ANÁLISE DO DESEMPENHO DA VIRTUALIZAÇÃO LEVE PARA AMBIENTES COM EDGE COMPUTING BASEADA EM NFV

Leon Valentim Porto Trindade

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Luís Henrique Maciel Kosmalski Costa

Rio de Janeiro
Fevereiro de 2018

ANÁLISE DO DESEMPENHO DA VIRTUALIZAÇÃO LEVE PARA AMBIENTES COM EDGE COMPUTING BASEADA EM NFV

Leon Valentim Porto Trindade

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Luís Henrique Maciel Kosmalski Costa, Dr.

Prof. Anelise Munaretto Fonseca, Dr.

Prof. Marcelo Gonçalves Rubinstein, D.Sc.

Prof. Miguel Elias Mitre Campista, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2018

Trindade, Leon Valentim Porto

Análise do Desempenho da Virtualização Leve para Ambientes com Edge Computing baseada em NFV/Leon Valentim Porto Trindade. – Rio de Janeiro: UFRJ/COPPE, 2018.

XV, 57 p.: il.; 29, 7cm.

Orientador: Luís Henrique Maciel Kosmalski Costa

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2018.

Referências Bibliográficas: p. 52 – 57.

1. Desempenho. 2. Contêiner. 3. Edge Computing. 4. Virtualização leve. I. Costa, Luís Henrique Maciel Kosmalski. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

À minha família. Em especial à minha mãe Mariolan e à minha avó Aida (In memoriam).

Agradecimentos

Agradeço a minha família, em especial a minha mãe Mariolan por todo carinho, apoio e motivação sempre.

Também gostaria de agradecer a Mayli por sempre estar ao meu lado e a sua família por terem me dado todo o apoio necessário para a superar as dificuldades que surgiram ao longo desse período de estudo.

Além disso, agradeço aos amigos da Embratel em especial ao José Silva, Antônio Silvério, Walderson Vidal e Carlos Cunha que me deram essa oportunidade para autodesenvolvimento, também aos RT's do Centro de Referência Tecnológica da Embratel especialmente a Carolina Neves, Bruno Peres, Joaquim Carvalho, Luiza Herback, Maurício Silva, Zeneide Veras, Rodrigo Vieira, Marcelo Gomes, Laila Sousa, Mauro Gomes, Aldair Freire, Fábio Biggi, Josemar e Genílson Santos que me ajudaram e me motivaram ao longo dessa jornada.

Gostaria agradecer ao meu orientador Luís Henrique pela paciência, apoio e todo conhecimento compartilhado a fim de me direcionar ao longo do mestrado ajudando a elaborar e organizar as ideias e textos com dicas sempre valiosas que foram essenciais para meu o aprendizado e crescimento profissional.

Agradeço também aos professores Miguel Mitre, Anelise Munaretto e Marcelo Rubinstein pela participação na banca examinadora desta dissertação.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ANÁLISE DO DESEMPENHO DA VIRTUALIZAÇÃO LEVE PARA AMBIENTES COM EDGE COMPUTING BASEADA EM NFV

Leon Valentim Porto Trindade

Fevereiro/2018

Orientador: Luís Henrique Maciel Kosmalski Costa

Programa: Engenharia Elétrica

O paradigma de *Network Function Virtualization* (NFV) está mudando a forma pela qual os serviços de rede são providos, permitindo o mesmo grau de versatilidade e agilidade já visto na computação na nuvem. De fato, o NFV propõe mudar as funções de rede que hoje são executadas nas chamadas *middleboxes*, um hardware dedicado, em funções virtuais. Ou seja, transformar soluções fechadas em simples imagens de software, chamadas *Virtual Network Function*s (VNFs), as quais podem ser consolidadas e executadas em servidores padrões da indústria. Porém, é possível implementar as funções virtuais através da virtualização tradicional baseada em hipervisores ou da virtualização leve, constituindo um compromisso entre flexibilidade e desempenho. Assim, este trabalho compara o desempenho em ambientes que utilizam tecnologia de virtualização com hipervisor aos que utilizam contêineres, a fim de verificar a mais adequada para ambientes operacionais aliados ao conceito de *Edge Computing* baseada em NFV. Com isso, pretende-se aferir a capacidade dessas tecnologias serem transparentes ao usuário em relação ao desempenho. Para tanto, foram feitos benchmarkings de *Central Processing Unit* (CPU), memória, *Input/Output* (I/O) de disco, rede e aplicação, evidenciando a baixa sobrecarga da virtualização leve que chega a se aproximar do cenário nativo em certos casos.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

