjunto de *benchmarks HiBench*. Os *benchmarks* foram categorizados da seguinte forma: *benchmarks* de CPU, I/O-*bound* e CPU com I/O-Bound. Os *benchmarks* vinculados à CPU usam uma quantidade significativa de CPU e o desempenho é limitado devido à falta de energia da CPU. Da mesma forma, os *benchmarks* ligados a I/O usam uma quantidade significativa de disco (CPU também significativa, mas apenas em termos de espera de I/O). Os *benchmarks* de CPU com I/O-Bound usam uma quantidade significativa de CPU e I/O, com a CPU sendo utilizada tanto para computação quanto aguardando I/O.

Os resultados foram verificados usando o micro-*benchmark Filebench*. Analisaram-se os resultados de desempenho de cada experimento e encontraram semelhanças e variações significativas de desempenho entre os diferentes *hypervisors*. Por exemplo, para *benchmarks* vinculados à CPU, descobriu-se que havia diferenças de desempenho insignificantes entre os *hypervisors*; no entanto, para *benchmarks* ligados a I/O e *benchmarks* que eram tanto CPU quanto I/O, havia diferenças consideráveis de desempenho. O *hypervisor* comercial foi considerado melhor na gravação de disco, enquanto o KVM foi melhor na leitura de disco. O Xen era melhor quando havia uma combinação de leitura e gravação de disco com cálculos intensivos de CPU. Aumentar o paralelismo de processamento para cargas de trabalho intensivas de I/O é prejudicial ao desempenho, mas pode aumentar o desempenho em cargas de trabalho intensivas de I/O e CPU.

4.8 Performance Evaluation and Comparison (...) (2013)

Elsayed e Abdelbaki (2013), caracterizam as melhorias na utilização de recursos de *hardware* após o uso da virtualização, além de comparar o desempenho dos principais *hypervisors* do mercado. A ideia é avaliar o oferecimento de recursos, interoperabilidade e estabilidade no ambiente de produção diário. Os *hypervisors* escolhidos para o teste foram o VMware ESXi versão 5, o Microsoft Hyper-V 2008 R2 e o Citrix Xen Server 6.0.2. As soluções foram colocadas em ambientes com as mesmas configurações de *hardware* e topologia de rede. As comparações são baseadas em um conjunto de testes de código aberto disponíveis publicamente, projetados para avaliar o desempenho da máquina em diferentes níveis de cargas de trabalho sustentadas semelhantes àquelas geradas por aplicações de negócios com uso intensivo de banco de dados. Os resultados são obtidos monitorando e analisando o efeito desse teste de estresse no processador, na memória e na I/O do disco.

Foram usados três servidores como hospedeiros de virtualização do modelo Intel com a seguinte especificação: Processador duplo X5570 2.93 GHz, 24 GB de memória DDR3, 5X146 GB de disco rígido SAS e 4 portas de rede de 1 Gbps. Para cada servidor, a VM foi criada com a seguinte configuração: 4 CPUs virtuais, 1 X 50GB HD, 20 GB de RAM, Microsoft Windows Server 2008 R2 Enterprise X64 e SQL Server 2008 R2. Foi usado o servidor de gerenciamento baseado em Intel com a seguinte especificação: Processador

Quad X7350 2,93 GHz, 8 GB de memória DDR3, 3 X 146 GB de disco rígido SAS e 4 portas de rede de 1 Gbps. Este servidor foi configurado com Microsoft Windows Server 2008 Enterprise X64 com algumas ferramentas utilizadas no gerenciamento e monitoramento do ambiente virtualizado.

Foram criadas medidas normalizadas de desempenho e carga de trabalho. Uma aplicação com uso intensivo de disco é executada em uma única máquina virtual hospedada em cada um dos *hypervisors*. Foi avaliado o desempenho usando o aplicativo de teste DVD *store version* 2 (DS2) e ferramentas de monitoramento de rede PRTG para comparação de três *hypervisors* de virtualização disponíveis na arquitetura X86: VMware ESXi 5, Microsoft Hyper-V 2008 R2 e Citrix Xen Server 6.0.2. O experimento é feito com a mesma configuração do sistema para todos os testes realizados. Cada cenário é dividido em dois testes. O primeiro teste usa o programa de *driver* de execução DS2 integrado para identificar qual *hypervisor* tem desempenho efetivo do banco de dados SQL. O segundo teste usa o tamanho de instância de banco de dados extra (10 GB de SQL) que simula 20 milhões de clientes, 100.000 produtos e 100.000 pedidos/mês como uma carga de trabalho pesada. O primeiro cenário começa com carga leve; apenas uma VM executando a instância SQL de 10 GB personalizada. O segundo cenário aumenta a carga criando 4 VMs; cada um executando a instância personalizada de 10 GB do SQL sobre cada *hypervisor*. O terceiro cenário compara entre Hyper-V 2008 R2 e Hyper-V 2012 RC como um novo produto no mercado para testar o aprimoramento no desempenho.

O primeiro teste identifica qual *hypervisor* tem desempenho efetivo de banco de dados SQL por meio da menor utilização da CPU e por ordem superior por minuto (OPM). A partir dos resultados anteriores do primeiro e do segundo cenário, percebeu-se que o VMware ESXi 5 tem uma liderança significativa e demonstrou maturidade no manuseio de CPU quanto de memória. Ele é seguido em ordem decrescente pelo Citrix Xen Server 6.0.2 e o Hyper-V 2008 R2 no primeiro cenário; e seguido em ordem decrescente pelo Hyper-V 2008 R2 e pelo Citrix Xen Server 6.0.2. O segundo teste é conduzido para identificar qual *hypervisor* tem utilização efetiva da CPU e memória consumida após submetê-los a uma carga pesada. Foi notável que a superioridade para o Citrix Xen Server 6.0.2 seguido pelo Hyper-V 2008 R2 e o VMware ESXi 5 vem no final no primeiro cenário. O segundo cenário indica que o Citrix Xen Server 6.0.2 está fora da corrida pelo desempenho após aumentar a carga. Isso se deve ao fato de que seu consumo de memória atingiu 98% e torna a resposta do servidor muito lenta, embora sua utilização de CPU seja a mais baixa. A superioridade é para o Hyper-V 2008 R2 seguido em ordem decrescente pelo VMware ESXi 5; devido à menor utilização da CPU do servidor *host* e à memória consumida ao aplicar o teste de estresse. O terceiro cenário demonstra que o Hyper-V 2012 RC tem aprimoramento no manuseio de memória e I/O de disco, mas com um aumento na utilização da CPU do que o Hyper-V 2008 R2.

4.9 System Performance evaluation of Paravirtualization (...) (2014)

S. et al. (2014) avalia o desempenho, por meio de *benchmarking*, de três *hypervisors* básicos, um com suporte à paravirtualização com Xen-PV, o segundo com suporte à virtualização de contêiner com OpenVZ e o último a virtualização completa com Xen Server.

O ambiente de teste é composto com máquinas, cujas configurações de *hardware* são semelhantes, com *hypervisors* que suportam as três tecnologias instalados e configurados. Para cada máquina, uma VM (Virtual Machine) com Debian Wheezy é carregada com tais *hypervisors*. Em sequência, cada *hypervisor* tem seu desempenho avaliado por meio da ferramenta UnixBench. Dentre os testes estão manipulação de *strings*, número de chamadas *execl* por segundo, *Pipe throughput*, capacidade de cópia de arquivo, troca de contexto baseada em *pipe*, criação de processo, taxa de *script* de *shell*, operações de ponto flutuante e desempenho de sistema.

A partir dos resultados do UnixBench, é possível analisar quem teve o melhor resultado em determinado teste. Para manipulação de strings, número de chamadas "*execl*"por segundo e *Pipe throughput*, o OpenVZ teve os melhores valores. Já o Xen Server teve os melhores resultados para taxas de cópia de arquivo, troca de contexto baseada em pipe, criação de processo, taxa de *script* de *shell*, operações de ponto flutuante e desempenho de sistema. A partir dos resultados angariados, nota-se que diferentes *hypervisors* apresentam diferenças em vários testes. Em alguns, o OpenVZ se sobressai, mas o Xen Server oferece o melhor desempenho na maioria dos casos. Já o Xen-PV teve os piores resultados.

4.10 Hypervisors vs. Lightweight Virtualization (...) (2015)

Morabito, Kjällman e Komu (2015) apresentam uma comparação detalhada de desempenho da virtualização tradicional baseada em *hypervisor* e novas soluções leves. Nas suas análises, foram usadas ferramentas de *benchmarks* a fim de entender os pontos fortes, fracos e anomalias introduzidas por essas diferentes plataformas em termos de processamento, armazenamento, memória e rede. A ideia é quantificar o nível de sobrecarga introduzido por essas plataformas e a lacuna existente em comparação a um ambiente não virtualizado.

O *benchmarking* foi feito usando KVM, LXC, Docker e OSv. Foi usado o desempenho nativo (não virtualizado) como um caso base para medir a sobrecarga da virtualização. As ferramentas de *benchmark* medem (usando cargas de trabalho genéricas) CPU, memória, I/O de disco e desempenho de I/O de rede. Foi repetido cada alvo de medição com ferramentas diferentes para verificar a consistência entre os diferentes resultados obtidos. Também foi repetida cada medição individual 15 vezes. O *hardware* usado para a experimentação empírica foi um Dell Precision T5500, com processador Intel Xeon X5560 (8M Cache, 2.80 GHz, 4 cores, 8 threads), Memória de 12 GB (3 x 4 GB) 1333 MHz DDR3 ECC R, Disco OCZ-VERTEX 128 GB, Rede de 10 Gb/s interface e Sistema Operacional

Ubuntu 14.04 (64-bit).

Para análise de CPU foram usados Y-cruncher, NBENCH e Linpack. Para o Disco, o Bonnie++. A memória foi avaliada com o *STREAM* e a rede foi avaliada com o Netperf. Como resultado, as soluções baseadas em contêiner e outros sistemas emergentes estão desafiando máquinas virtuais baseadas em *hypervisor* tradicionais em computação em nuvem. As novas tecnologias são mais leves, facilitando assim uma implantação mais densa de serviços. Os resultados mostram claramente que o desempenho do *hypervisor* KVM melhorou drasticamente nos últimos anos. No entanto, especialmente a eficiência de I/O de disco ainda pode representar um gargalo para alguns tipos de aplicações, embora uma avaliação adicional para I/O de disco ainda seja necessária devido a alguma inconsistência entre as diferentes ferramentas. O nível de sobrecarga introduzido pelos contêineres pode ser considerado quase insignificante. Levando em consideração todas as diferenças entre o LXC e o Docker, confirmou-se que os contêineres têm um bom desempenho, embora a versatilidade e a facilidade de gerenciamento sejam recompensadas em termos de segurança.

4.11 Performance Overhead Comparison between Hypervisor (...) (2017)

Li et al. (2017) traz à tona a questão dos provedores de nuvem e consumidores em entender até que ponto uma solução de virtualização candidata incorre em influência na QoS da nuvem. Dadas as características das soluções de virtualização baseadas em *hypervisor* e as baseadas em contêiner, uma hipótese intuitiva poderia ser: um serviço baseado em contêiner exibe melhor desempenho do que seu serviço VM baseado em *hypervisor* correspondente. Para averiguar essa premissa, os autores utilizaram uma máquina física com recursos "apenas o suficiente"como linha de base para investigar quantitativamente e comparar as sobrecargas de desempenho entre as virtualizações baseadas em contêiner e em *hypervisor*, mais especificamente o Docker e o VMWare Workstation 12 Pro, respectivamente.

Uma vez que a comparação entre os *overheads* de desempenho do contêiner e da VM é essencialmente baseada em sua avaliação de desempenho, esse trabalho pode ser definido como um estudo de avaliação de desempenho que pertence ao campo de ECS (*Experimental Computer Science*), com a metodologia DoKnowMe para orientar as implementações de avaliação neste estudo. DoKnowMe é uma metodologia de avaliação abstrata sobre a analogia de classe na programação orientada a objetos. Ao integrar artefatos de conhecimento de domínios específicos, DoKnowMe pode ser personalizado em metodologias específicas (por analogia de “objeto”) para facilitar a avaliação de diferentes sistemas de computação concretos.

As propriedades físicas avaliadas, com suas respectivas métricas, foram a Comunicação (*throughput* de dados), Computação (avaliação da latência), Memória (*throughput* de dados) e o armazenamento (*throughput* de dados e velocidade na transação). Os *benchmarks* usados foram o *Iperf, HardInfo, STREAM e Bonnie++*, respectivamente. Como resultado, desempenho médio do contêiner é geralmente melhor do que o da VM e é até

comparável ao da máquina física com relação a muitos recursos. Especificamente, o contêiner tem menos de 4% de sobrecarga de desempenho em termos de *throughput* de dados de comunicação, latência de computação, *throughput* de dados de memória e *throughput* de dados de armazenamento. No entanto, a virtualização baseada em contêiner pode atingir um gargalo na velocidade de transação de armazenamento, com sobrecarga de até 50%. Conforme mencionado anteriormente, os autores interpretaram a taxa de transferência de dados do tamanho do *byte* na velocidade da transação de armazenamento, porque cada *byte* essencialmente chama uma transação de disco aqui.

Em contraste, embora a VM forneça o pior desempenho na maioria dos casos, ela pode ter um desempenho tão bom quanto a máquina física ao resolver o problema de N-Queens ou gravar dados de tamanho pequeno no disco. A perda de desempenho resultante de virtualizações é mais visível na variabilidade de desempenho. Por exemplo, a sobrecarga de variabilidade do contêiner pode chegar a mais de 500% em relação ao cálculo de Fibonacci e à operação da tríade de memória. Da mesma forma, embora o contêiner geralmente mostre menos variabilidade de desempenho do que a VM, ainda existem casos excepcionais: o contêiner tem a maior variação de desempenho no trabalho de computação da transformação de Fourier, enquanto mesmo a variabilidade de desempenho da VM não é pior do que a da máquina física durante a execução Trabalhos de CryptoHash, N-Queens e Raytracing. No geral, esse trabalho revela que as sobrecargas de desempenho dessas duas tecnologias de virtualização podem variar não apenas de acordo com os recursos, mas também de acordo com a natureza da carga de trabalho. Embora a solução baseada em contêiner seja sem dúvida leve, a tecnologia baseada em *hypervisor* não traz uma sobrecarga de desempenho maior em todos os casos.

4.12 Virtual machine consolidation (...) (2019)

Gonzalez e Juiz (2019) apresentam um *survey* sobre sobrecarga de consolidação de máquina virtual. Os fatores de influência indiretos são analisados ao longo do trabalho. Os autores propõem uma categorização que classifica os trabalhos de pesquisa mais importantes sobre virtualização e sobrecarga de consolidação de máquinas virtuais.

Os autores trazem uma taxonomia dos fatores que mais influenciam a sobrecarga. A sobrecarga da virtualização e da consolidação da máquina virtual depende de vários fatores. A tecnologia de virtualização tem uma sobrecarga inerente devido à camada de *software* adicionada, o gerenciador de máquina virtual. Além disso, se houver várias máquinas virtuais consolidadas no mesmo servidor físico, uma sobrecarga adicional devido à contenção da máquina virtual deve ser levada em consideração. Ao consolidar as máquinas virtuais, é necessário levar em consideração duas fontes de *overhead*: uma do gerenciador da máquina virtual e a resultante do gerenciamento do acesso aos recursos físicos de várias máquinas virtuais. Para avaliar o *overhead* devido à consolidação da máquina virtual, a seleção da carga de trabalho em relação ao *benchmarking* é crucial. Dependendo do objetivo do *benchmarking*, pode-se selecionar um *macrobenchmark* ou um *microbenchmark*. Por um lado, um *macrobenchmark* mede todo o sistema e geralmente modela alguma carga de trabalho; eles podem ser *benchmarks* sintéticos ou de aplicação.

Por outro lado, um *microbenchmark* mede uma parte específica do sistema; eles podem ser considerados como um subconjunto de referências sintéticas no sentido de que são artificiais. No entanto, eles não tentam modelar nenhuma carga de trabalho real.

A virtualização em nível de sistema pode ser implementada por meio de quatro técnicas: virtualização completa, paravirtualização, virtualização assistida por *hardware* e virtualização parcial. Dependendo da implementação, o sistema precisa lidar com mais ou menos camadas de tradução extras. Já o *hypervisor* pode ser implementado de duas maneiras: na parte superior do *hardware* físico ou na parte superior do sistema operacional (*hypervisor* tipo I ou tipo II, respectivamente). A abordagem de implementação do *hypervisor* de uma forma ou de outra afetará a sobrecarga de virtualização. Como os *hypervisors* tipo II precisam de um sistema operacional para serem implantados, a sobrecarga consequente será maior do que a sobrecarga inerente do *hypervisor* tipo I. Por fim, um servidor físico é composto principalmente de três componentes: CPU, memória principal e I/O de disco. A sobrecarga da virtualização depende do dispositivo que é responsável pela execução da carga de trabalho, já que as máquinas virtuais desejam acessar recursos físicos simultaneamente por meio do *hypervisor*.

Para estudar a sobrecarga, o uso de *microbenchmarks* é mais comum do que o uso de *macrobenchmarks*. Esse fato se deve à facilidade de estudar a sobrecarga em um dispositivo específico e não em todo o sistema. Como consequência, há também uma lacuna de pesquisa no estudo da virtualização em todo o sistema. Do ponto de vista da tecnologia de virtualização, a técnica assistida por HW é a abordagem mais utilizada para realizar a avaliação de *overhead*. Este fato se deve à evolução do *hardware*, dada a lacuna no estudo das demais tecnologias de virtualização.

A implementação de *hypervisor* mais estudada é o tipo I. Isso ocorre porque as máquinas virtuais são implementadas o mais próximo possível do *hardware*. A implementação do tipo II requer que mais camadas sejam implementadas e o desempenho é pior. É necessário estudar a sobrecarga de consolidação da máquina virtual em *hypervisores* do tipo II para tentar reduzir a degradação do desempenho. Levando em consideração o dispositivo do sistema, o dispositivo mais estudado é a CPU. Isto se deve ao fato de a CPU ser o dispositivo mais utilizado em um sistema. Como consequência, sua degradação de desempenho terá um impacto com mais peso em todo o sistema. Do ponto de vista dos grupos (filas), existem alguns fatos.

A paravirtualização e a virtualização completa não são adequadas para experimentação real, bem como *hypervisor* tipo II devido às implementações mais pesadas em comparação com as do tipo I. Além disso, a CPU é o dispositivo mais estudado em trabalhos reais de experimentação. Então, a maioria dos modelos de sistema e simulações estudam o comportamento da CPU. Pode-se afirmar que a maioria dos estudos mede a degradação do desempenho em termos de tempo de resposta. No entanto, é equivalente a expressar essa métrica em termos de taxa de transferência. Além disso, do ponto de vista da análise do sistema, verifica-se que as abordagens mais utilizadas referem-se ao estudo de um dispositivo físico específico e ao estudo de cada máquina virtual em vez de todo o conjunto deles. Já a tecnologia de contêiner permite usar a virtualização de uma maneira mais leve. No entanto, muitas desvantagens de segurança e desempenho são inerentes a essa tecnologia. Portanto, faltam estudos sobre a sobrecarga de desempenho em máquinas

virtuais.

O fator de influência sobre a sobrecarga mais estudado é a diferença entre ambientes virtualizados e não virtualizados sem levar em conta o número de máquinas virtuais consolidadas. Além disso, *microbenchmarks* são usados na virtualização assistida por *hardware* e avaliação de *hypervisors* tipo I. A partir desses estudos, o tempo de resposta é a principal métrica para medir a degradação do desempenho na CPU. Como consequência, falta o estudo dos demais fatores de influência do *overhead*, o que pode ser considerado como um futuro trabalho de pesquisa.

4.13 Redação de Obras do Gênero

Realizando uma análise mais apurada da quantidade de obras relacionadas ao tema nos últimos 12 anos que foram submetidas e disponibilizadas na plataforma do Google Scholar é possível observar que o maior índice de profusão destes artigos e obras ocorreram nos anos de 2007 e 2013. Este fenômeno observado pode ser correlacionado com o advento dos grandes *players* de *cloud computing* no mundo (Amazon Web Services em 2006, Microsoft Azure em 2010 e Google Cloud Platform em 2013) e, consequentemente, um aumento na concorrência como é representado na Figura e Tabela 4.1.

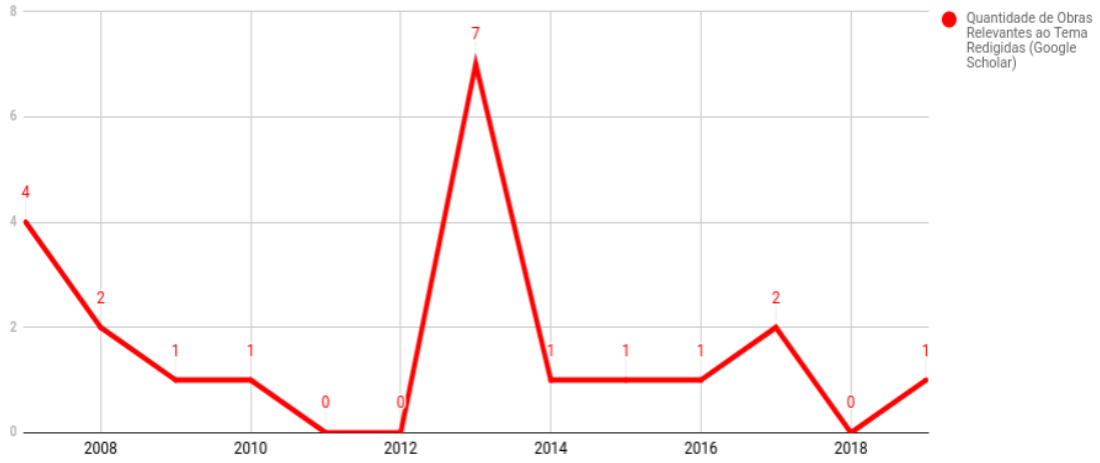

Figura 4.1: Relação de Obras Redigidas Sobre o Tema
Fonte: Próprio Autor

Tabela 4.1: Obras Desenvolvidas Sobre o Tema

Titulo	**Ano**	**Bare Metal**	**Emulação**	**Para virtualização**	**Virt. Leve**
Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors	2007	Xen Server	Não	Não	Linux VServer
Quantifying the Performance Isolation Properties of Virtualization Systems	2007	Xen Server	Não	Xen-PV	Solaris contêineres e OpenVZ
Virtualization Performance	2008	Não	vSphere	Não	Não
A Synthetical Performance Evaluation of OpenVZ, Xen and KVM	2010	Xen Server	Não	KVM	OpenVZ
Performance Evaluation of Container-based Virtualization for High Performance Computing	2013	Não	Não	Não	Linux VServer, OpenVZ e LXC
A Component-Based Performance Comparison of Four Hypervisors	2013	Hyper-V, KVM e vSphere	KVM	Não	Não
Performance Overhead Among Three Hypervisors: An Experimental Study using Hadoop Benchmarks	2013	CVM, KVM e Hyper-V	Não	Não	Não
Performance Evaluation and Comparison of The Top Market Virtualization Hypervisors	2013	vSphere, Xen Server e Hyper-V	Não	Não	Não
System Performance evaluation of Paravirtualization, Container virtualization and Full virtualization	2014	Xen Server	Não	Xen-PV	OpenVZ
Hypervisors vs. Lightweight Virtualization: a Performance Comparison	2015	Não	KVM	Não	Docker e LXC
Performance Overhead Comparison between Hypervisor and Container based Virtualization	2017	VMware Workstation	Não	Não	Docker
Virtual machine consolidation: a systematic review of its overhead influencing factors	2019	Hyper-V, KVM e vSphere	QEMU KVM	KVM	Não

Fonte: Próprio Autor

Capítulo 5

Metodologia

Esta obra está metodologicamente ancorada por meio de pesquisa analítica de abordagem quantitativa através de análise documental e dados coletados mediante testes de desempenho em ambiente controlado com as ferramentas citadas anteriormente. Os dados desta obra terão origem bibliográfica e, principalmente, laboratorial com o desenvolvimento de quatro estruturas de computação em nuvem IaaS distintas de alto desempenho fisicamente similares voltadas para coleta de dados de desempenho de dispositivos de armazenamento. Cada estrutura de nuvem contará com um determinado tipo de virtualização com o *hypervisor/software* mais recomendado para a mesma na atualidade: virtualização tradicional baseada em *hardware* (VMware vSphere Esxi 6.7), virtualização tradicional baseada em *software* (QEMU/KVM 4.2) e paravirtualização (KVM paravirtual com VirtIO Drivers).

5.1 Modelagem da Infraestrutura de Testes

Para desenvolvimento dos testes foram configuradas quatro estruturas físicas similares. Cada estrutura contou com:

- 1 servidor para hypervisor Dell PowerEdge R740, modelo 2018, de dois processadores escaláveis Intel Xeon de 2ª geração com até 48 núcleos de processamento modelo Silver 4210 de 2.20GHz e 24 slots DIMM DDR4, conforme figura 5.1 (DELL BRASIL, 2020b).

Figura 5.1: Dell PowerEdge R740
Fonte: (DELL BRASIL, 2020b)

- 1 servidor para armazenamento Dell PowerEdge R640, modelo 2018, de dois processadores escaláveis Intel Xeon de 2ª geração com até 28 núcleos por processador modelo Silver 4210 de 2.20GHz, 24 slots DIMM DDR4, armazenamento de 64 TB

de Hard Disk Drive (HDD) (8 discos de 8 TB cada) e armazenamento de 480 GB de Solid-State Drive (SSD) (2 discos de 240 GB cada) como servidor de armazenamento, conforme figura 5.2 (DELL BRASIL, 2020a).

Figura 5.2: Dell PowerEdge R640
Fonte: (DELL BRASIL, 2020a)

- Interconexão dos equipamentos em fibra ótica multimodo de 10 Gb/s (*gigabits* por segundo). As interfaces físicas de rede em cada equipamento estão em *bonding* (agregação de múltiplas interfaces de rede em uma única interface lógica) de categoria "ativo-*backup*", conforme figura 5.3.

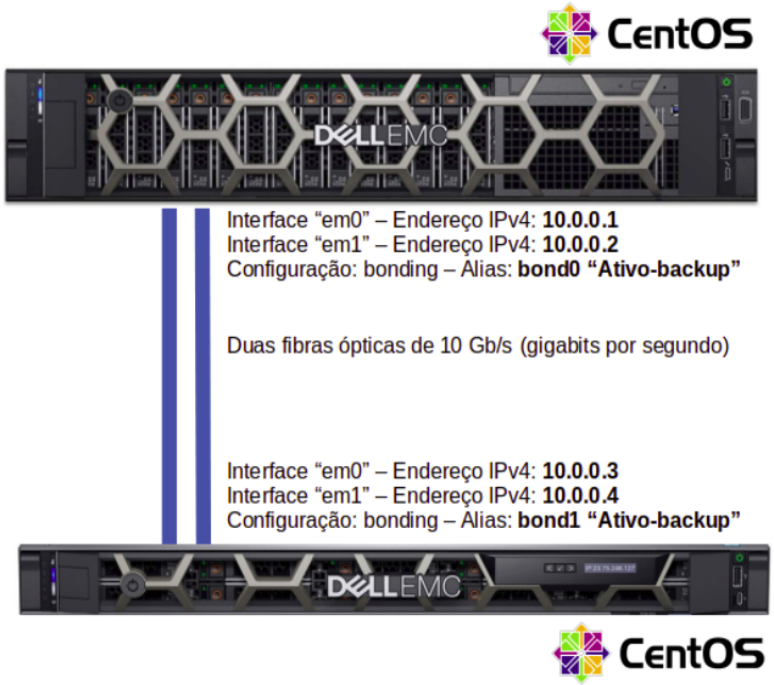

Figura 5.3: Esquema lógico entre os equipamentos de um conjunto
Fonte: Próprio Autor

Além disso, mais 1 servidor para hypervisor Dell PowerEdge R740, modelo 2018, de dois processadores escaláveis Intel Xeon de 2ª geração com até 48 núcleos de processamento modelo Silver 4210 de 2.20GHz e 24 slots DIMM DDR4 foi utilizado como servidor de orquestração de nuvem IaaS centralizado com a instalação do OpenNebula.

Toda estrutura recebeu alocação em um datacenter tier III com sistema de refrigeração que mantém os equipamentos a uma temperatura variando entre 20.0 à 21.5 graus Célsius,

dupla alimentação elétrica nos equipamentos computacionais quanto nos aparelhos de refrigeração e garantia de disponibilidade de 99,995%.

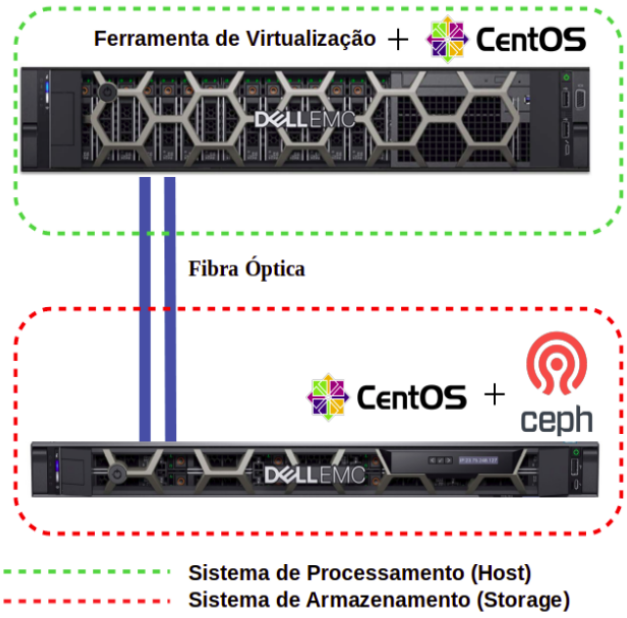

Figura 5.4: Conjunto - esquema lógico entre processamento e armazenamento
Fonte: Próprio Autor

Quatro conjuntos (um sistema de processamento e um sistema de armazenamento, conforme figura 5.4) foram desenvolvidos e previamente testados para evidenciar que quesitos como taxa de processamento, velocidade das interconexões, latência no transporte dos dados fossem da ordem de 5% de variação máxima entre elas. Tais testes serão evidenciados no próximo capítulo.

5.2 Ferramentas e Configurações

Em cada uma das infraestruturas, com exceção da plataforma de virtualização, foram instaladas as mesmas aplicações, plataformas e *softwares*.

5.2.1 Orquestração de Computação em Nuvem

Segundo Open Nebula (2020) o OpenNebula é uma solução *open source* desenvolvida no ano de 2005 por Ignacio Llorente e Ruben Montero, cujo código foi liberado no ano de 2008. Desenvolvido para gerenciamento de ambientes de computação em nuvem privadas, públicas e hibridas, o OpenNebula tem como objetivo uma gestão mais eficiente, simples e escalável de máquinas virtuais em infraestruturas centralizadas e distribuídas. Este orquestrador IaaS possui uma arquitetura composta por um único *host* que é responsável pela administração da nuvem e os outros *hosts* que são responsáveis pela orquestração da virtualização das máquinas virtuais.

O motivo de escolha do OpenNebula, diante de outras soluções IaaS conhecidas no mercado como OpenStack e CloudStack, é sua estrutura modular que possibilita uma

configuração e administração mais simples do que seus concorrentes. Além disso, o OpenNebula possui integração completa com as principais ferramentas de virtualização do mercado na atualidade: KVM, VMware vSphere e Xen Server, o que o faz ser a ferramenta mais recomendada para os testes desta obra.

Conforme Open Nebula (2020) sua estrutura é composta das seguintes ferramentas:

- ONED: gerenciamento de periféricos como rede, dispositivos de armazenamento, teclado e similares.
- MMSCHED: gerenciamento de (re)alocação de VMs entre os *hosts* físicos.
- ONECCTD: administração de contas e recursos de usuários.
- OCCI: gerencia as APIs que realizam comunicação com provedores externos como a Amazon EC2, na qual o OpenNebula possui integração.
- SUNSTONE: interface web gráfica de administração do OpenNebula.

Instalado na versão 5.10.1 de forma centralizada em um equipamento Dell PowerEdge R740, o OpenNebula é o responsável pela orquestração das três estruturas físicas montadas e seus respectivos virtualizadores. Como a administração deste orquestrador é independente dos nós, não é necessária à sua instalação em cada estrutura.

5.2.2 Orquestração de Armazenamento

De acordo com Aghayev et al. (2019) o Ceph foi arquitetado originalmente por Sage Weil em seus estudos para o doutorado. Sua primeira versão considerada estável incumbiu em 2012. Esta aplicação consiste em uma forma de armazenamento disseminado que fornece alta confiabilidade, escalabilidade e facilidade na gestão, tornando-se ideal para infraestruturas que lidam com uma grande quantidade de dados. Uma vantagem na utilização é a gratuidade do *software*. Fornece de maneira singular o armazenamento de blocos, arquivos e objetos com exclusividade.

Segundo Aghayev et al. (2019) esta ferramenta de armazenamento possui capacidade de replicar dados em um processo distribuído e torná-los tolerantes a falhas, podendo ser escalonados até o grau de *hexabyte*. Para tal procedimento são utilizados *hardwares* comuns que não exigem suporte original de fábrica. Em suma, a aplicação provê uma autocorreção, autogerenciamento, com o objetivo de mitigar tempo de gerenciamento e reduzir custos. Em sua composição existem três principais elementos, sendo eles: interface de comunicação aplicação de armazenamento, sistema de armazenagem para depositar dados e metadados, por fim, um *cluster* de servidores de metadados.

Quando se trata de arquiteturas comuns, os usuários se comunicam por meio de um elemento centralizado, por exemplo: *gateways,* corretores, APIs etc. Estes atuam como único meio de *input* para um subsistema de alta complexidade. Uma configuração nesses padrões delimita uma margem para desempenho e escalabilidade, introduzindo um único local de falha. Logo, se o elemento ceder, toda aplicação cairá também.

5.2.3 Virtualização

Para desenvolvimento dos testes foram configurados quatro ambientes virtualizados, que foram instalados respectivamente em cada uma das quatro estruturas físicas descritas anteriormente. Segue abaixo a descrição das mesmas e, posteriormente, a Tabela 7.1 demonstra a organização.

Virtualização Bare Metal com VMware vSphere Esxi

Desenvolvida para ter virtualização tradicional por *hardware*, foi instalado o VMware vSphere Esxi 6.7 como *hypervisor* no Dell PowerEdge R740 e o Ceph 14.2.0 como orquestrador de armazenamento no Dell PowerEdge R640, segregando o conjunto de armazenamento em dois grupos: exclusivamente HDD e exclusivamente SSD. A estrutura contou com orquestração de computação em nuvem pelo OpenNebula instalado em um servidor à parte da estrutura de testes.

Virtualização Por Emulação Com QEMU/KVM

Desenvolvida para ter virtualização tradicional por *software*, foi instalado o QEMU/KVM 4.2 como *hypervisor* no Dell PowerEdge R740 e o Ceph 14.2.0 como orquestrador de armazenamento no Dell PowerEdge R640, segregando o conjunto de armazenamento em dois grupos: exclusivamente HDD e exclusivamente SSD. A estrutura contou com orquestração de computação em nuvem pelo OpenNebula instalado em um servidor à parte da estrutura de testes.

Paravirtualização Com KVM e VirtIO

Desenvolvida para ter paravirtualização, foi instalado o KVM com *drivers* VirtIO 4.2 como *hypervisor* no Dell PowerEdge R740 e o Ceph 14.2.0 como orquestrador de armazenamento no Dell PowerEdge R640, segregando o conjunto de armazenamento em dois grupos: exclusivamente HDD e exclusivamente SSD. A estrutura contou com orquestração de computação em nuvem pelo OpenNebula instalado em um servidor à parte da estrutura de testes.

Virtualização Leve Com Docker

Desenvolvida para ter virtualização leve, foi instalado o Docker Engine 19 como pseudo *hypervisor* no Dell PowerEdge R740 e o Ceph 14.2.0 como orquestrador de armazenamento no Dell PowerEdge R640, segregando o conjunto de armazenamento em dois grupos: exclusivamente HDD e exclusivamente SSD. A estrutura contou com orquestração de computação em nuvem pelo OpenNebula instalado em um servidor à parte da estrutura de testes.

Tabela 5.1: Características das Estruturas De Testes

	VMware Esxi	**QEMU / KVM**	**KVM Com VirtIO**	**Docker Engine**
Versão	6.7	4.2	4.2	19.03.12
Licença	Proprietária	GLP (Aberta)	GLP (Aberta)	GLP (Aberta)
Modelo de Virtualização	Tradicional por Hardware Bare Metal	Tradicional por Software Emulada	Paravirtual	Virtualização Leve
Equipamento Físico	Dell PowerEdge R740 e Dell PowerEdge R640	Dell PowerEdge R740 e Dell PowerEdge R640	Dell PowerEdge R740 e Dell PowerEdge R640	Dell PowerEdge R740 e Dell PowerEdge R640
Storage	Ceph 14.2.0	Ceph 14.2.0	Ceph 14.2.0	Ceph 14.2.0
S.O Hospedeiro	VMware vCenter Server	Red Hat Enterprise Linux CentOS 7	Red Hat Enterprise Linux CentOS 7	Red Hat Enterprise Linux CentOS 7
S.O Convidado	Red Hat Enterprise Linux CentOS 7	Red Hat Enterprise Linux CentOS 7	Red Hat Enterprise Linux CentOS 7	Red Hat Enterprise Linux CentOS 7

Fonte: Próprio Autor

5.3 Modelos de Testes

Conforme explanado anteriormente, a primeira etapa dos testes é analisar o desempenho de máquinas virtuais à nível de infraestrutura em um mesmo ambiente de computação em nuvem. Isto é, realizar uma análise dos principais fatores que contribuem para o desempenho de uma VM: processamento (CPU e armazenamento primário (memória RAM)), armazenamento secundário (disco) e periféricos (rede, teclado, mouse, etc.).

Para os testes foi definido como base o modelo IaaS de orquestração de nuvem. Foram utilizadas máquinas virtuais Red Hat Enterprise Linux CentOS 7 orquestradas pelo Open Nebula na mesma infraestrutura física diferenciado apenas pelo paradigma de virtuali-

zação (bare metal, emulação, paravirtualização e virtualização leve) e seus respectivos virtualizadores.