LIGHTWEIGHT VIRTUALIZATION PERFORMANCE ANALYSIS FOR NFV-BASED EDGE COMPUTING ENVIRONMENTS

Leon Valentim Porto Trindade

February/2018

Advisor: Luís Henrique Maciel Kosmalski Costa

Department: Electrical Engineering

The *Network Function Virtualization* (NFV) paradigm is changing the way network services are provided, allowing the same degree of versatility and agility as ever seen in cloud computing. In fact, NFV proposes to change the network functions that are currently performed in the so-called middleboxes, a dedicated hardware, in virtual functions. That is, turn closed solutions into simple software images, called *Virtual Network Function* (VNF), which can be consolidated and run on industry standard servers. However, it is possible to implement as virtual functions through traditional virtualization in hypervisors or in lightweight virtualization, constituting a compromise between flexibility and performance. Thus, this work compares the performance in environments that use virtualization technology with hypervisor to those that uses a container, for the purpose of verify the best for operational environments allied to the concept of Edge Computing in NFV. However, it is intended to gauge the ability for the technologies to be transparent to the user regarding performance. In order to do so, comparisons of CPU, memory, disk I/O, network and application were made, evidencing a low overhead of lightweight virtualization that comes close to the native scenario in certain cases.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AHCI *Advanced Host Controller Interface*

ALU *Arithmetic and Logic Unit*

API *Application Programming Interface*

AUFS *AnotherUnionFS*

CAPEX *Capital Expenditure*

CDN *Content Delivery Networks*

CFS *Completely Fair Scheduler*

cgroups *Control Groups*

COTS *Commercial off-the-shelf*

CoW *Copy-on-Write*

CPE *Customer Premises Equipment*

CPL *Current Privilege Level*

CPU *Central Processing Unit*

CRM *Customer Relationship Management*

CSAM *Code Scanning and Analysis Manager*

CR3 *Control Register 3*

CU *Control Unit*

DocLite *Docker Container-bases technology Lightweight Benchmarking*

DPDK *Data Plane Development Kit*

EPC *Evolved Packet Core*

EPT *Extended Page Table*

ERP *Enterprise Resource Planning*

ETSI *European Telecommunication Standards Institute*

HD *Hard Disk*

HTTP *Hypertext Transfer Protocol*

IMS *IP Multimedia Subsystem*

IOPS *Input Output per Second*

IoT *Internet of Things*

IP *Internet Protocol*

IPC *Interprocess Communication*

ISP *Internet Service Provider*

ISV *Independent Software Vendor*

I/O *Input/Output*

KVM *Kernel-based Virtual Machine*

LXC *Linux Containers*

LRU *Least Recently Used*

MEC *Mobile Edge Computing*

NAT *Network Address Translation*

NF *Network Function*

NFV *Network Function Virtualization*

NFVI *Network Function Virtulization Infrastructure*

NIC *Network Interface Card*

NSC *Network Service Chaining*

OCI *Open Container Initiative*

OPEX *Operational Expenditure*

OTT *Over the Top*

PATM *Patch Manager*

PCI *Peripheral Component Interconnect*

PID *Process Identifier*

POSIX *Portable Operating System Interface*

QCI *QoS Class Identifier*

QEMU *Quick Emulator*

QoE *Quality of Experience*

QoS *Quality of Service*

RAM *Random Access Memory*

RAN *Radio Access Network*

REST *Representational State Transfer*

RTT *Round Trip Time*

SATA *Serial ATA*

SBC *Single Board Computer*

SD-WAN *Software Defined-Wide Area Network*

SLA *Service Level Agreement*

SR-IOv *Single Root I/O Virtualization*

SSD *Solid-State Drive*

TCP *Transmission Control Protocol*

TI Tecnologia da Informação

TLB *Translation Look-aside Buffer*

UDP *User Datagram Protocol*

UTS *UNIX Time Sharing*

vCPE *Virtual Customer Premises Equipment*

VDI *Virtual Disk Image*

VM *Virtual Machine*

VMM *Virtual Machine Monitor*

VMCS *Virtual Machine Control Structure*

VNF *Virtual Network Function*

VPS *Virtual Private Server*

Capítulo 1

Introdução

Os provedores de serviços de telecomunicações deparam-se constantemente com desafios de mercado e assim buscam reduzir custos, sejam em termos de *Capital Expenditure* (CAPEX) e de *Operational Expenditure* (OPEX). Além disso, buscam o aumento da agilidade de configuração dos serviços e otimização da utilização de recursos. O NFV promete resolver esses desafios porque permite a diminuição do *Time-to-Market* de serviços de rede nas operadoras, uma vez que o lançamento de serviços pode ser feito com menores investimentos.

Além disso, o NFV permite uma maior flexibilidade de configuração, migração, automação e gerência. Com isso, desde processos técnicos até de governança podem ser evoluídos, agregando mais dinamismo e acelerando ciclos de inovação nas empresas de tecnologia. O NFV promove uma mudança disruptiva na organização das empresas, uma vez que demanda planejamento tanto para reorganizar infraestrutura técnica e configuração lógica de rede quanto funções e departamentos pessoais que em alguns casos deixarão de existir.

Atualmente, a infraestrutura de redes das operadoras possui algumas características bem definidas: nós topologicamente fixos, algumas funções de rede com redundância dentro de *black boxes*, demora no lançamento de novos serviços pela operadora, novos serviços significam instalação de novos equipamentos e lógica de controle em cada dispositivo com difícil evolução. Sendo assim, características que contrastam com a flexibilidade, portabilidade e resiliência propostas pela virtualização.

A premissa do NFV é evoluir a execução de funções de rede em servidores dedicados para *Virtual Network Function*s (VNFs), executadas em servidores baseados em arquitetura x86, de uso comercial. O conceito de execução de funções de rede em software e em plataformas de computação genéricas não é novo, uma vez que funções de rede baseadas em software estão disponíveis há anos, como o *Quagga Open Source Router* [1] e o *strongSwan IPsec Gateway* [2] que são executados em computadores com arquitetura x86.

Quando um servidor é compartilhado por diversos usuários, a sua utilização aumenta e o desempenho desejado por cada usuário deverá ser o mesmo obtido com hardware dedicado. Uma das maiores vantagens das tecnologias de virtualização nos *datacenters* é o melhor uso do hardware disponível podendo inclusive compartilhá-lo em arquiteturas multi-inquilino, torna-se também uma das grandes desvantagens da virtualização.

Com usuários simultâneos, o desempenho pode ser comprometido quando o número de utilizadores ou o número de tarefas demandadas torna-se elevado. Isso irá afetar o sistema como um todo e resultará em um efeito cascata podendo resultar em uma resposta lenta e um desempenho ruim para todos os utilizadores. Para evitar esse cenário as operadoras necessitam de uma excelente planejamento desses sistemas para o seu uso sob alta demanda e assim poder obter um desempenho próximo aos de servidores dedicados.

Embora a virtualização possua diversas vantagens e seja uma tecnologia proeminente da atualidade, ela também possui algumas desvantagens. No caso de desempenho, o desempenho do sistema virtualizado geralmente é menor que o do sistema ou rede sem virtualização. Apesar das máquinas virtuais serem isoladas do hardware físico, elas ainda estão conectadas a esse hardware no sentido que uma falha nesse hardware também reflete em uma falha na *Virtual Machine* (VM).

Quando a virtualização é investigada de perto, são observadas vantagens e desvantagens. Então, se faz necessário o conhecimento e análise do impacto da virtualização através da medição de desempenho em cenários com diversos tipos de virtualização para compará-los a um referencial sem virtualização e assim verificar o nível de sobrecarga introduzido pelas diferentes técnicas de virtualização empregadas.

As operadoras de telecomunicações demandam um desempenho previsível nas VNFs para que elas sejam capazes de atender aos *Service Level Agreement*s (SLAs), acordos sobre os níveis de serviço contratados. A virtualização permite a portabilidade com a capacidade de migrar máquinas virtuais, ou seja, fornecendo a habilidade de mover uma máquina virtual em execução de um servidor para outro.