5.3.1 Processamento

Tal análise consiste em observar, quantificar e classificar a taxa de processamento dos componentes CPU e armazenamento primário (memória RAM) virtualizados pelos paradigmas descritos anteriormente. De acordo com UFPE (2006) os testes de desempenho de processamento mais conhecidos são:

- Teste de avaliação: compara os resultados obtidos nos testes com parâmetros pré estabelecidos em ambientes controlados.
- Teste de contenção: analisa se o objetivo a ser testado pode lidar com as demandas simultâneas de vários agentes em um mesmo recurso.
- Teste de carga: observa o desempenho do objeto do teste sob os mais diferentes tipos de trabalho de processamento dentro de um delta definido para condições normais de trabalho.
- Teste de stress: com os mesmos objetivos do teste anterior, este verifica o comportamento de desempenho do objetivo do teste quando exposto em condições de cargas de processamento além do definido.

Nesta obra, que visa analisar o poder de processamento de VMs em diferentes paradigmas de virtualização, os testes realizados são: testes de carga e de stress.

5.3.2 Armazenamento

Para realizar uma análise completa do desempenho de dispositivos de armazenamento secundário (popularmente conhecidos como disco) de uma VM, é necessário observar, quantificar e classificar a taxa de leitura e escrita de dados, além de latência/atraso no armazenamento virtualizado pelos paradigmas descritos anteriormente.

Seguindo a linha de pesquisa do trabalho de Li et al. (2017), nos testes de dispositivos de armazenamento em VM é necessária a coleta de três tipos de dados distintos: *Input/Output per Second* (IOPS), calculado como o número de operações de entrada e saída por segundo aplicada sobre dispositivos de armazenamento, velocidade de transmissão de blocos de dados, que se calcula com a taxa de transferência destes blocos em kB/s, MB/s, GB/s, TB/s, dentre outros. E latência de transmissão calculada em milissegundos. Com tais métricas é possível determinar detalhadamente o desempenho de um dispositivo de armazenamento (LI et al., 2017).

Os dados foram coletados através de média simples de dez execuções do conjunto de *benchmarks* e estes possuem intervalo de confiança de 95% .

5.3.3 Ferramentas e Benchmarks

Esta sessão é reservada para explanar quais são as ferramentas e *benchmarks* utilizados para a análise de desempenho em VMs em ambiente de computação em nuvem.

Li et al. (2017) afirma que a adoção de uma ferramenta de *benchmark* capaz de realizar testes mais precisos depende de um único fator: simplicidade. Quanto menos complexa e mais leve na execução do sistema operacional a ferramenta for, menos ela impactará nos resultados aumentando, assim, a precisão na coleta.

Segue a seguir a estruturação de cada tipo de análise: processamento (CPU e memória RAM) e armazenamento ("disco") virtualizados.

Análise de Processamento

Pinto, Schnorr e Legrand (2017) afirmam que aferir o desempenho de processamento de forma adequada de uma estrutura através de uma métrica específica é complexo. Diferente de uma medição de dispositivos de armazenamento, onde os parâmetros de desempenho são facilmente quantificáveis, a análise do poder computacional da CPU e da memória RAM não pode ser realizada por métodos de análise de consumo simples. A dinâmica do processamento de uma determinada aplicação envolve diversos fatores como: fila de processamento, paralelismo, I/O de comunicação entre partes processadas e outros. Portanto, quaisquer análises que se pautem em dados como estes não possuem boa confiabilidade.

O método mais utilizado para testar o desempenho de processamento é a análise por tempo de resposta (ATR). Esta é uma abordagem eficaz para o estudo de desempenho de uma infraestrutura física ou virtualizada (PINTO; SCHNORR; LEGRAND, 2017).

Além do parâmetro de simplicidade da ferramenta de testes, Li et al. (2017) sugere que os testes de análise de CPU e memória RAM sejam realizados através diversos testes cuja carga de trabalho seja distribuída igualmente (sem grandes variações no processamento) e esta mesma carga seja imputada por períodos distintos (desde milésimos de segundo à dias ou semanas). Para tanto, testes de processamento que obedecem a esses requisitos se utilizem cálculos matemáticos de baixa complexidade, mas de grande esforço de CPU e memória RAM como, por exemplo, cálculo da sequência Fibonacci, exponenciais e outros (LI et al., 2017).

Conforme já explanado anteriormente, os testes realizados nesta obra são de carga e de stress em níveis distintos. Assim, o elemento de métrica adequado para tais testes será a ATR e, consequentemente, todos os dados obtidos serão na unidade de medida de tempo no Sistema Internacional de Unidades: segundo(s) com acurácia de 95%.

Para realizar análise de desempenho de processamento foram desenvolvidos e executados dois códigos Python versão 3.6 com objetivo de realizar análise de stress de CPU com uma sequência de operações de multiplicação não paralelizáveis. A adoção da linguagem Python se deve principalmente porque o *typing system* na versão 3.6 é estável, de melhor performance em comparação com versões anteriores como ferramentas como: *core support* e *forward references*.

O diferencial desta obra nos testes de CPU e memória RAM está na segregação nos testes de *stress* em dois cenários: o primeiro é um cenário de alto consumo de CPU e baixo consumo de memória RAM, enquanto o segundo é um cenário de alto consumo de CPU

e memória RAM. Desta forma, é possível observar com precisão o comportamento das ferramentas de virtualização em cenários de diferentes tipos de carga de processamento sem incorrer em adicionar variáveis indesejadas aos *benchmarks*, melhorando, assim, o que Li et al. (2017) e demais realizaram até o momento.

Stress de CPU

O primeiro código consiste em realizar stress apenas em CPU, com baixa carga na memória RAM. Para isso, o código gerado seguiu a linha de Li et al. (2017) onde realiza a exponencial com uma base inteira fixa: 2 (dois) e um expoente variável inteiro e maior que zero. Além disso, o código possui a função "*time*"que calcula a hora que a função de exponenciação foi iniciada e terminada para, assim, com a subtração tempo final e tempo inicial, indicar com precisão o tempo de execução. Segue o código em português estruturado abaixo:

Início do Código

```
importa biblioteca de contador de tempo

variável 'c' recebe em inteiro com o seguinte texto: "Número de Operações: "

variável 'a' recebe o inicio do contador do tempo

variável 'var' recebe 2 elevado à variável 'c'

exibir variável 'var'

variável 'b' recebe contador final do tempo

exibir resultado de b - a
```

Fim do Código

Segue o código em Python 3 abaixo:

Begin

```
import time

c = int(input("Numero de Execucoes: "))

a = time.time()
```

```
var=2**c

print(var)

b = time.time()

print (b -a)
```

End

Stress de CPU e Memória RAM

O segundo código consiste em realizar stress em CPU acrescida de escrita de dados em memória RAM. Para isso, o código gerado popula um vetor com um número alto de índices com variáveis inteiras, randômicas no range entre "zero"e "um". Além disso, o código possui a função "time"que calcula a hora que a função foi iniciada e terminada para, assim, com a subtração tempo final e tempo inicial, indicar com precisão o tempo de execução. Segue o código em português estruturado abaixo:

Início do Código

```
importa biblioteca 'randômica'

importa biblioteca de contador de tempo

variável 'c' recebe em inteiro com o seguinte texto: "Número de Operações: "

variável 'saída' recebe o vetor com os valores randômicos

    para valores onde o range é variável 'c' fazer:

    variável 'a' recebe o inicio do contador do tempo

vetor é preenchido com os valores randômicos entre zero e um

exibir variável 'saída'

variável 'b' recebe contador final do tempo

exibir resultado de b - a
```

Fim do Código

Segue o código em Python 3 abaixo:

Begin

```
import random

import time

c = int(input("Número de Operações: "))

saida = []

    for _ in range(c):

    a = time.time()

saida.append(random.randint(0, 1))

print(saida)

b = time.time()

print(b - a)
```

End

Para maiores informações sobre os códigos utilizados, vide Apêndice A.

Ambiente de Testes de CPU e Memória RAM

Para os testes com CPU e memória RAM virtualizados foram criadas quatro máquinas virtuais estruturalmente idênticas, Linux CentOS 7, com 1 CPU, 1 GB de memória e 100 GB de dispositivo de armazenamento, com Python 3.6 instalado e dispostas da seguinte ordem:

- A primeira virtualizada no VMware vSphere, em ambiente bare metal, orquestrada pelo OpenNebula, paradigma de armazenamento físico com Ceph em HDD e armazenamento lógico SCSI. Para comunicação com o hardware a VM executa o VMware Tools 6.7.
- A segunda virtualizada no QEMU/KVM em ambiente emulado, orquestrada pelo OpenNebula, com paradigma de armazenamento físico com Ceph em HDD e armazenamento lógico SCSI. Para comunicação com o *hardware* a VM executa o Qemu Guest Agent 4.

- A terceira virtualizada no KVM com o driver VirtIO em ambiente paravirtualizado, orquestrada pelo OpenNebula, com paradigma de armazenamento físico com Ceph em HDD e armazenamento lógico SCSI. Para comunicação com o *hardwar*e a VM executa o VirtIO Drivers.
- A quarta virtualizada no Docker, em ambiente de contêineres, orquestrada pelo OpenNebula, com paradigma de armazenamento físico com Ceph em HDD e armazenamento lógico SCSI. Para comunicação com o *hardware* o sistema de conteineres executa o Docker Engine 19.03.12.

Conforme explanado anteriormente, para esta análise será utilizado o teste de *stress* controlado através de cálculo de exponenciais cujos períodos variam entre milésimos de segundo a semanas de execução. Três tipos de dados foram coletado durante o experimento: tempo de resposta em segundos ou seus correspondentes mínimo, tempo de resposta em segundos ou seus correspondentes máximo e tempo de resposta em segundos ou seus correspondentes médio.

Análise de Armazenamento

Observando os requisitos propostos por Li et al. (2017), para realizar análise de desempenho de dispositivo de armazenamento foram escolhidas duas ferramentas de *benchmarks*: Fio e IOping. Segue abaixo a explanação de ambas.

Flexible I/O

Criado por Jens Axboe, o Flexible I/O (Fio) foi originalmente desenvolvido para realizar testes de desempenho em cargas de trabalho em ambientes Linux. O Fio é capaz de simular uma determinada carga de trabalho de entrada e saída do dispositivo de armazenamento sem a necessidade de se desenvolver um teste personalizado para cada cenário.

Responsável por realizar as medições de IOPS e velocidade de transmissão de blocos de dados de tamanhos específicos ou aleatórios, o Fio gera um determinado número de processos similares mediante execução de ações de entrada e saída do dispositivo de armazenamento conforme especificação do usuário. Desta forma é possível simular, próximo à realidade, diversos ambientes com tipos de cargas de entrada e saída em dispositivo de armazenamento sem que outros fatores, como cargas de trabalho paralelas ou atividades indesejadas ao teste, influenciem nos resultados.

De acordo com os testes conduzidos por Li et al. (2017), para uma verificação mais assertiva do desempenho de dispositivos de armazenamento é necessária a adoção de um padrão para os testes. Os autores sugerem que seja criado um arquivo com o dobro do tamanho total da memória RAM da VM para obtenção de bons resultados. Este trabalho expandiu os critérios e utilizou um arquivo quatro vezes superior à memória RAM de cada VM: um arquivo de 4 GB em VMs com 1 GB de memória RAM. A fim de reduzir a possibilidade de inconsistência dos dados obtidos nos testes, ao invés de segregar o bloco de 4 GB em blocos de tamanho e quantidades aleatórios, foram utilizados múltiplos de 4 GB, respectivamente blocos de 4 kB e 4 MB para manter a padronização no tamanho e quantidade de blocos trafegados, reduzindo, assim, a possibilidade de desvio da acurácia.

O Fio foi configurado para gerar estes cenários através dos seguintes comandos:

Testes de 4kB:

Início do Código

```
fio --randrepeat=1 --ioengine=libaio --name=test1 --filename=a --bs=4k
--size=4G
```

Fim do Código

Testes de 4MB:

Início do Código

```
fio --randrepeat=1 --ioengine=libaio --name=test2 --filename=b --bs=4096k
--size=4G
```

Fim do Código

Onde os parâmetros são:

randrepeat: quantidade de execuções do teste;

ioengine: tipo de biblioteca. Neste caso, a biblioteca é a libaio;

name: nome do teste;

filename: nome do arquivo gerado;

bs: tamanho do bloco de dados do arquivo gerado;

size: tamanho do arquivo;

IOping

IOping é uma ferramenta para medição do tempo de resposta de entradas e saídas do dispositivo de armazenamento em tempo real. Este é semelhante ao *ping*, porém ao invés de mostrar a latência de rede, mostra a latência em operações de IOPS do dispositivo de armazenamento da mesma forma em que o*ping* mostra a latência da rede. O IOping é executado em paralelo ao *benchmark* de medição de desempenho para obtenção de resultados de latência com determinadas cargas de trabalho no dispositivo de armazenamento. Para executar o teste de latência foi executado o seguinte comando:

```
ioping -c10
```

Ambiente de Testes de Armazenamento Secundário (Disco)

Para os testes com armazenamento virtualizado foram criadas oito máquinas virtuais estruturalmente idênticas, Linux CentOS 7, com 1 CPU, 1 GB de memória RAM e 100 GB de armazenamento, todas orquestradas pelo OpenNebula, dispostas da seguinte ordem:

- A primeira no (VMware vSphere (bare metal)), paradigma de armazenamento físico com Ceph em HDD e armazenamento lógico SCSI. Para comunicação com o *hardware* a VM executa o VMware Tools 6.7.
- A segunda no (VMware vSphere (bare metal)), paradigma de armazenamento físico com Ceph em SSD e armazenamento lógico SCSI. Para comunicação com o *hardware* a VM executa o VMware Tools 6.7.
- A terceira no (QEMU/KVM (emulação)), com paradigma de armazenamento físico com Ceph em HDD e armazenamento lógico SCSI. Para comunicação com o *hardware* a VM executa o Qemu Guest Agent 4.
- A quarta no (QEMU/KVM (emulação)), com paradigma de armazenamento físico com Ceph em SSD e armazenamento lógico SCSI. Para comunicação com o *hardware* a VM executa o Qemu Guest Agent 4.
- A quinta no KVM utilizando VirtIO (paravirtualização), com paradigma de armazenamento físico com Ceph em HDD e armazenamento lógico SCSI. Para comunicação com o *hardware* a VM executa o VirtIO Drivers.
- A sexta no KVM com o driver VirtIO (paravirtualização), com paradigma de armazenamento físico com Ceph em SSD e armazenamento lógico SCSI. Para comunicação com o *hardware* a VM executa o VirtIO Drivers.
- A sétima em Docker (contêiner), com paradigma de armazenamento físico com Ceph em HDD e armazenamento lógico SCSI. Para comunicação com o *hardware* a VM executa o Docker Engine 19.03.12.
- A oitava em Docker (contêiner), com paradigma de armazenamento físico com Ceph em SSD e armazenamento lógico SCSI. Para comunicação com o *hardware* a VM executa o Docker Engine 19.03.12.

Capítulo 6

Testes de Controle

Conforme Li et al. (2017), Xavier et al. (2013) e Hwang et al. (2013) em uma experiência controlada deste porte onde serão comparados desempenhos de diversos tipos de ambientes deve haver a existência do denominado teste de controle. No caso deste *benchmark* as ferramentas citadas no capítulo anterior serão testadas primariamente na estrutura desenvolvida sem quaisquer modelos de virtualização ativos, ou seja, os testes de CPU, RAM e armazenamento serão aferidos primariamente no próprio *hardware*. Tal experimentação, que não sofre nenhuma intervenção de métodos de virtualização, serve como elemento de comparação com o teste experimental, no qual serão introduzidos os ambientes virtualizados.

Tal processo serve para corroborar dois grandes fatores: analisar a correlação entre desempenho real e desempenho virtualizado e certificar que, com exceção dos modelos de virtualização, as demais condições são mantidas iguais durante todo o procedimento, possibilitando evidenciar as diferenças constatadas, ou não, no final do *benchmark*.

6.1 Validação da Infraestrutura

Conforme explanado em capítulos anteriores, sob apoio de Hwang et al. (2013) e Xavier et al. (2013), a estrutura desenvolvida para os testes tem como objetivo reproduzir, de maneira igualitária, o *hardware* e a orquestração de nuvem computacional com exceção do mecanismo virtualizador e seu respectivo paradigma de virtualização. Devido à natureza dos testes a serem conduzidos fez-se necessário que a estrutura como um todo fosse validada em um teste de controle de desempenho sem quaisquer elementos de virtualização. Tal teste serve para corroborar que a infraestrutura está com desempenho semelhante e que quaisquer modificações em testes com os virtualizadores apontem para que estas não sejam falsos-positivos advindos de alterações da estrutura física.

Como se trata de quatro estruturas distintas que necessitam ser estritamente iguais em questão de desempenho foram realizadas a execução dos *benchmarks* de desempenho diretamente sobre o *hardware* verificando suas métricas de desempenho, esperando que estas apresentassem desempenho similar com uma acurácia mínima de 95%.

Além disso, um teste complementar foi realizado: as estruturas foram realocadas de modo que equipamentos de armazenamento pudesse ter seus *hosts* invertidos com o intuito

de verificar se equipamentos físicos diferentes poderiam manter um resultado dentro da assertividade prevista.

É valido ressaltar que ambas estruturas (principal e alternativa) são arranjos representativos compostos de quatro conjuntos de máquinas físicas (um *host* e um *storage*) cada. Portanto uma composição de máquinas físicas (um Dell R740 e um Dell R640) formam um conjunto. Quatro conjuntos formam uma estrutura. A estrutura possui um cálculo de média simples observando o desvio padrão dos conjuntos que a compõe para servir de base para a comparação com os elementos de virtualização. A subdivisão entre os conjuntos é observada nas tabelas 6.1 e 6.2 para as estruturas principal e alternativa, respectivamente.

Tabela 6.1: Estrutura Principal - Composição dos Conjuntos

Conjunto I	Conjunto II	Conjunto III	Conjunto IV
Host 1 e Storage 1	Host 2 e Storage 2	Host 3 e Storage 3	Host 4 e Storage 4

Fonte: Próprio Autor

A Tabela 6.1 possui a composição é utilizada como padrão para esta obra.

Tabela 6.2: Estrutura Alternativa - Composição dos Conjuntos

Conjunto I	Conjunto II	Conjunto III	Conjunto IV
Host 1 e Storage 2	Host 2 e Storage 4	Host 3 e Storage 3	Host 4 e Storage 1

Fonte: Próprio Autor

A tabela 6.2 é utilizada como alternativa para esta obra com o objetivo de evidenciar que a troca de máquinas dos conjuntos é irrelevante ao desempenho de uma estrutura como um todo.

6.2 Processamento

Conforme explanado anteriormente, para esta análise será utilizado o teste de stress controlado através de cálculo de exponenciais cujos períodos variam entre milésimos de segundos a semanas de execução. Cinco tipos de dados foram coletados durante o experimento: tempo de resposta em segundos ou seus correspondentes mínimos, tempo de resposta em segundos ou seus correspondentes máximo e tempo de resposta em segundos ou seus correspondentes médios.

Os experimentos foram realizados com dois códigos distintos: stress apenas para CPU e stress para CPU e memória RAM. A primeira rodada de testes consistiu em executar dez vezes cada código em cada uma das quatro VMs reiniciando-as a cada teste para limpeza dos registros de memória.

6.2.1 Resultados

É valido ressaltar que, os gráficos de resultados originalmente são em forma de hipérboles. Entretanto para fins de melhor observação, o eixo y dos gráficos foi convertido em escala logarítmica.

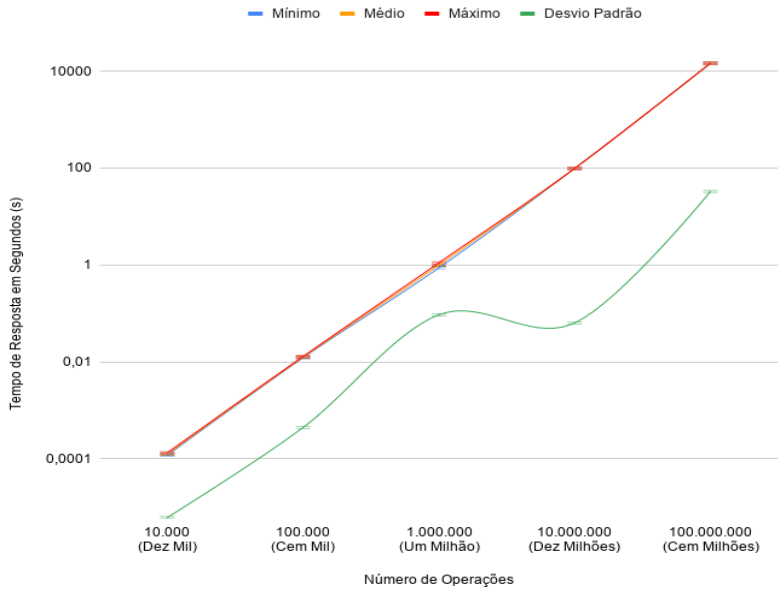

Figura 6.1: Desvio Padrão na Estrutura Principal - Teste de CPU
Fonte: Próprio Autor

O teste da figura 6.1 aplicou o benchmark apenas de CPU na estrutura principal. É possível observar que ao se aplicarem testes exclusivos de processamento sem stress à memória RAM não existe uma variação visível da composição dos dados ao longo da quantidade de operações. Além disso, os dados mínimos e máximos deste teste possuem pouca variância entre si. O maior índice de desvio foi observado próximo de um milhão de operações, onde a linha de desvio se aproxima mais da linha dos dados dos testes.

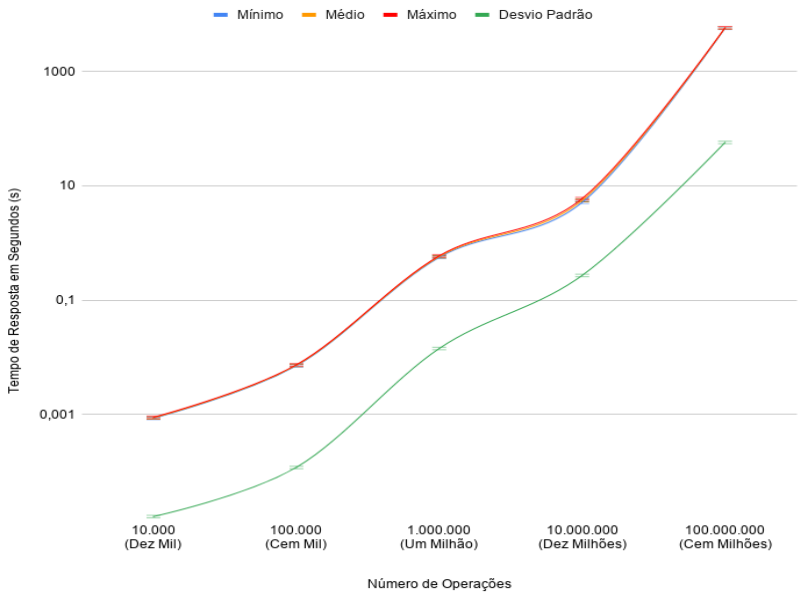

Figura 6.2: Desvio Padrão na Estrutura Principal - Teste de CPU e RAM
Fonte: Próprio Autor

O teste da figura 6.2 aplicou o benchmark de CPU e RAM na estrutura principal. É possível observar que ao se aplicarem testes que incluem escritas e leituras em vetores na memória RAM existe uma variação visível da composição dos dados ao longo da quantidade de operações. Entretanto, os dados mínimos e máximos deste teste possuem pouca variância entre si. O maior índice de desvio foi observado próximo de dez milhões de operações, onde a linha de desvio se aproxima mais da linha dos dados dos testes.

ESTRUTURAS ALTERNATIVAS

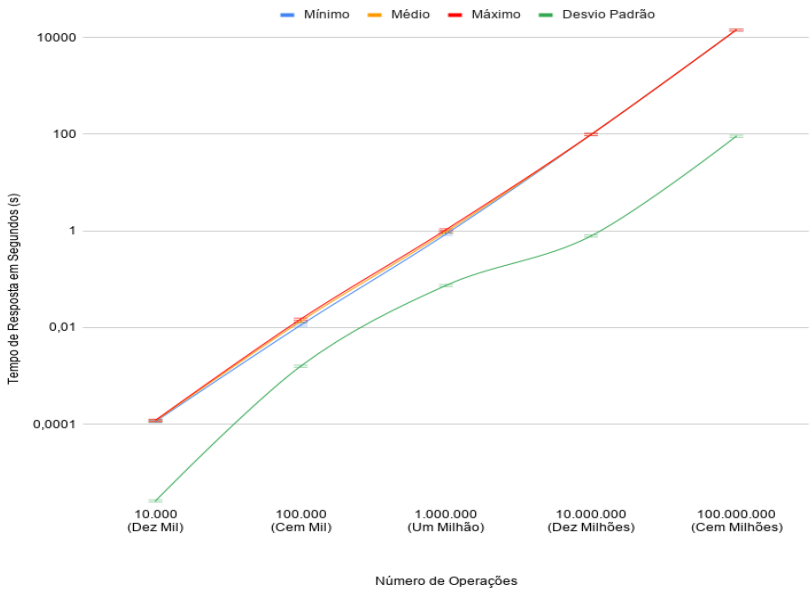

Figura 6.3: Desvio Padrão na Estrutura Alternativa - Teste de CPU
Fonte: Próprio Autor

O teste da figura 6.3 aplicou o benchmark apenas de CPU na estrutura alternativa. É possível observar que ao se aplicarem testes exclusivos de processamento sem stress à memória RAM não existe uma variação visível da composição dos dados ao longo da quantidade de operações como observado na estrutura principal. Além disso, os dados mínimos e máximos deste teste possuem pouca variância entre si, conforme observado em sua estrutura irmã. O maior índice de desvio, no entanto, foi observado cerca de cem mil à um milhão de operações, onde a linha de desvio se aproxima mais da linha dos dados dos testes.

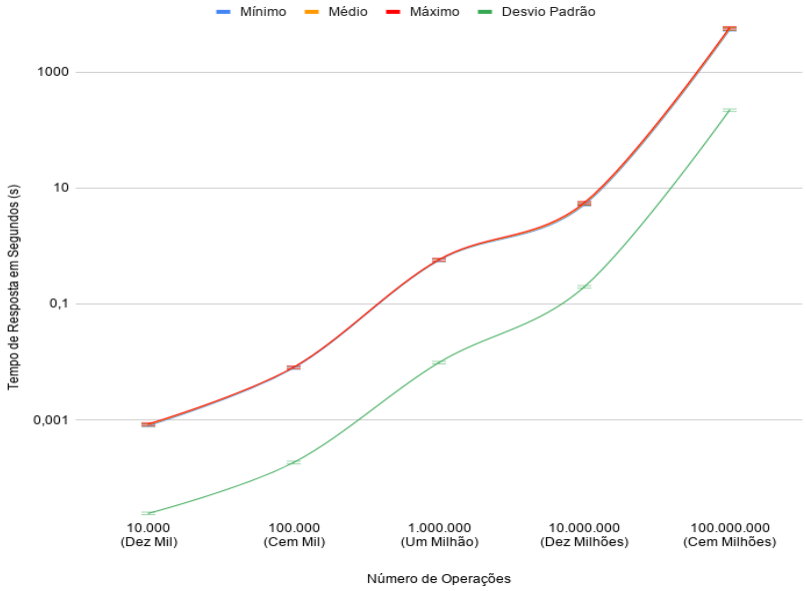

Figura 6.4: Desvio Padrão na Estrutura Alternativa - Teste de CPU e RAM
Fonte: Próprio Autor

O teste da figura 6.4 aplicou o benchmark de CPU e RAM na estrutura alternativa. É possível observar que ao se aplicarem testes que incluem escritas e leituras em vetores na memória RAM existe uma variação visível da composição dos dados ao longo da quantidade de operações conforme observado na estrutura principal. Os dados mínimos e máximos deste teste possuem pouca variância entre si como observado com sua estrutura irmã. O maior índice de desvio também foi observado próximo de dez milhões de operações, onde a linha de desvio se aproxima mais da linha dos dados dos testes.

6.2.2 Análise

Claramente é possível observar que as estruturas (principal e alternativa) possuem os mesmos comportamentos quando aplicados testes de CPU e CPU e RAM. O desvio geral das estruturas ficou abaixo de 3%. Tal dado inibe quaisquer fatores de *hardware* e oscilações de desempenho da estrutura em si possa influenciar significativamente nos testes posteriores com os elementos de virtualização.

6.3 Armazenamento

Três tipos de dados foram coletados durante o experimento: IOPS, velocidade de transmissão em MB/s e latência (popularmente conhecida como atraso) calculada por milissegundo (ms). As duas primeiras coletadas com o *software* "Fio", enquanto a terceira foi simultaneamente coletada pela aplicação "IOping". Os experimentos foram realizados com dois blocos de dados de tamanhos distintos: 4kB e 4MB.

6.3.1 Resultados em Hard Disk Drive (HDD)

Os testes em HDD consistiram em executar dez vezes o conjunto *benchmark* em cada um dos quatro conjuntos de máquinas físicas que constituem a estrutura principal utilizando blocos de 4kB, executar em cada um dos quatro conjuntos de máquinas físicas que constituem a estrutura alternativa e, posteriormente, executar o mesmo processo com blocos de 4 MB.

A figura 6.5 demonstra os principais resultados para blocos de 4 kB.

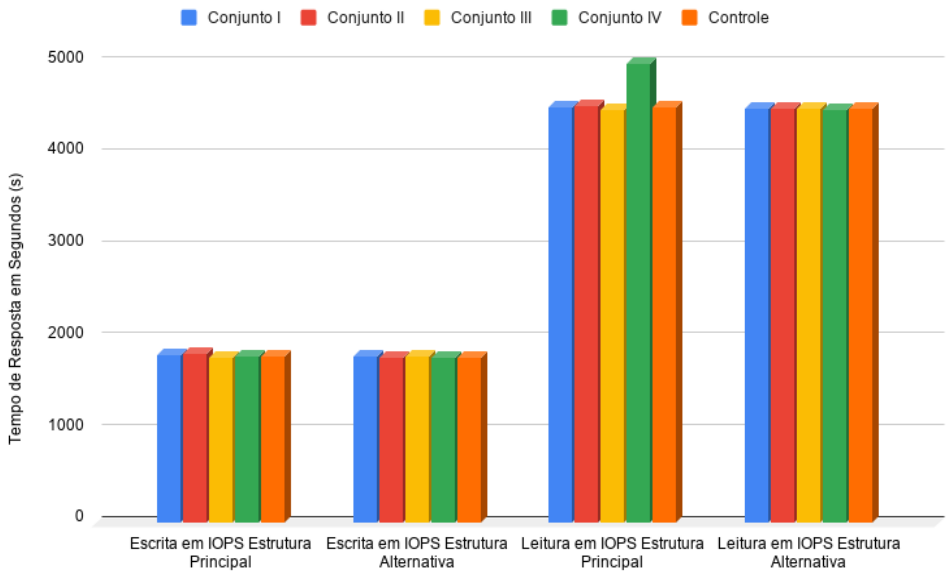

Figura 6.5: Comparativo Escrita e Leitura em HDD - Blocos 4 kB - Controle
Fonte: Próprio Autor

A figura 6.6 demonstra os principais resultados para blocos de 4 MB.

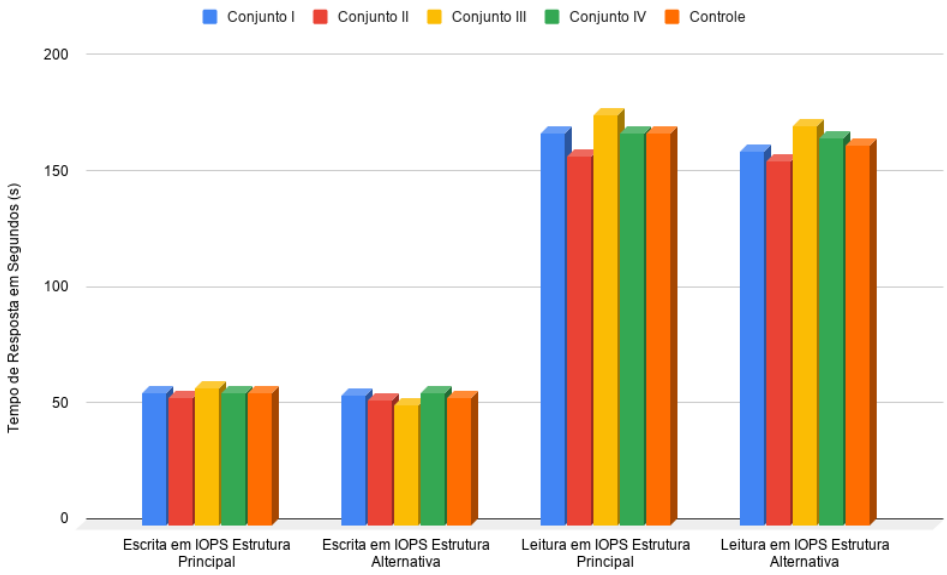

Figura 6.6: Comparativo Escrita e Leitura em HDD - Blocos 4 MB - Controle
Fonte: Próprio Autor

6.3.2 Resultados em Solid-State Drive (SSD)

Os testes em SSD consistiram em executar dez vezes o conjunto de *benchmark* em cada um dos quatro conjuntos de máquinas físicas que constituem a estrutura principal utilizando blocos de 4kB, executar em cada um dos quatro conjuntos de máquinas físicas que constituem a estrutura alternativa e, posteriormente, executar o processo com blocos de 4 MB.

A figura 6.7 demonstra os principais resultados para blocos de 4 kB.

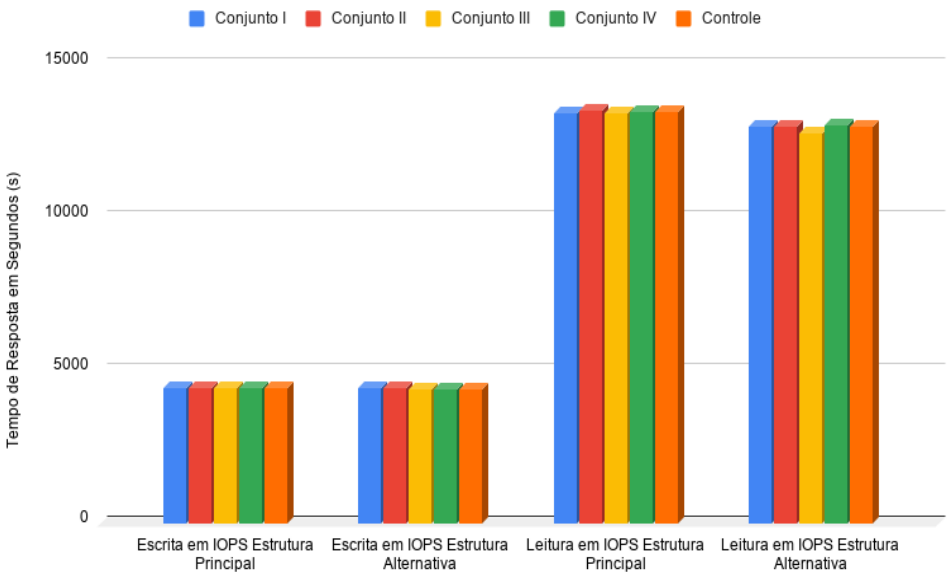

Figura 6.7: Comparativo Escrita e Leitura em SSD - Blocos 4 kB - Controle
Fonte: Próprio Autor

A figura 6.8 demonstra os principais resultados para blocos de 4 kB.

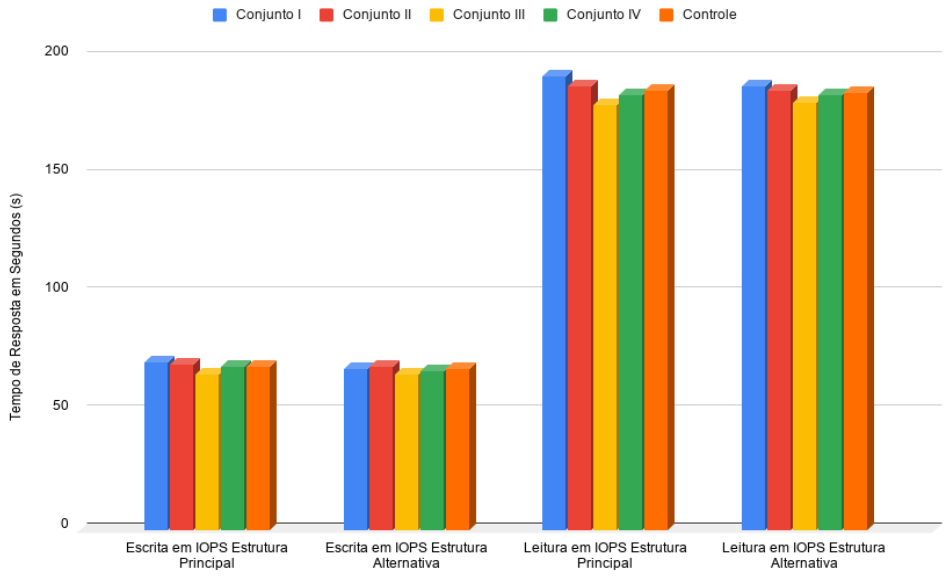

Figura 6.8: Comparativo Escrita e Leitura em SSD - Blocos 4 MB - Controle
Fonte: Próprio Autor

6.3.3 Análise

É possível observar que o desempenho do SSD de modo geral foi superior ao HD na ordem de 3,5 vezes para blocos de 4kB e 4MB.

Foi calculada média simples quatro estruturas para definir os dados de controle principal e alternativo. Em todos os casos, o desvio padrão dos dados das estruturas entre si ficaram oscilando entre 1,5% e 5% conforme também observado por Hwang et al. (2013). Tal dado indica que a acurácia do ambiente de armazenamento via Ceph está de acordo com o previsto e quaisquer variações de desempenho que existirem à nível físico são pouco influentes aos modelos de virtualização.

Capítulo 7

Resultados

7.1 Processamento

Os experimentos foram realizados com dois códigos distintos: *stress* apenas para CPU e *stress* para CPU e memória RAM. A primeira rodada de testes consistiu em executar dez vezes cada código em cada uma das quatro VMs reiniciando-as a cada teste para limpeza dos registros de memória. Nestes experimentos foram coletados os tempos de resposta em segundos como dados para os comparativos.