Provedores de serviços de telecomunicações precisarão dessa *live migration*, ou migração ao vivo, para entregar uma alta disponibilidade quando realizarem a manutenção do sistema ou quando moverem uma VM para uma plataforma de computação com mais recursos. Além disso, a portabilidade tem como consequência a habilidade de desenvolver uma VNF onde quer que ela seja necessária. Então, consistência na infraestrutura é indispensável para ajudar nesse desenvolvimento, garantindo o desempenho para o cliente.

Virtualizar as funções de rede, testá-las e então certificá-las em várias plataformas de infraestrutura é um desafio importante para os fabricantes de equipamentos

de rede. Em muitos casos, eles oferecem plataformas de computação parceiras ou ecossistemas a fim de fornecer uma oferta integrada. A oferta de uma solução integrada faz com eles tenham que garantir que a VNF irá atuar como esperado, uma vez que isso será verificado em termos de interoperabilidade e desempenho. Além disso, as operadoras de telecomunicações necessitam de um desempenho previsível das funções de rede virtuais para que elas sejam capazes de atender aos SLAs.

Ademais, o conceito de computação da borda modifica o paradigma de topologia centralizada para descentralizada, pela utilização de recursos de computação, rede e armazenamento que estão mais próximos do usuário. Isso move o conteúdo e o serviço para longe dos nós centrais, ou seja, dos *datacenters* ou da nuvem para as extremidades lógicas da rede. Idealmente, seja a um único salto de distância do usuário ou quanto mais perto estiver a aplicação e o conteúdo, melhor será a qualidade de experiência do usuário. A eliminação de um ambiente completamente centralizado faz com que alguns gargalos e potenciais pontos de falha sejam removidos.

A *edge computing* busca a redução do tempo de resposta ou latência fazendo cache ou descarregando o conteúdo nas pontas. As vantagens da utilização desse conceito aliadas aos benefícios do NFV fazem com que novos serviços possam ser desenvolvidos e consequentemente a avaliação da qualidade desses serviços se torna necessária. Essas avaliações incluirão diferentes arquiteturas e seus requisitos específicos para as diversas áreas com potencial crescimento, como por exemplo a Internet das Coisas (*Internet of Things* (IoT)) com o conceito de *Fog Computing*, 5G com o conceito de *Mobile Edge Computing* (MEC), *Software Defined-Wide Area Network* (SD-WAN) com o roteamento baseado em políticas.

Com o surgimento da virtualização leve, tem-se mais alternativas concorrendo com a virtualização tradicional para implementação de serviços na borda da rede de diversas áreas de tecnologia. O contêiner é um ambiente de execução autocontido que compartilha o mesmo *kernel* do sistema operacional hospedeiro e que é isolado de outras instâncias de contêineres [3].

Ademais, os contêineres prometem acabar com a sobrecarga proporcionada pela virtualização completa, a qual utiliza técnicas como tradução binária para manipular os sistemas operacionais visitantes, além de uma grande flexibilidade no gerenciamento, migração, atualização e customização das imagens de sistema.

Análise de desempenho do plano de dados de NFV é um aspecto crítico para ajudar os *Internet Service Provider*s (ISPs) na transição para uma rede baseada em virtualização. Estruturas compartilhadas e ambientes multi-inquilino introduzem complexidades em cenários que devem ser testados e necessitam de avaliações mais completas devido a dependência introduzida por componentes de software da *Network Function Virtulization Infrastructure* (NFVI) e isolamento da camada de virtualização. Em ambientes compartilhados o desempenho da VNF não depende

somente da implementação dela e sim do conjunto de entidades que instanciam e gerenciam todo seu ciclo de vida.

Visando atender às necessidades da computação na borda, a presente dissertação analisa o desempenho das camadas subjacentes de diferentes funções de rede em uma plataforma x86. Uma vez que essas funções podem utilizar diferentes tipos de virtualização, um *benchmarking* foi realizado a fim de avaliar os possíveis gargalos destas soluções e qual é a mais apta para ser inserida na borda da rede garantindo um desempenho satisfatório aos usuários. O *benchmarking* é focado em cinco frentes consideradas as mais relevantes, as quais são: I/O de disco, memória, rede, aplicação e processamento.