7.1.1 Teste de Processamento - Apenas CPU

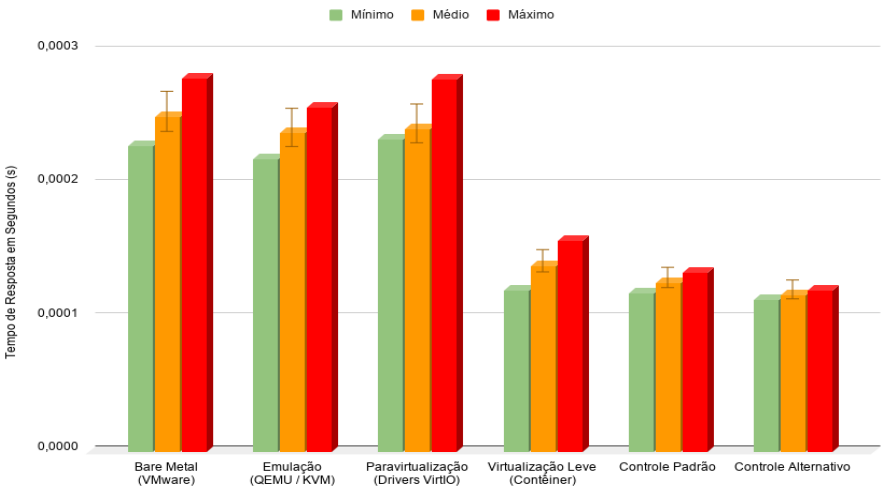

Figura 7.1: Teste Apenas CPU - Dez Mil Operações
Fonte: Próprio Autor

Observa-se que em dez mil operações, figura 7.1, a virtualização leve é a com o menor tempo de resposta (melhor performance) diante dos controles, com cerca de 9,8% de tempo de resposta superior ao controle. Em seguida, a emulação é cotada com cerca de 88,9%. A paravirtualização se encontra com cerca de 91,3%. Por último, bare metal é cotada com 98,4% de tempo de resposta superior ao controle.

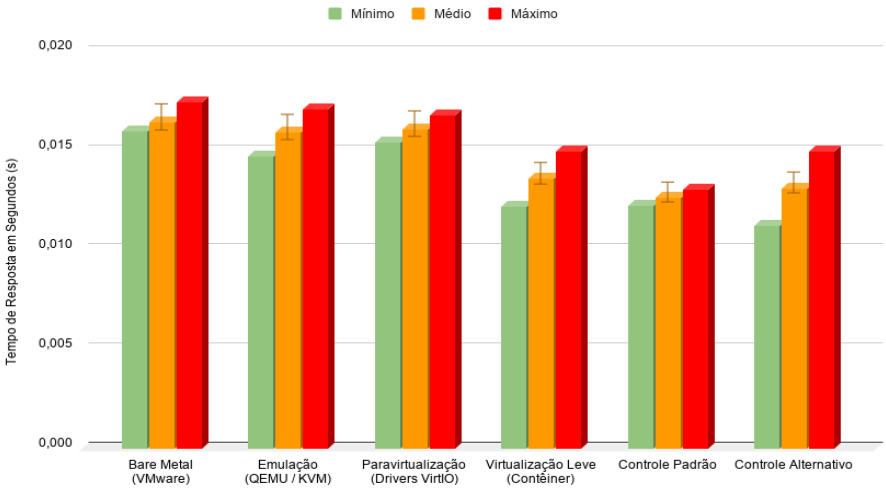

Figura 7.2: Teste Apenas CPU - Cem Mil Operações
Fonte: Próprio Autor

Observa-se que em cem mil operações, figura 7.2, a virtualização leve é a com o menor tempo de resposta (melhor performance) diante dos controles, com cerca de 7,49% de tempo de resposta superior ao controle. Em seguida, a emulação é cotada com cerca de 25,96%. A paravirtualização se encontra com cerca de 27,31%. Por último, bare metal é cotada com 30% de tempo de resposta superior ao controle.

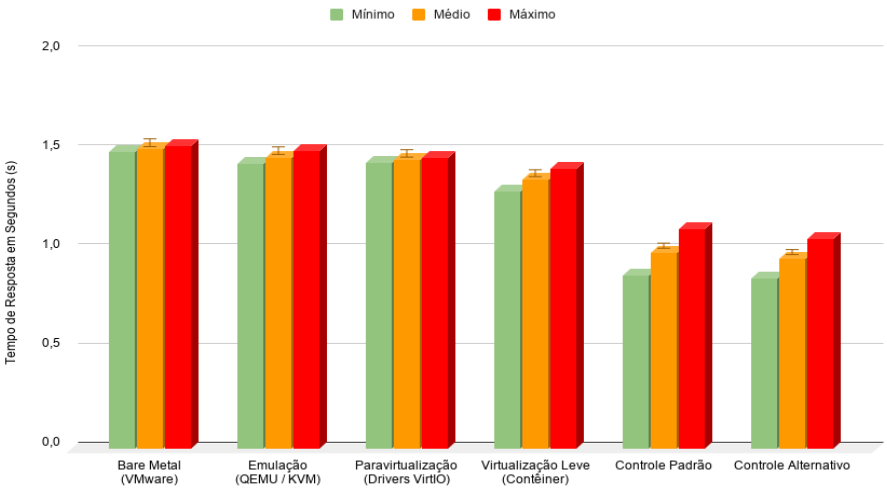

Figura 7.3: Teste Apenas CPU - Um Milhão de Operações
Fonte: Próprio Autor

Observa-se que em um milhão de operações, figura 7.3, a virtualização leve é a com o menor tempo de resposta (melhor performance) diante dos controles, com cerca de 36,91% de tempo de resposta superior ao controle. Em seguida, a paravirtualização é cotada com cerca de 46,99%. A emulação se encontra com cerca de 48,41%. Por último, bare metal é cotada com 52,44% de tempo de resposta superior ao controle. É possível observar que os

desvios entre mínimos e máximos dos paradigmas de virtualização diminuíram em relação à execução anterior, enquanto os modelos de controle possuíram aumento no desvio de mínimos e máximos.

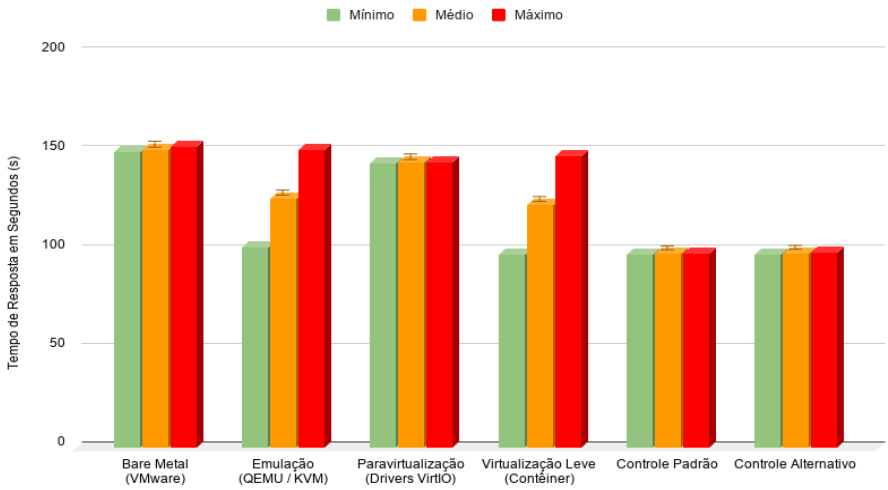

Figura 7.4: Teste Apenas CPU - Dez Milhões de Operações
Fonte: Próprio Autor

Observa-se que em dez milhões de operações, figura 7.4, a virtualização leve é a com o menor tempo de resposta (melhor performance) diante dos controles, com cerca de 25,22% de tempo de resposta superior ao controle. Em seguida, a emulação é cotada com cerca de 28,53%. A paravirtualização se encontra com cerca de 46,99%. Por último, bare metal é cotada com 53,5% de tempo de resposta superior ao controle. É possível observar que os desvios mínimos e máximos da bare metal e da paravirtualização são quase nulos, similares ao controle.

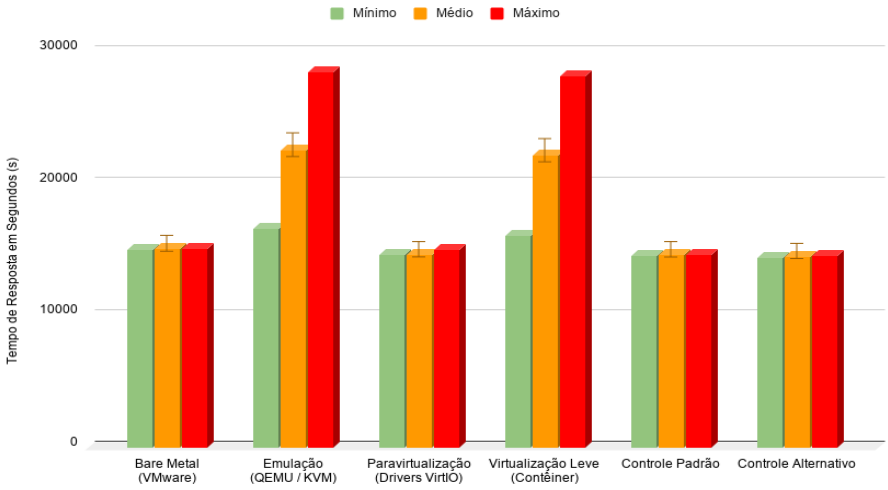

Figura 7.5: Teste Apenas CPU - Cem Milhões de Operações
Fonte: Próprio Autor

Em cem milhões de operações, figura 7.5, existe uma inversão completa no cenário inicial: a paravirtualização é o paradigma com o menor tempo de resposta (melhor performance) diante dos controles, com cerca de 0,003% em relação ao controle e apresenta baixo desvio. Em seguida, bare metal é cotada com cerca de 3,12% de desempenho inferior ao controle, também apresentando comportamento de baixo desvio. A virtualização leve e a emulação apresentaram os piores desempenhos com 51,6% e 54,47%, respectivamente. Além disso, ambas apresentam alto desvio.

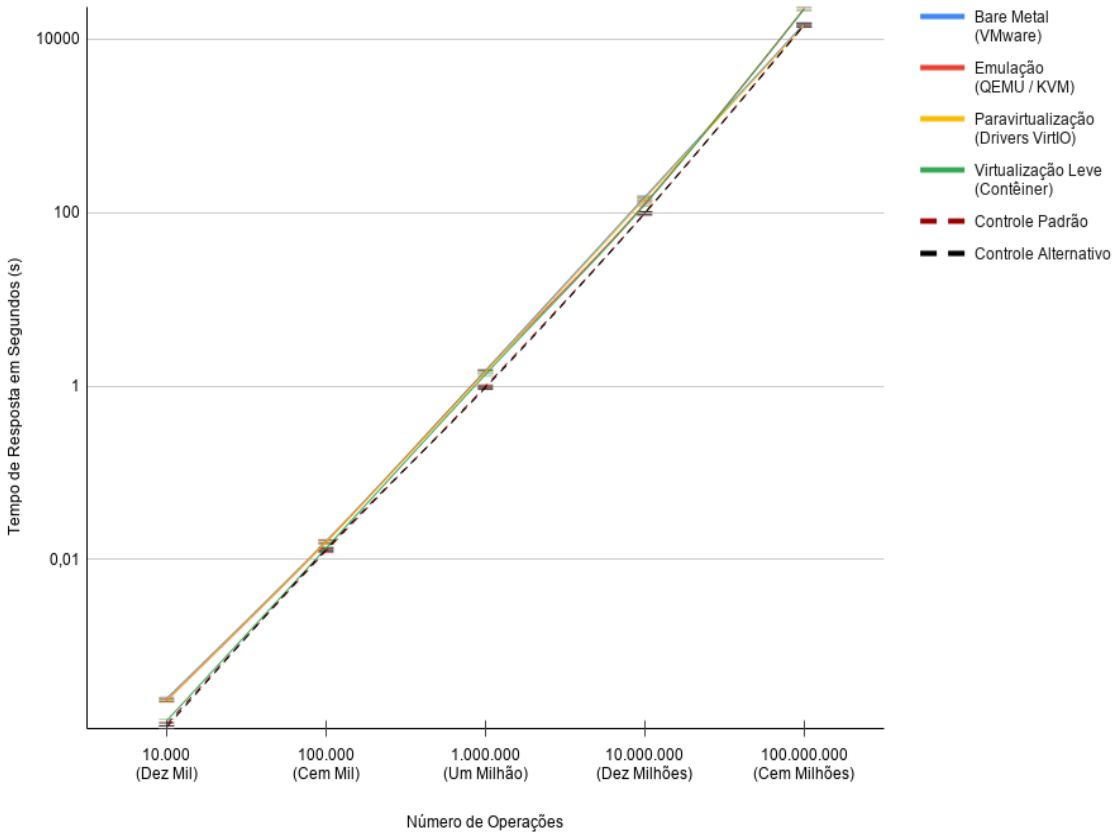

Figura 7.6: Comparativo de Tempo de Resposta - Teste de CPU
Fonte: Próprio Autor

Em geral os paradigmas de virtualização se diferem pouco dos controles. É possível observar pela figura 7.6 que em cargas de processamento que demandem pouco tempo de resposta do processador virtualizado a virtualização leve e a emulação são os paradigmas com melhor desempenho. Contudo em ambientes com cargas de processamento de maior tempo é possível verificar que paradigmas como bare metal e a paravirtualização se comportam melhor e com menos desvios de máximos e mínimos do que as anteriores citadas.

Pode-se afirmar que processamentos que demandem maior tempo da CPU sejam mais favorecidos em ambientes de virtualização completa baseada em *hardware* (bare metal) e paravirtualização do que em ambientes virtualização completa baseada em *software* (emulação) ou de virtualização leve (contêineres) (FAYYAD; PERNEEL; TIMMERMAN,

2013).

7.1.2 Teste de Processamento - CPU e RAM

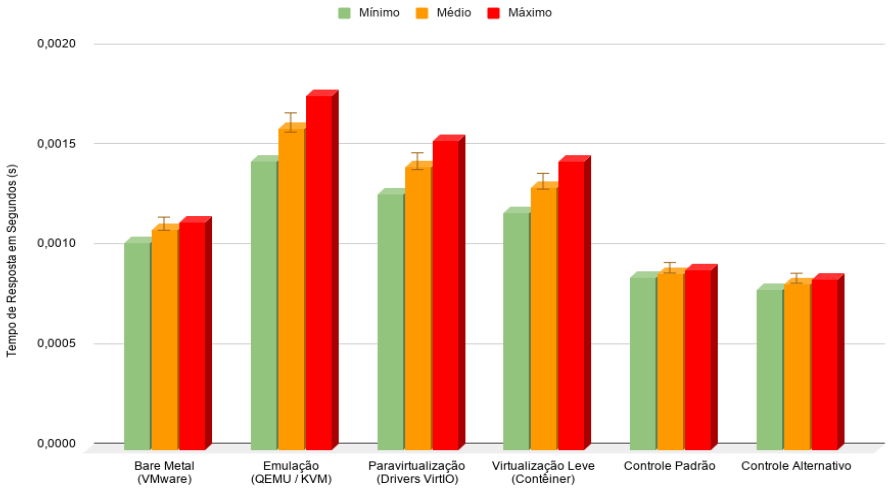

Figura 7.7: Teste CPU e RAM - Dez Mil Operações
Fonte: Próprio Autor

Observa-se que em dez mil operações, figura 7.7, a bare metal é a com o menor tempo de resposta (melhor performance) diante dos controles, com cerca de 25,09% de tempo de resposta superior ao controle. Em seguida, a virtualização leve é cotada com cerca de 49,31%. A paravirtualização se encontra com cerca de 60,69%. Por último, emulação é cotada com 82,75% de tempo de resposta superior ao controle.

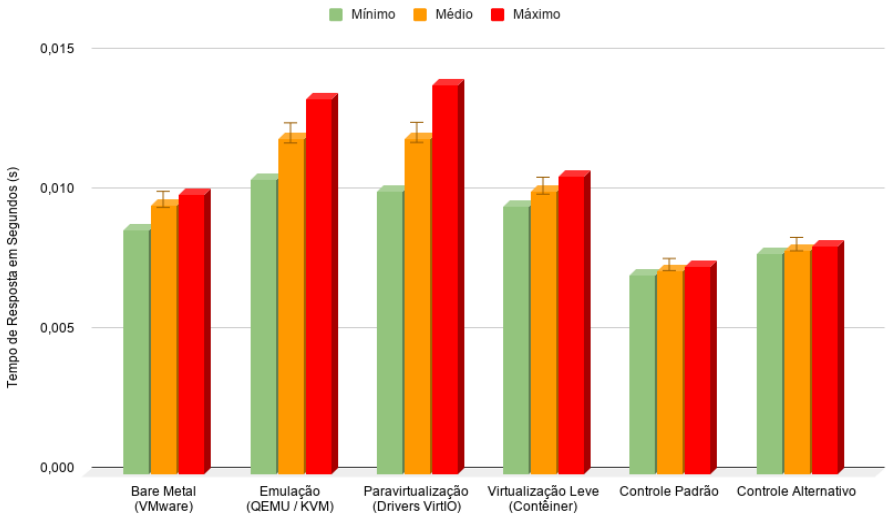

Figura 7.8: Teste CPU e RAM - Cem Mil Operações
Fonte: Próprio Autor

Observa-se que em cem mil operações, figura 7.8, a bare metal é a com o menor tempo de resposta (melhor performance) diante dos controles, com cerca de 32,03% de tempo de resposta superior ao controle. Em seguida, a virtualização leve é cotada com cerca de 38,76%. A emulação se encontra com cerca de 64,76%. Por último, paravirtualização é cotada com 65,03% de tempo de resposta superior ao controle.

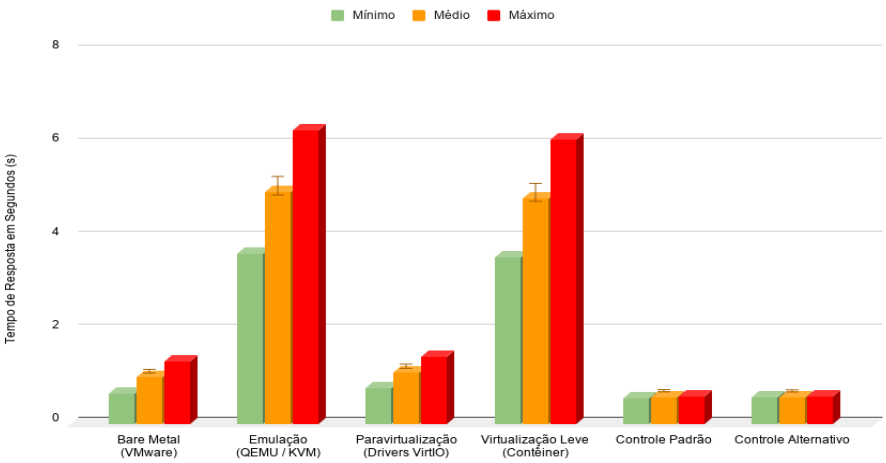

Figura 7.9: Teste CPU e RAM - Um Milhão de Operações
Fonte: Próprio Autor

Observa-se que em um milhão de operações, figura 7.9, a bare metal é a com o menor tempo de resposta (melhor performance) diante dos controles, com cerca de 72,17% de tempo de resposta superior ao controle. Em seguida, a paravirtualização é cotada com cerca de 90,66%. A virtualização leve se encontra com cerca de 736,29%. Por último, emulação é cotada com 760,84% de tempo de resposta superior ao controle. Observa-se que os desvios da bare metal e da paravirtualização são quase nulos, similares ao controle.

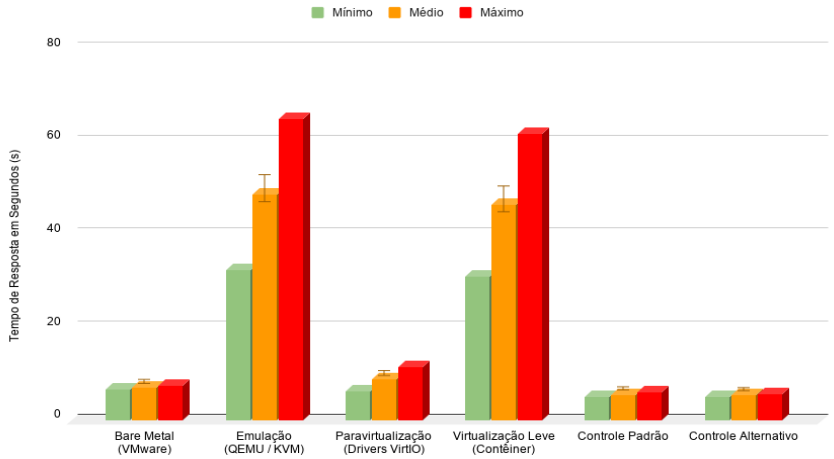

Figura 7.10: Teste CPU e RAM - Dez Milhões de Operações
Fonte: Próprio Autor

Observa-se que em dez milhões de operações, figura 7.10, a bare metal é a com o menor tempo de resposta (melhor performance) diante dos controles, com cerca de 26,51% de tempo de resposta superior ao controle. Em seguida, a paravirtualização é cotada com cerca de 59,17%. A virtualização leve se encontra com cerca de 734,23%. Por último, emulação é cotada com 775,48% de tempo de resposta superior ao controle.

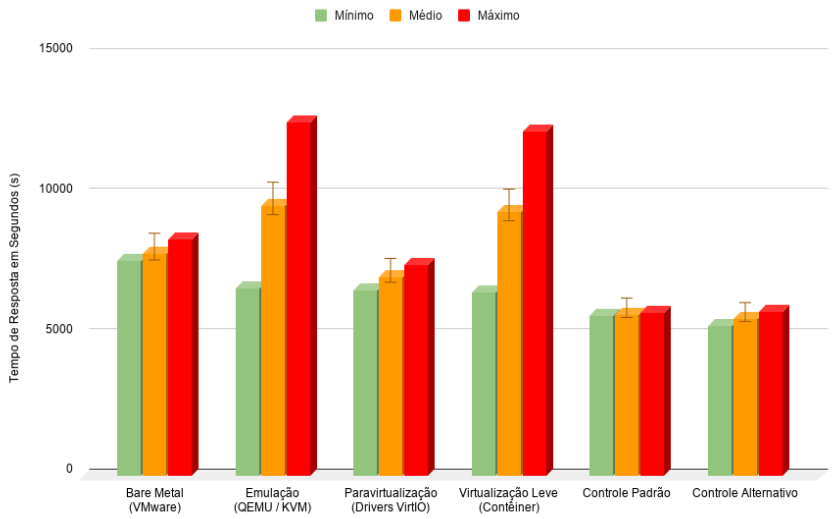

Figura 7.11: Teste CPU e RAM - Cem Milhões de Operações
Fonte: Próprio Autor

Observa-se que em dez milhões de operações, figura 7.11, a paravirtualização se torna o paradigma com o menor tempo de resposta diante dos controles, com cerca de 23,17% de tempo de resposta superior ao controle. Em seguida, a bare metal é cotada com cerca de 37,9%. A virtualização leve se encontra com cerca de 63,73%. Por último, emulação é cotada com 67,73% de tempo de resposta superior ao controle.

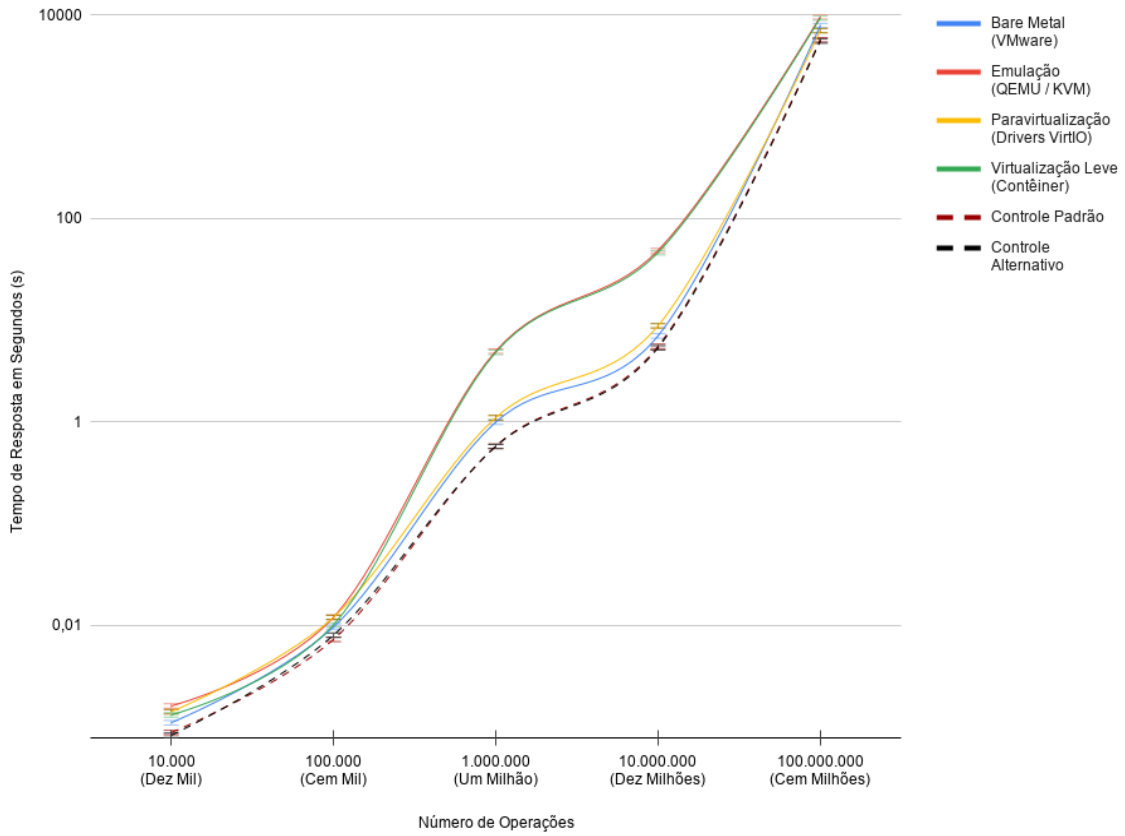

Figura 7.12: Comparativo de Tempo de Resposta - Teste de CPU e RAM
Fonte: Próprio Autor

Neste cenário os paradigmas de virtualização possuem um desvio maior ao controle. Principalmente a virtualização leve e a emulação acima de cem mil e abaixo de cem milhões de operações. É possível observar pela figura 7.12 que em cargas de processamento que demandem pouco tempo de resposta do processador e da memória RAM virtualizados a virtualização leve e a bare metal são os paradigmas com melhor performance. Contudo em ambientes com cargas de processamento de maior tempo é possível verificar que paradigmas como bare metal e a paravirtualização dominam o cenário apresentando desempenho de quatro à seis vezes o desempenho comparado à emulação e a virtualização leve em momentos de maior desvio (acima de cem mil e abaixo de cem milhões de operações), apresentando menos desvios de máximos e mínimos.

Desta forma processamentos que demandem maior tempo da CPU e maior consumo de RAM sejam mais favorecidos em ambientes de virtualização completa baseada em *hardware* (bare metal) e paravirtualização do que em ambientes virtualização completa baseada em *software* (emulação) ou de virtualização leve (contêineres).

Apesar dos resultados dos testes apontarem categoricamente para uma melhor performance em cenários de alto fluxo de processamento de paradigmas de virtualização que sejam mais próximos ao *hardware* fui necessário mais um teste que pudesse eliminar fatores de paralelismo que pudessem estar envolvidos no esquema *hardware* virtualizador. Desta forma existe a necessidade de certificar que a CPU e a memória dedicadas a má-

quina virtual não sofreram com paralelismo de outras máquinas funcionando ao mesmo tempo no mesmo *hardware*.

7.1.3 Testes de Paralelismo

Com o intuito de validar que as variações e desvios registrados são advindos exclusivamente dos paradigmas de virtualização e seus respectivos motores, foi realizado um teste amostral aumentando o número de VMs e o número de núcleos por VM para observar se existiriam quaisquer alterações além do que se já fora observado anteriormente. Devido à natureza dos testes, apenas o *benchmark* de CPU fora executado em testes que variavam da seguinte forma: 1 VM com 4 núcleos de processamento com dez rodadas de testes de dez mil, cem mil, um milhão e dez milhões e 4 VMs com 1 núcleo de processamento com dez rodadas de testes de dez mil, cem mil, um milhão e dez milhões. Em ambos os cenários os *benchmarks* foram executados em paralelo: nas máquinas de quatro cores, quatro *benchmarks* foram executados simultâneamente (um para cada núcleo) enquanto para as quatro VMs de um núcleo, cada uma executou um *benchmark*. Seguem, a seguir, os resultados:

1 VM com 4 Núcleos

Dez Mil Operações

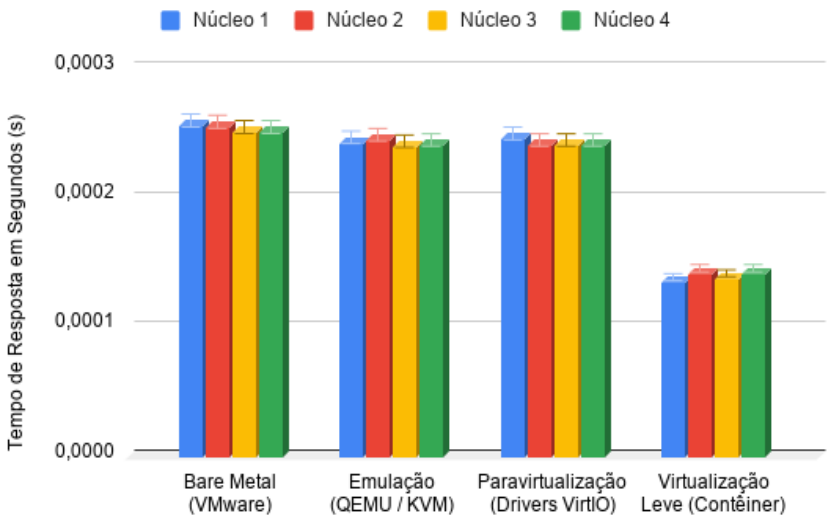

Figura 7.13: 1 VM com 4 Núcleos de Processamento - Dez Mil Operações
Fonte: Próprio Autor

De acordo com a figura 7.13, em dez mil operações os paradigmas de virtualização apresentam os seguintes desvios percentuais em relação à media: bare metal: 1,05%, emulação: 0,92%, paravirtualização: 1,05% e virtualização leve: 2,41%.

Cem Mil Operações

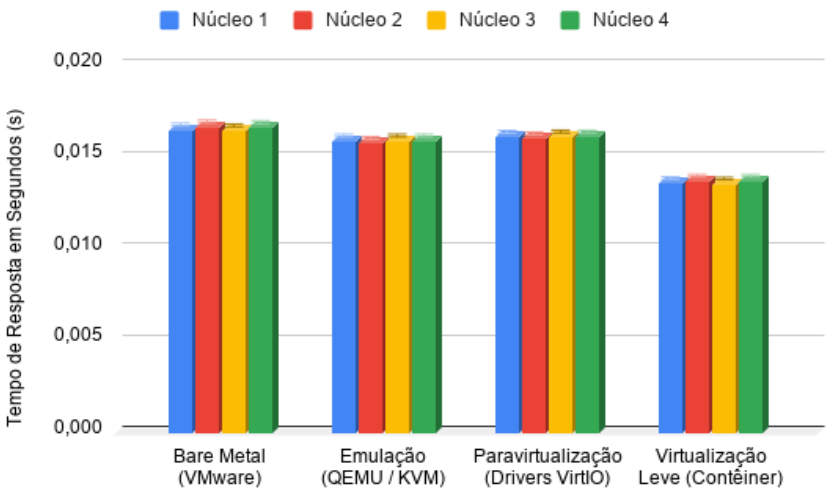

Figura 7.14: 1 VM com 4 Núcleos de Processamento - Cem Mil Operações
Fonte: Próprio Autor

De acordo com a figura 7.14, em cem mil operações os paradigmas de virtualização apresentam os seguintes desvios percentuais em relação à media: bare metal: 0,59%, emulação: 0,41%, paravirtualização: 0,3% e virtualização leve: 0,61%.

Um Milhão de Operações

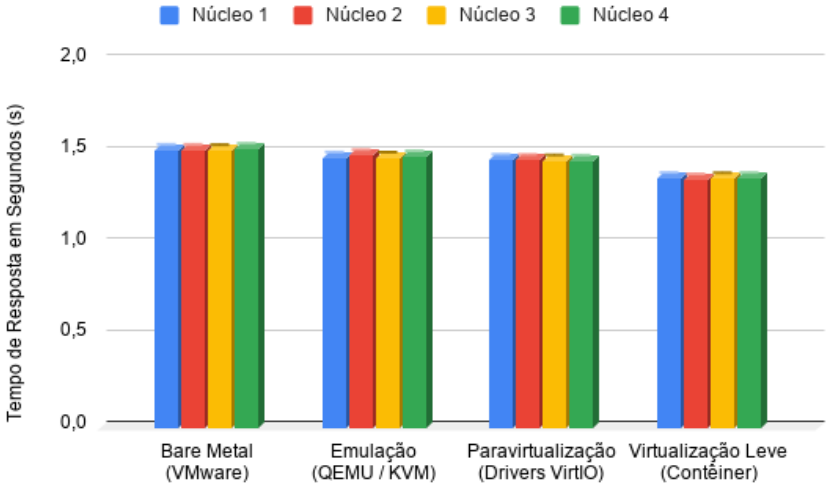

Figura 7.15: 1 VM com 4 Núcleos de Processamento Um Milhão de Operações
Fonte: Próprio Autor

De acordo com a figura 7.15, em um milhão de operações os paradigmas de virtualização apresentam os seguintes desvios percentuais em relação à media: bare metal: 0,22%, emulação: 0,41%, paravirtualização: 0,13% e virtualização leve: 0,25%.

Dez Milhões de Operações

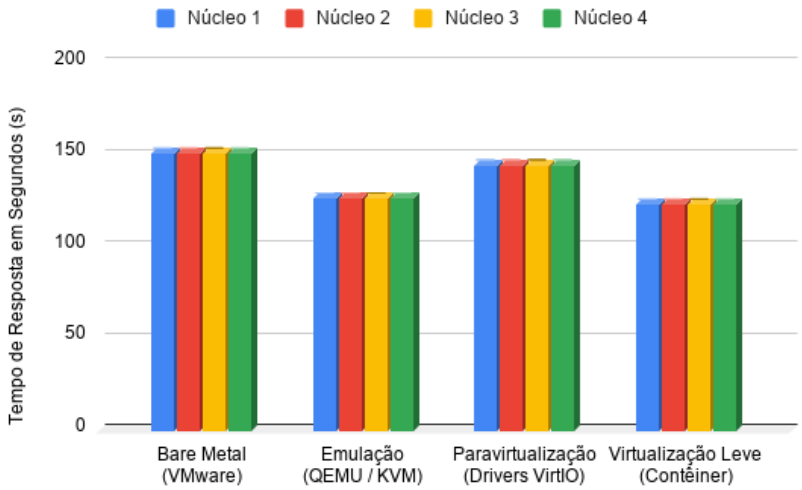

Figura 7.16: 1 VM com 4 Núcleos de Processamento - Dez Milhões de Operações
Fonte: Próprio Autor

De acordo com a figura 7.16, em dez milhões de operações os paradigmas de virtualização apresentam os seguintes desvios percentuais em relação à media: bare metal: 0,003%, emulação: 0,004%, paravirtualização: 0,001% e virtualização leve: 0,005%.

4 VMs com 1 Núcleo

Dez Mil Operações

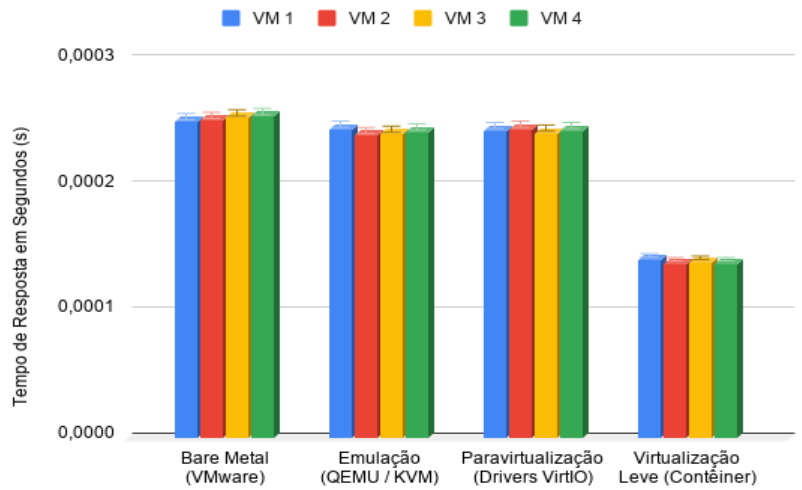

Figura 7.17: 4 VMs com 1 núcleo de Processamento - Dez Mil Operações
Fonte: Próprio Autor

De acordo com a figura 7.17, em dez mil operações os paradigmas de virtualização apresentam os seguintes desvios percentuais em relação à media: bare metal: 0,72%,

emulação: 0,91%, paravirtualização: 0,51% e virtualização leve: 1,02%.

Cem Mil Operações

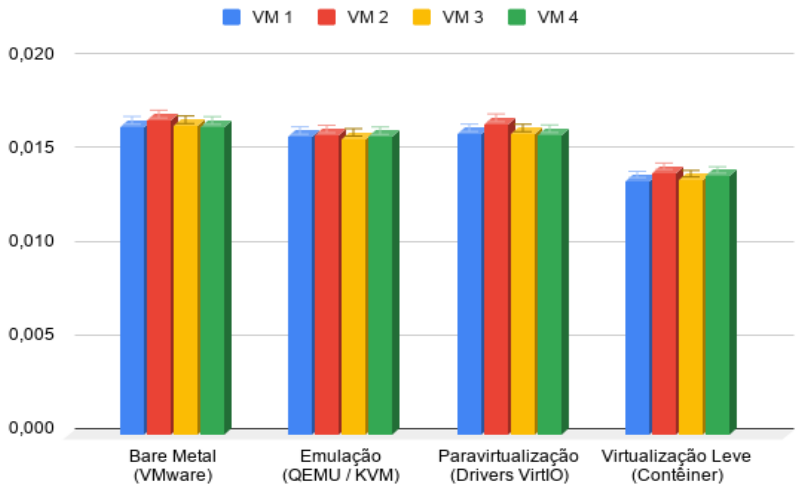

Figura 7.18: 4 VMs com 1 núcleo de Processamento - Cem Mil Operações
Fonte: Próprio Autor

De acordo com a figura 7.18, em cem mil operações os paradigmas de virtualização apresentam os seguintes desvios percentuais em relação à media: bare metal: 0,99%, emulação: 0,46%, paravirtualização: 1,72% e virtualização leve: 1,47%.