Uma contribuição importante do presente trabalho é relacionada ao preparo das redes de telefonia móvel para o 5G. Uma vez que esta utilizará o conceito de Edge Computing para permitir um novo *QoS Class Identifier* (QCI) nas redes móveis relacionado ao valor máximo de latência necessário para determinada aplicação. Com a tendência de virtualização da *Radio Access Network* (RAN) para permitir uma maior flexibilidade do *fronthaul* das operadoras e prepará-las para funções como *Network Slicing*, surge o questionamento de qual tecnologia de virtualização utilizar na borda da rede e este trabalho tenta responder esta questão analisando tipos de virtualização atuais para inserção em um ambiente mais próximo do usuário.

Este trabalho é aplicável também a diversas áreas emergentes como IoT com o conceito de Fog Computing para permitir redes que ofereçam um contexto de usuário e assim gerando serviços customizáveis que fornecem, por exemplo, a localização de usuário em tempo real servindo como base para aplicações como carros autônomos ou cirurgias remotas. Além disso, o ganho para quem fornece os serviços fazendo com que sua infraestrutura de rede seja otimizada através da possibilidade de haver um planejamento de capacidade da rede diferente, já que o *hot data* será descarregado na borda e somente o *cold data* irá para o core contribuindo inclusive para a engenharia de tráfego de *backbones* de operadoras e diminuindo os gastos com infraestrutura.

A presente dissertação está organizada da seguinte forma. O Capítulo 2 apresenta os principais conceitos e diferentes tipos de virtualização. O Capítulo 3 apresenta o conceito de contêineres e algumas soluções que utilizam essa tecnologia. O Capítulo 4 explica o conceito de virtualização de funções de rede, enquanto o Capítulo 5 apresenta os trabalhos relacionados e suas principais diferenças em relação a presente dissertação. O Capítulo 6 apresenta a metodologia utilizada na avaliação de desempenho das ferramentas de implementação do NFV. Já o Capítulo 7 apresenta os resultados experimentais da análise de desempenho. Por fim, o Capítulo 8 conclui o trabalho e aponta direções de pesquisa futuras.

Capítulo 2

Virtualização

Na década de 1940, os primeiros sistemas com eletrônica digital não possuíam sistemas operacionais. Os computadores dessa época eram totalmente primitivos comparados aos de hoje e os programas eram frequentemente executados no computador bit a bit nas linhas dos concentradores mecânicos. Já na década de 50, os sistemas executavam uma tarefa por vez, o que permitia somente uma pessoa utilizar a máquina, ou seja, todos os recursos computacionais estavam disponíveis a um único usuário.

O primeiro grande sistema, considerado por muitos o primeiro sistema operacional, foi desenhado pelo laboratório de pesquisa da General Motors para seu mainframe IBM 701 no começo de 1956. Os sistemas operacionais são o software que faz o hardware tornar-se utilizável. O hardware fornece o poder computacional bruto. O sistema operacional faz com que esse poder computacional esteja convenientemente disponível para os usuários, através do gerenciamento do hardware cuidadosamente a fim de obter um bom desempenho.

Os sistemas operacionais também podem ser considerados como gerenciadores de recursos. Um sistema operacional determina quais recursos computacionais serão utilizados para resolver determinado problema e a ordem na qual eles serão utilizados. Em geral, os sistema operacionais possuem três funções básicas. A primeira é a alocação e atribuição de recursos computacionais como dispositivos de I/O, software, unidades de processamento central, etc. A segunda função básica é escalonamento de tarefas coordenando recursos e tarefas seguindo uma prioridade determinada. A terceira é a monitoração das tarefas do sistema mantendo o histórico de operações, notificando os usuários ou operadores conforme alguma condição de erro é encontrada.

A virtualização surgiu em 1960 a partir da IBM, que tinha como objetivo particionar os grandes *mainframes* da época em instâncias lógicas que seriam executadas em um único hardware de um determinado *mainframe* hospedeiro. Essa necessidade da IBM surgiu da incapacidade dos *mainframes* de aproveitar seu caro tempo de

cálculo, ou seja, somente um usuário poderia utilizá-lo por vez. Com isso, a virtualização veio juntamente com o conceito de *time sharing* (compartilhamento de tempo), no qual múltiplos usuários utilizavam o mesmo hardware para aumentar a eficiência do trabalho conforme é visto na Figura 2.1.