Um Milhão Operações

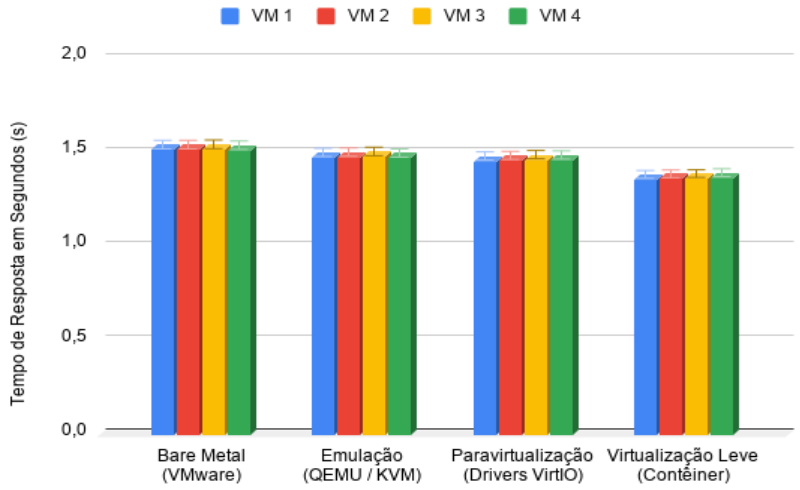

Figura 7.19: 4 VMs com 1 núcleo de Processamento - Um Milhão de Operações
Fonte: Próprio Autor

De acordo com a figura 7.19, em um milhão de operações os paradigmas de virtualização apresentam os seguintes desvios percentuais em relação à media: bare metal: 0,14%,

emulação: 0,26%, paravirtualização: 0,2% e virtualização leve: 0,27%.

Dez Milhões de Operações

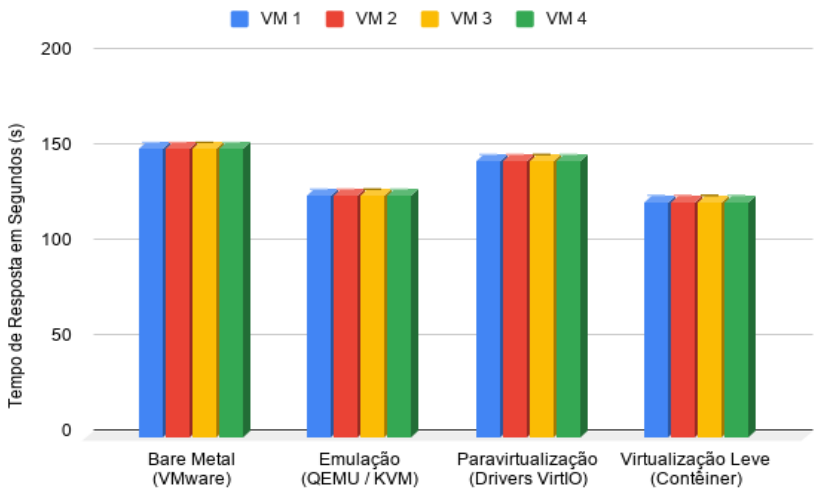

Figura 7.20: 4 VMs com 1 núcleo de Processamento - Dez Milhões de Operações
Fonte: Próprio Autor

De acordo com a figura 7.20, em dez milhões de operações os paradigmas de virtualização apresentam os seguintes desvios percentuais em relação à media: bare metal: 0,006%, emulação: 0,003%, paravirtualização: 0,001% e virtualização leve: 0,01%.

De acordo com os resultados obtidos em ambos os testes e observando o fato de que a estrutura no quesito de desvio dos testes de processamento apresentar uma baixa variação 0,001 A 2,5% é possível ratificar que tais variações observadas nos testes anteriores de desempenho são efetivamente advindas dos modelos de virtualização e seus respectivos motores responsáveis. Tendo em vista que bare metal e paravirtualização apresentaram sempre um comportamento similar de melhor desempenho em processos que demandem maior tempo de processamento, independente de paralelismos, é possível triangular tais resultados ao levantamento teórico que afirma que ambos paradigmas, por apresentar estrutura mais próxima ao *hardware*, possuem menor tempo de resposta nos processos de conversão de comandos "máquina física - *hypervisor*", o que é uma explicação plausível do fenômeno observado nos *benchmarks*. Uma possibilidade de explicação deste comportamento é que em cargas mais altas de processamento existam enfileiramentos de processos de conversão de comandos "máquina física - *hypervisor*"que são melhor tratados em virtualizadores que tenham maior contato com o *hardware*. Enquanto virtualizadores que apresentam emulação em alguma etapa do processo de virtualização sofrem perda de desempenho devido ao seu distanciamento do *hardware*.

7.2 Armazenamento

Seguindo métricas definidas por Hwang et al. (2013) e Xavier et al. (2013), a primeira rodada de testes consistiu em executar dez vezes os testes citados em capítulos anteriores em cada uma das oito máquinas virtuais utilizando blocos de 4kB nos ambientes HDD e SSD. Dois tipos de dados foram coletados: IOPS e latência (atraso). A primeira coletadas com o *software* "Fio", enquanto a segunda foi simultaneamente coletada pelo "IOping".

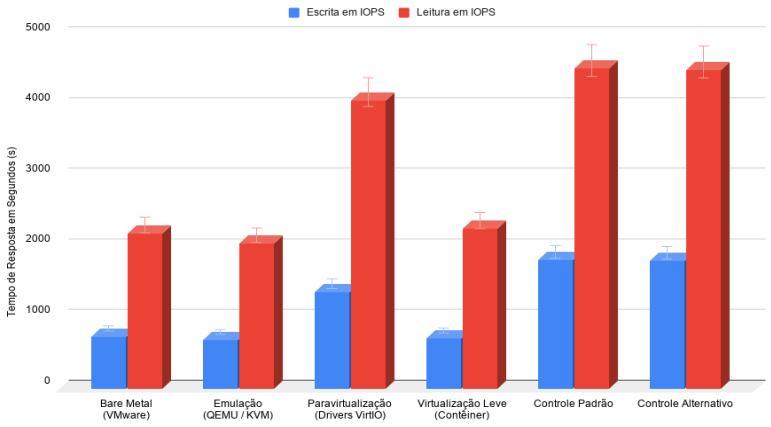

Figura 7.21: Comparativo Desempenho de Armazenamento em HDD - Blocos de 4kB
Fonte: Próprio Autor

De acordo com a figura 7.21, no comparativo desempenho de armazenamento em HDD em blocos de 4kB os paradigmas de virtualização apresentam os seguintes percentuais em relação ao controle: bare metal: 40,33% em escrita e 48,54% em leitura, emulação: 37,63% em escrita e 45,36% em leitura, paravirtualização: 75,24% em escrita e 90,12% em leitura e virtualização leve: 38,84% em escrita e 49,93% em leitura.

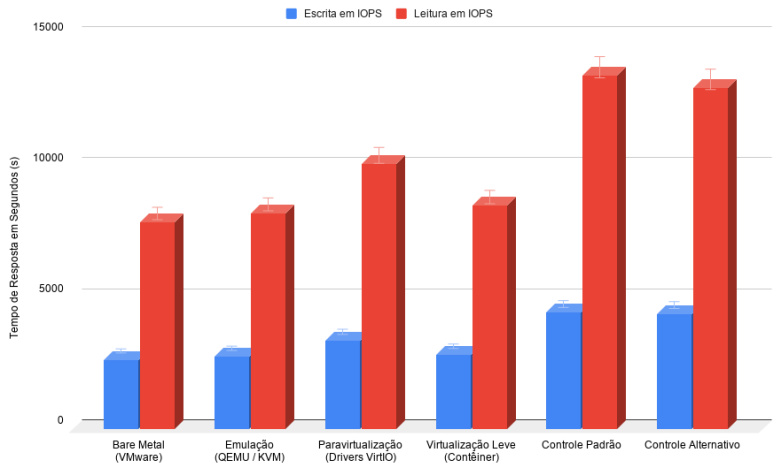

Figura 7.22: Comparativo Desempenho de Armazenamento em SSD - Blocos de 4kB
Fonte: Próprio Autor

De acordo com a figura 7.22, no comparativo desempenho de armazenamento em SSD em blocos de 4kB os paradigmas de virtualização apresentam os seguintes percentuais em relação ao controle: bare metal: 59,48% em escrita e 58,53% em leitura, emulação: 61,82% em escrita e 61,12% em leitura, paravirtualização: 76% em escrita e 75% em leitura e virtualização leve: 63,6% em escrita e 63,19% em leitura.

A segunda rodada de testes, exatamente similar à primeira, com exceção que são utilizados blocos de 4MB nos ambientes HDD e SSD. Seguem, a seguir, os resultados.

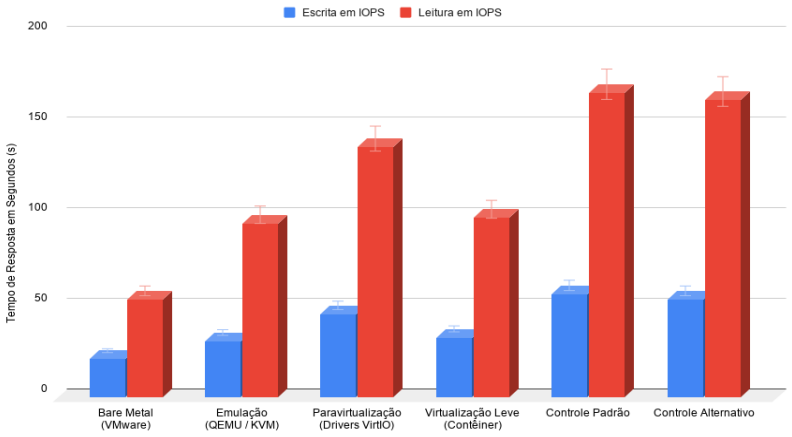

Figura 7.23: Comparativo Desempenho de Armazenamento em HDD - Blocos de 4MB
Fonte: Próprio Autor

De acordo com a figura 7.23, no comparativo desempenho de armazenamento em HDD em blocos de 4MB os paradigmas de virtualização apresentam os seguintes percentuais em relação ao controle: bare metal: 36,84% em escrita e 32,14% em leitura, emulação: 54,39% em escrita e 57,14% em leitura, paravirtualização: 80,7% em escrita e 82,14% em leitura e virtualização leve: 57,89% em escrita e 58,93% em leitura.

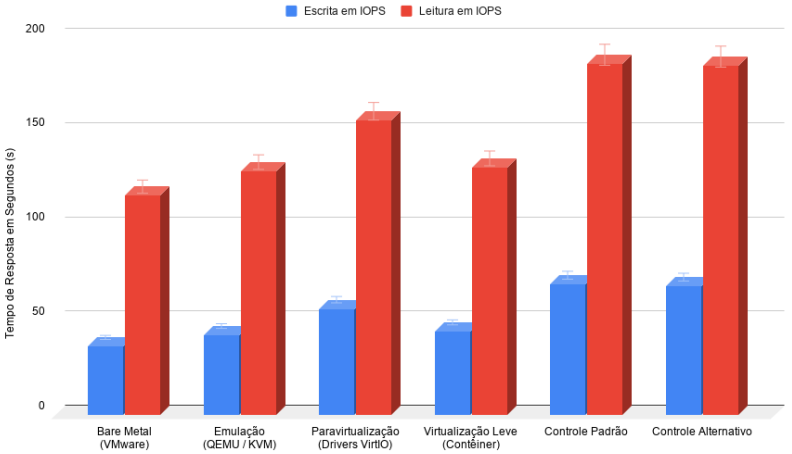

Figura 7.24: Comparativo Desempenho de Armazenamento em SSD - Blocos de 4MB
Fonte: Próprio Autor

De acordo com a figura 7.24, no comparativo desempenho de armazenamento em SSD em blocos de 4kB os paradigmas de virtualização apresentam os seguintes percentuais em relação ao controle: bare metal: 52,17% em escrita e 62,37% em leitura, emulação: 60,87% em escrita e 69,35% em leitura, paravirtualização: 81,16% em escrita e 83,87% em leitura e virtualização leve: 63,77% em escrita e 70,43% em leitura.

É possível observar que, em todos os aspectos (escrita e leitura) e ambientes (HDD e SSD) a paravirtualização é a que possui o melhor desempenho. Isso é corroborado pela literatura onde se afirma que os *drivers* para armazenamento secundário possuem latência expressivamente menor na comunicação *hardware - hypervisor*.

A tabela 7.1 explana a relação de desempenho de armazenamento em HDD em cada um dos quatro paradigmas de virtualização nos testes de 4kB e 4MB.

Tabela 7.1: Comparativo de Desempenho em HDD Entre Paradigmas

Comparação	**Blocos de 4 kB**	**Blocos de 4 MB**
Paravirtualização x Bare Metal	Paravirtualização 86% superior	Paravirtualização 119% superior
Paravirtualização x Emulação	Paravirtualização 99% superior	Paravirtualização 48% superior
Emulação x Bare Metal	Bare Metal 7% superior	Emulação 48% superior
Paravirtualização x Contêiner	Paravirtualização 93,5% superior	Paravirtualização 39% superior
Emulação x Contêiner	Contêiner 3,5% superior	Contêiner 6,4% superior
Bare Metal x Contêiner	Bare Metal 3,8% superior	Contêiner 57% superior

Fonte: Próprio Autor

A tabela 7.2 explana a relação de desempenho de armazenamento em SSD em cada um dos quatro paradigmas de virtualização nos testes de 4kB e 4MB.

Tabela 7.2: Comparativo de Desempenho em SSD Entre Paradigmas

Comparação	**Blocos de 4 kB**	**Blocos de 4 MB**
Paravirtualização x Bare Metal	Paravirtualização 27% superior	Paravirtualização 56% superior
Paravirtualização x Emulação	Paravirtualização 22% superior	Paravirtualização 27% superior
Emulação x Bare Metal	Emulação 4% superior	Emulação 17% superior
Paravirtualização x Contêiner	Paravirtualização 19,5% superior	Paravirtualização 27% superior
Emulação x Contêiner	Contêiner 2,9% superior	Contêiner 4,8% superior
Bare Metal x Contêiner	Contêiner 6,9% superior	Contêiner 22,2% superior

Fonte: Próprio Autor

Conforme figura 7.25 a terceira rodada de testes consistiu em executar o IOping em cada uma das oito VMs em HDD coletando e realizando a média simples dos seguintes dados: latência mínima, máxima e média. Seguem, a seguir, os resultados.

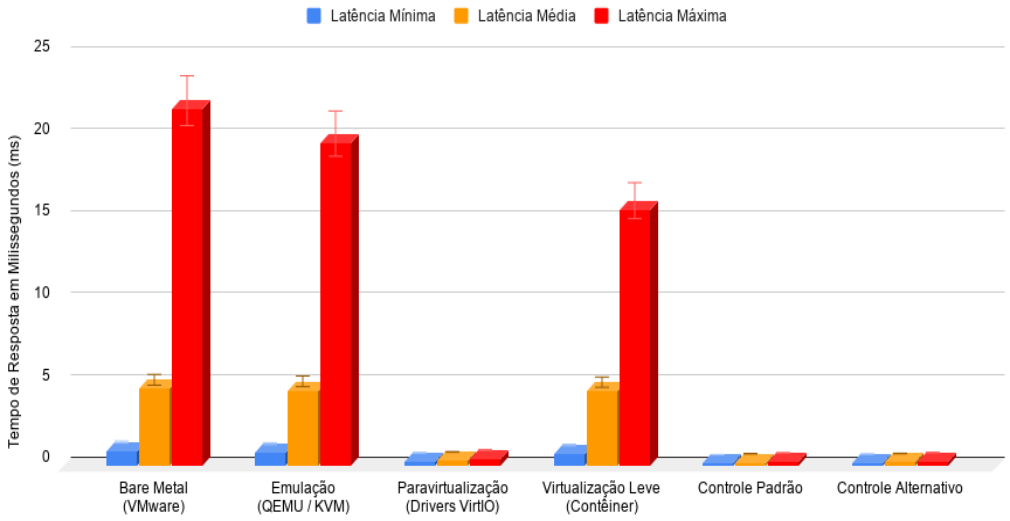

Figura 7.25: Comparativo de Latência - Armazenamento HDD
Fonte: Próprio Autor

Conforme figura 7.26 quarta rodada de testes consistiu em executar o IOping em cada uma das oito VMs em SSD coletando e realizando a média simples dos seguintes dados: latência mínima, máxima e média. Seguem, a seguir, os resultados.

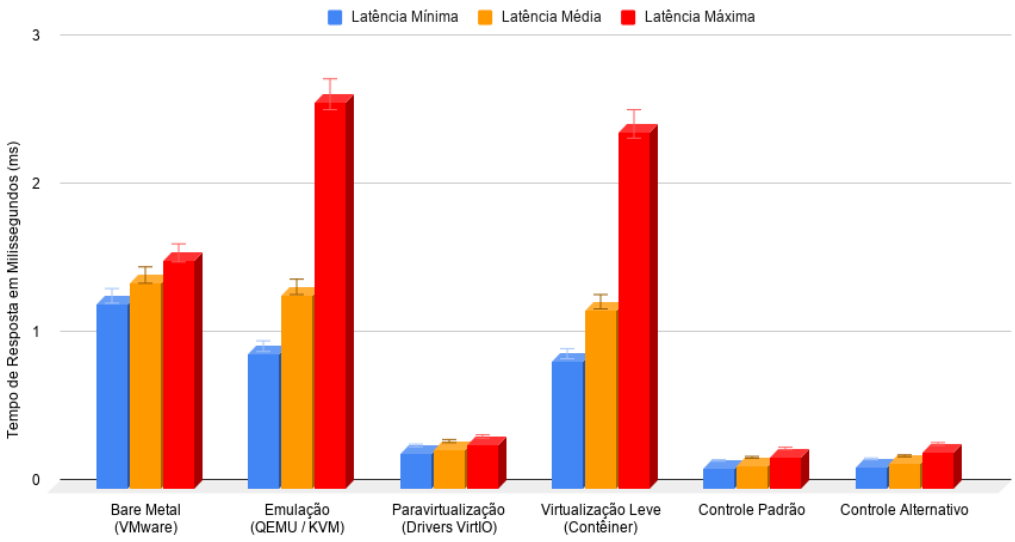

Figura 7.26: Comparativo de Latência - Armazenamento SSD
Fonte: Próprio Autor

No quesito latência o maior desempenho é observado pelo paradigma que obtiver o menor resultado. No comparativo em questão é possível observar que a latência em dispositivos de armazenamento SSD é cerca de 230% menor ao HDD nos paradigmas bare metal e emulação e apenas 16% menor ao HDD na paravirtualização. Além disso, a paravirtualização é o paradigma que é melhor performativo em qualquer cenário: no HDD a latência da paravirtualização é cerca de 400% menor que as latências medidas entre o bare metal e a emulação e no SSD a latência da paravirtualização é cerca de 1500% menor que as latências medidas entre o bare metal e a emulação. A emulação vem em seguida em questão de latência muito próxima ao bare metal (HDD e SDD) na ordem de 2% menor comparado à este último.

Capítulo 8

Análise de Tempo em Mão De Obra

Este capítulo é reservado para demonstrar o tempo investido (em minutos) para a implementação do sistema de nuvem computacional como um todo. A análise consiste em observar dois parâmetros: o tempo de implantação e o tempo de manutenção da estrutura.

O tempo de implantação corresponde ao somatório dos seguintes itens: tempo de implementação dos recursos físicos (construção da estrutura no datacenter, cabeamento, instalação de redundância elétrica, etc.), tempo de configuração (inserção dos IPs nas controladoras, comunicação com o orquestrador, etc.), tempo de implantação do virtualizador e tempo de implantação do Ceph.

O tempo de manutenção é o somatório de todas os minutos pós-implantação para alterações, manutenções e outras correções na estrutura de nuvem computacional.

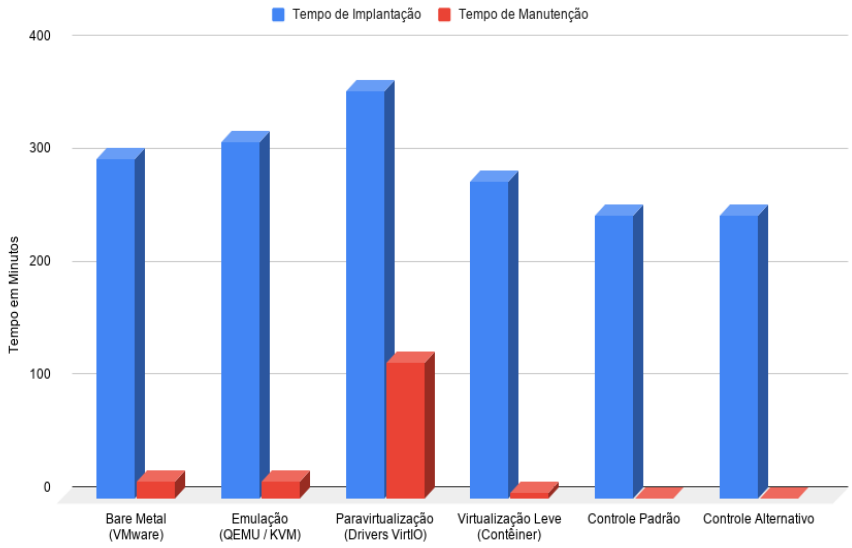

Figura 8.1: Tempo investido na configuração e manutenção das estruturas
Fonte: Próprio Autor

Conforme figura 8.1 é possível observar que a estrutura de virtualização leve utilizando o Docker foi a menos custosa à nível de tempo de trabalho de implementação e manutenção do sistema. Tal dado corrobora para o que Morabito, Kjällman e Komu (2015)

afirma onde o Docker é a ferramenta mais leve e maleável para implementações em *cloud computing*.

Logo em seguida, a estrutura bare metal é a menos custosa a nível de implementação e similar à emulação em tempo de manutenção. Isso se deve ao fato de que a VMware provê ferramentas que facilitam a implementação do sistema. É válido ressaltar que esta é uma ferramenta paga e os custos de licença devem ser considerados em eventuais cálculos financeiros.

Em terceiro lugar vem a emulação. A implementação do KVM atrelada ao QEMU torna a implantação desta estrutura um pouco mais custosa. Entretanto a nível de manutenção, esta não consumiu muitos recursos de tempo.

A estrutura mais custosa tanto à nível de implementação quanto de manutenção foi a paravirtualizada. Implementar e manter o sistema de comunicação direta de drivers para melhor desempenho de dispositivos secundários com VirtIO Drivers foi bastante custosa em relação às demais: 1,28 vezes mais custosa que o Docker para implementar e 24 vezes mais custosa que o mesmo para manutenção.

Capítulo 9

Considerações Finais

Esta dissertação apresentou a avaliação do impacto de desempenho em um sistema de computação em nuvem de alto desempenho de quatro tecnologias de virtualização mais utilizadas na atualidade: virtualização completa baseada em *hardware* (bare metal), do VMware vSphere ESXi, virtualização completa baseada em *software* (emulação), do QEMU/KVM, a paravirtualização, do KVM com os drivers VirtIO e da virtualização leve, do Docker, em que estas quatro ferramentas foram comparadas com relação ao desempenho de processamento e em dispositivos de armazenamento.

Conforme afirmado por Xavier et al. (2013) e corroborado por esta obra, isolar recursos como CPU, Memória RAM, armazenamento secundário (disco) e rede, com exceção do primeiro, é extremamente complexo. Contudo, esta obra conseguiu uma segregação mais satisfatória para os testes devido à natureza da estrutura criada e dos *benchmarks* aplicados.

Na comparação entre o VMware vSphere ESXi, QEMU/KVM, KVM com VirtIO Drivers e Docker, este último se mostrou melhor em quase todos os cenários de processamento com exceção de processos que demandassem maior tempo do processamento e da memória RAM, sendo vencido pela paravirtualização neste cenário em específico. Em análise de armazenamento a paravirtualização mostrou ser superior em desempenho em todos os cenários propostos pelo *benchmark* utilizado: IOPS e latência.

Desta forma conclui-se que o paradigma de paravirtualização no contexto do uso do KVM com os drivers VirtIO é uma tecnologia que possibilita um melhor desempenho de comunicação entre o armazenamento virtualizado e o físico para aplicações que utilizem por mais tempo a CPU e a RAM virtualizadas e/ou qualquer cenário de escrita/leitura em disco. Já o Docker é mais recomendado para processamentos que utilizem a CPU e a RAM por menor tempo.

Diante da natureza dos testes realizados e de todos os resultados obtidos, outros questionamentos foram formados e não puderam ter uma resolução clara e efetiva nesta obra, como:

- Um mesmo paradigma de virtualização em diferentes virtualizadores causaria variação nos resultados dos *benchmarks*?;
- Se todo ambiente fosse reproduzido em processadores AMD e os testes repetidos os resultados seriam os mesmos?;

- Em situações de maior estresse para o *hardware* a curva de desempenho de CPU, memória RAM e disco das máquinas virtuais seria a mesma?;
- Como o desempenho de aplicações como serviços Web ou bancos de dados se comportarão em cada um dos ambientes de virtualização?;
- Os mesmos resultados observados em estruturas de *cloud computing* seriam observados em estruturas de computação na borda (*edge computing*) e computação em névoa (*fog computing*)?;
- A adição de um *benchmark* de rede e demais periféricos influenciaria nos resultados?;
- Testes realizados em outros sistemas operacionais voltados para a nuvem computacional como Windows Server apresentariam os mesmos resultados que o Linux CentOS?;

Tais questionamentos são extremamente relevantes para a análise de desempenho de elementos virtualizados em ambientes de *cloud computing*. Tal importância elege estes questionamentos para servirem de base para trabalhos futuros de análises mais aprofundadas.

Referências Bibliográficas

Aghayev, A. et al. File systems unfit as distributed storage backends:lessons from 10 years of ceph evolution. In: . [s.n.], 2019. p. 17. Disponível em: <https://dl.acm.org/doi/pdf/10.1145/3341301.3359656>. Acesso em: 14 abr 2020. 52

ALTAHER, A. W. *Virtualization Hypervisor Servers: Basics, Concepts, Methods*. [S.l.]: LAP LAMBERT Academic Publishing, 2017. 18, 19, 20, 23

Armstrong, D.; Djemame, K. Performance issues in clouds: An evaluation of virtual image propagation and i/o paravirtualization. *The Computer Journal*, v. 54, n. 6, p. 836–849, 2011. 25, 26

Babu, S. A. et al. System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xenserver. In: *2014 Fourth International Conference on Advances in Computing and Communications*. [S.l.: s.n.], 2014. p. 247–250. 18, 22

CHE, J. et al. A synthetical performance evaluation of openvz, xen and kvm. In: . USA: IEEE Computer Society, 2010. ISBN 9780769543055. Disponível em: <https://doi.org/10.1109/APSCC.2010.83>. 36

CHIERICI, A.; VERALDI, R. A quantitative comparison between xen and kvm. *Journal of Physics: Conference Series*, v. 219, p. 042005, 05 2010. 19

DELL BRASIL. Poweredge r640. 2020. Disponível em: <https://www.dell.com/pt-br/work/shop/enterprise-products/poweredge-r640/spd/poweredge-r640#techspecs_section>. 50

DELL BRASIL. Poweredge r740. 2020. Disponível em: <https://www.dell.com/pt-br/work/shop/enterprise-products/poweredge-r740/spd/poweredge-r740#techspecs_section>. 49

DEUTSCH, L. P.; SCHIFFMAN, A. M. Efficient implementation of the smalltalk-80 system. In: *Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. [S.l.: s.n.], 1984. p. 297–302. 21

Dong, Y. et al. Hyvi: A hybrid virtualization solution balancing performance and manageability. *IEEE Transactions on Parallel and Distributed Systems*, v. 25, n. 9, p. 2332–2341, 2014. 19, 25

Elsayed, A.; Abdelbaki, N. Performance evaluation and comparison of the top market virtualization hypervisors. In: *2013 8th International Conference on Computer Engineering Systems (ICCES)*. [S.l.: s.n.], 2013. p. 45–50. 41

ERL, T.; MAHMOOD, Z.; PUTTINI, R. *Cloud Computing: Concepts, Technology Architecture*. 1. ed. USA: Prentice Hall, 2013. <Disponívelem:https://www.amazon.com/Cloud-Computing-Concepts-Technology-Architecture/dp/0133387526>. 15, 25, 26, 27, 28

FAYYAD, H.; PERNEEL, L.; TIMMERMAN, M. Full and para-virtualization with xen: A performance comparison. *Journal of Emerging Trends in Computing and Information Sciences*, Volume 10, p. 719–727, 09 2013. 19, 23, 76

FORBES BRASIL. A aquisição feita em 2016 que virou trunfo da amazon. Forbes, 2019. Disponível em: <https://forbes.com.br/negocios/2019/03/a-aquisicao-feita-em-2016-que-virou-trunfo-da-amazon/>. 31, 32

GONZALEZ, B. B.; JUIZ, C. Virtual machine consolidation: a systematic review of its overhead influencing factors. *The Journal of Supercomputing*, v. 76, 10 2019. 45

GORDON, A. et al. *ELI: Bare-Metal Performance for I/O Virtualization*. [S.l.]: ACM Architectural Support for Programming Languages Operating Systems (ASPLOS), 2012. <Disponívelem:https://www.researchgate.net/publication/232606925_ELI_Bare-Metal_Performance_for_IO_Virtualization>. Acesso em: 18 nov 2019. 30

GUSEV, M. et al. Windows azure: Resource organization performance analysis. In: VILLARI, M.; ZIMMERMANN, W.; LAU, K.-K. (Ed.). *Service-Oriented and Cloud Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. p. 17–31. 31

HUNTER, T.; PORTER, S. *Google Cloud Platform for Developers: Build highly scalable cloud solutions with the power of Google Cloud Platform*. Packt Publishing, 2018. ISBN 9781788830836. Disponível em: <https://books.google.com.br/books?id=dO1mDwAAQBAJ>. 31

Hwang, J. et al. A component-based performance comparison of four hypervisors. In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. [S.l.: s.n.], 2013. p. 269–276. 39, 63, 71, 85

JOURDAN, M. et al. *Conf. on Programming Languages Design and Implementation*. [S.l.]: White Plains, NY, 1990. 22

KAVIS, M. J. *Architecting The Cloud - Design Decisions For Cloud Computing Service Models Saas Paas And Iaas*. [S.l.]: Wiley, 2014. 28, 29

Kotas, C.; Naughton, T.; Imam, N. A comparison of amazon web services and microsoft azure cloud platforms for high performance computing. In: *2018 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.: s.n.], 2018. p. 1–4. 31

Li, J. et al. Performance overhead among three hypervisors: An experimental study using hadoop benchmarks. In: *2013 IEEE International Congress on Big Data*. [S.l.: s.n.], 2013. p. 9–16. 40

LI, Z. et al. *Performance Overhead Comparison between Hypervisor and Container based Virtualization*. 2017. <Disponívelem:https://arxiv.org/pdf/1708.01388.pdf>. Acesso em: 04 abr 2020. 44, 55, 56, 57, 60, 63

LINGESWARAN, R. *Para virtualization vs Full virtualization vs Hardware assisted Virtualization*. 2017. <Disponívelem:https://www.unixarena.com/2017/12/para-virtualization-full-virtualization-hardware-assisted-virtualization.html>. Acesso em: 7 set 2019. 22, 23

MATTHEWS, J. N. et al. Quantifying the performance isolation properties of virtualization systems. In: . New York, NY, USA: Association for Computing Machinery, 2007. ISBN 9781595937513. ExpCS '07: Proceedings of the 2007 Workshop on Experimental Computer Science. Disponível em: <https://doi.org/10.1145/1281700.1281706>. 35

MCDOUGALL, R.; ANDERSON, J. Virtualization performance: Perspectives and challenges ahead. *SIGOPS Oper. Syst. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 44, n. 4, dez. 2010. ISSN 0163-5980. Disponível em: <https://doi.org/10.1145/1899928.1899933>. 30, 36

MITCHELL, N. J.; ZUNNURHAIN, K. Google cloud platform security. In: . New York, NY, USA: Association for Computing Machinery, 2019. ISBN 9781450367332. Disponível em: <https://doi.org/10.1145/3318216.3363371>. 32

MORABITO, R.; KJALLMAN, J.; KOMU, M. *Hypervisors vs. Lightweight Virtualization: a Performance Comparison*. [S.l.]: IEEE International Conference on Cloud Engineering, 2015. <Disponívelem:https://www.researchgate.net/publication/273756984_Hypervisors_vs_Lightweight_Virtualization_A_Performance_Comparison>. Acesso em: 20 dez 2019. 18

Morabito, R.; Kjällman, J.; Komu, M. Hypervisors vs. lightweight virtualization: A performance comparison. In: *2015 IEEE International Conference on Cloud Engineering*. [S.l.: s.n.], 2015. p. 386–393. 43, 90

Narula, S.; Jain, A.; Prachi. Cloud computing security: Amazon web service. In: *2015 Fifth International Conference on Advanced Computing Communication Technologies*. [S.l.: s.n.], 2015. p. 501–505. 31

Open Nebula. Open nebula documentation. 2020. Disponível em: <http://docs.opennebula.io/5.10/>. Acesso em: 05 out 2020. 51, 52

PINTO, V. G.; SCHNORR, L. M.; LEGRAND, A. *Análise de Aplicação baseada em Tarefas em Arquitetura Híbrida CPU/GPU*. 2017. <Disponívelem:https://sol.sbc.org.br/index.php/eradrs/article/view/4747/4664>. Acesso em: 05 abr 2020. 56

PORTNOY, M. *Virtualization Essentials*. 2. ed. [S.l.]: Sybex, 2016. <Disponívelem:https://www.amazon.com/Virtualization-Essentials-2nd-Matthew-Portnoy/dp/1119267722>. 17, 18

POTHERI, M. *Virtualized High Performance Computing (HPC) Reference Architecture (Part 1 of 2)*. [S.l.: s.n.], 2018. <Disponívelem:https://blogs.vmware.com/apps/2018/09/vhpc-ra-part1.html>. Acesso em: 16 dez 2019. 18

Raho, M. et al. Kvm, xen and docker: A performance analysis for arm based nfv and cloud computing. In: *2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. [S.l.: s.n.], 2015. p. 1–8. 24

S., A. B. et al. System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xenserver. In: *2014 Fourth International Conference on Advances in Computing and Communications*. [S.l.: s.n.], 2014. p. 247–250. 43

SMITH, J.; NAIR, R. *Virtual machines: versatile platforms for systems and processes*. [S.l.]: Elsevier, 2005. 20, 21

SOLTESZ, S. et al. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. Association for Computing Machinery, New York, NY, USA, v. 41, n. 3, 2007. ISSN 0163-5980. Disponível em: <https://doi.org/10.1145/1272998.1273025>. 34

TANENBAUM, A. S. *Structured computer organization*. [S.l.]: Pearson Education India, 2016. 20

UFPE, U. F. de P. Conceito: Teste de desempenho. In: . [s.n.], 2006. p. 1. Disponível em: <https://www.cin.ufpe.br/~gta/rup-vc/core.base_rup/guidances/concepts/performance_testing_37A31809.html>. Acesso em: 05 out 2020. 55

Walters, J. P. et al. A comparison of virtualization technologies for hpc. In: *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*. [S.l.: s.n.], 2008. p. 861–868. 22, 23

WATTS, S.; RAZA, M. Saas vs paas vs iaas: Whats the difference and how to choose. BMC, 2019. Disponível em: <https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/>. 29

Xavier, M. G. et al. Performance evaluation of container-based virtualization for high performance computing environments. In: *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. [S.l.: s.n.], 2013. p. 233–240. 30, 38, 63, 85, 92

YARALI, A. *Cloud, Fog, and Edge: Technologies and Trends in Telecommunications Industry*. [S.l.]: Nova Science Pub Inc, 2018. <Disponívelem:https://www.amazon.com/Cloud-Fog-Edge-Technologies-Telecommunications/dp/1536144436/ref=sr_1_4>. 15, 25, 26, 27, 28, 31

Zhang, X.; Dong, Y. Optimizing xen vmm based on intel® virtualization technology. In: *2008 International Conference on Internet Computing in Science and Engineering*. [S.l.: s.n.], 2008. p. 367–374. 19, 23

Apêndice A

Códigos Para Testes de Processamento

```
import time #importa a função de tempo do sistema operacional

c = int(input("Número de Operações: ")) #coleta a quantidade de operações a realizar
a = time.time() #armazena a hora específica do inicio da Execução da Função
var=2**c #calcula a potenciação de 2 pelo expoente escolhido.
print(var) #mostra em tela o número gerado pela execução da função
b = time.time() #armazena a hora específica do final da execução da Função
print(b - a) #mostra o tempo gasto para Executar a função

#Criado por Giovane de Morais no dia 11 de abril de 2020.
```

Figura A.1: Código Para Stress Exclusiva de CPU
Fonte: Próprio Autor

```
import random #importa a função de geração de número aleatórios
import time #importa a função de tempo do sistema operacional

c = int(input("Número de Operações: ")) #coleta a quantidade de operações a realizar
saida = [] #criado o vetor para armazenamento dos dados randomicos
for _ in range(c):
    a = time.time() #armazena a hora específica do inicio da Execução da Função
    saida.append(random.randint(0, 1)) #popula o vetor com dados randomicos no range 0 à 100
print(saida) #imprime na tela o vetor
b = time.time() #armazena a hora específica do final da execução da Função
print(b - a) #mostra o tempo gasto para Executar a função

#Criado por Giovane de Morais no dia 11 de abril de 2020.
```

Figura A.2: Código Para Stress de CPU e memória RAM
Fonte: Próprio Autor