Figura 2.1: Diversos usuários utilizando os mesmo recursos de hardware.

No final da década de 70 os primeiros chips foram lançados pela Intel e na década seguinte os microprocessadores foram responsáveis pela mudança no foco de algumas empresas e assim deram o primeiro passo para a popularização do microcomputador e da arquitetura x86 que é utilizada até hoje nos computadores pessoais e nos *datacenters*, obviamente com muita evolução ao longo dos anos. O termo x86 se refere a um conjunto de instruções de linguagens para a máquina presente em certos processadores da Intel e de algumas outras companhias. Ele define essencialmente as regras de uso e vocabulário para o chip.

Um datacenter é um espaço físico responsável por uma modalidade de serviço de valor agregado que oferece recursos de processamento e armazenamento de dados em larga escala para que organizações de qualquer porte e até mesmo profissionais liberais possam ter ao seu alcance uma estrutura de grande capacidade, flexibilidade, alta segurança e igualmente capacitada do ponto de vista de hardware e software para processar e armazenar informações.

Soluções de virtualização multiplicam o acesso ao hardware do computador permitindo que diferentes fatias virtuais sejam executadas sobre ele. Em um ambiente de *datacenter* legado, sem virtualização, um simples computador poderia somente executar um único sistema operacional por um certo tempo. Esse sistema seria responsável por controlar todos os dispositivos de hardware do computador, como CPU, memória, controladores de disco, disco rígido, placas de vídeo, placas de rede, e outros dispositivos periféricos. Com a virtualização, múltiplos usuários têm a ilusão do controle total sobre o hardware cada um em sua fatia virtual, ou seja, em seu próprio espaço isolado de maneira lógica.

Essa abordagem ainda é empregada em muitos *datacenters*, quando somente

um serviço é oferecido por uma dada máquina, geralmente para obter um nível elevado de desempenho. Contudo, tanto espaço físico no armário quanto consumo de energia, escalabilidade e atualizações de hardware são alguns pontos que necessitam de atenção. A virtualização pode resolver esses desafios permitindo que diferentes sistemas operacionais compartilhem o mesmo hardware e então consolidando fatias virtuais em uma única máquina física. Sendo assim, cada fatia virtual possui uma quantidade dedicada de recursos do servidor físico.

A virtualização é peça fundamental para a computação na nuvem e o NFV. Os seus principais benefícios incluem independência de hardware, isolamento, ambientes de usuário seguros e aumento da escalabilidade. Recentemente, observou-se um aumento no número de soluções e de tecnologias de virtualização fazendo com que as chamadas soluções tradicionais baseadas em hipervisores hoje concorram com a virtualização leve baseada em contêineres que, em tese, possui menor sobrecarga computacional. Além disso, houve a introdução de novas técnicas híbridas que prometem combinar as vantagens das anteriores. As seções seguintes fazem uma breve revisão destes conceitos.

2.1 Virtualização Tradicional

Uma solução de virtualização tradicional, também conhecida como virtualização completa, emprega uma camada de software chamada de hipervisor, que se situa entre o hardware e as máquinas virtuais. O hipervisor controla o acesso ao hardware e realiza a alocação de recursos para cada VM, além de fornecer isolamento entre elas. Com isso, as VMs utilizam diferentes sistemas operacionais e hospedam suas próprias aplicações. Em geral, é possível dizer que cada VM possui seu próprio ambiente computacional virtualizado, isto é, seu próprio sistema operacional e sendo assim sua própria abstração de hardware.

Há hipervisores conhecidos como do tipo I, os quais são executados diretamente no hardware ou usualmente chamado de *bare-metal*. Esse tipo de hipervisor utiliza uma virtualização completa baseada em hardware, como pode ser visto ver na Figura 2.2(a), implementando uma modificação de um sistema operacional e assim tornando desnecessário um sistema operacional hospedeiro intermediário, uma vez que a própria solução é quem hospeda os diversos sistemas operacionais visitantes.

Essa modificação de sistema operacional é geralmente um *kernel* desenvolvido por empresas seguindo o modelo de sistema operacional *Portable Operating System Interface* (POSIX) para fornecer funcionalidades similares que podem ser encontradas em outros sistemas operacionais, como criação e controle de processos, sistemas de arquivos e *threads*. Um exemplo é o `vmkernel` desenvolvido pela VMWare para seu hipervisor VMWare ESXi que atua como um sistema operacional executado

diretamente no hardware e gerencia recursos como memória, processadores físicos, armazenamento e controladores de rede.

Por outro lado, há hipervisores do tipo II, que utilizam uma virtualização completa baseada em software, como é ilustrado na Figura 2.2(b), sendo executados sobre um sistema operacional hospedeiro. É este sistema operacional hospedeiro que é executado diretamente sobre o hardware gerando mais uma camada de sobrecarga ao sistema.

Apesar de adicionar mais uma camada de abstração no hardware o que acarreta em uma perda de desempenho, os hipervisores do tipo II possuem a vantagem de suportar uma vasta gama de hardware hospedeiro. Isso se deve ao fato de que é o sistema operacional hospedeiro que trata o gerenciamento do hardware. Além disso, os hipervisores do tipo II geralmente são bem mais fáceis de instalar que os hipervisores do tipo I.

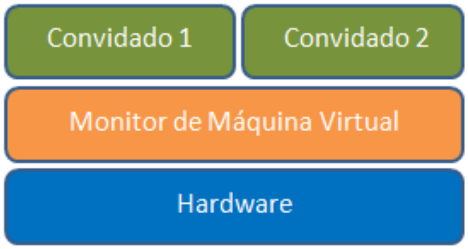
Um conceito importante é o de modelo de proteção, uma vez que CPUs modernos utilizam um esquema de proteção de recursos onde vários anéis de proteção são criados e cada um deles representa um nível diferente de confiança na execução de tarefas. A arquitetura x86 oferece 4 anéis, numerados de 0 a 3 e quanto menor o número do anel, maior é a representação do nível de confiança no código. Muitas vezes os números do anéis são referenciados pelo seu *Current Privilege Level* (CPL). A parte do sistema operacional, a qual necessita da maioria de privilégios executa no anel 0, os drivers dos dispositivos executam no anel 1 ou acima e as aplicações no anel 3, conforme é observado na Figura 2.3(d).

Além disso, algumas instruções como as I/Os do sistema necessitam de privilégios elevados. Quando uma aplicação do usuário necessita executar essas instruções, uma função do sistema operacional é chamada, a qual realiza a execução do código no kernel do sistema operacional, localizado no anel 0. Uma vez que essa função termina sua execução, esta retorna o resultado obtido e a execução do código da aplicação continua, com os privilégios anteriores de menor valor. Tal técnica é conhecida como *trap and emulate*, já a camada de menor privilégio, onde as aplicações são executadas é o espaço do usuário e a camada com o maior privilégio é conhecida como espaço do *kernel*.

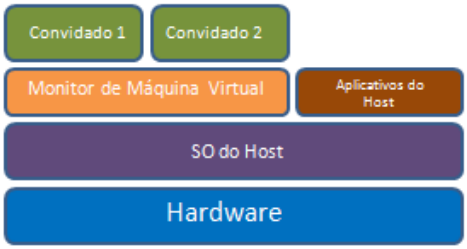
Algumas outras técnicas como *binary translation*, *direct execution* e *binary patching* são frequentemente utilizadas nas soluções de virtualização. Na *binary translation* as instruções sensíveis no código do sistema operacional visitante são substituídas tanto por *hypercalls*, as quais manipulam de forma segura as instruções quanto por `opcodes` (códigos de operação) indefinidos que resultam em uma *trap* do CPU e esta, por sua vez, é manipulada pelo hipervisor.

Ademais, a técnica *direct execution* faz com que somente o código do *kernel* seja traduzido, evitando assim a sobrecarga advinda da tradução de partes desnecessárias

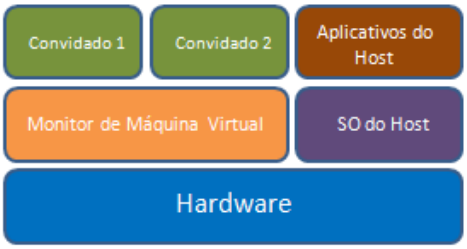
da aplicação pela técnica *binary translation*. Já a técnica de *binary patching* é similar a *binary translation*, exceto pelo fato de ser mais leve uma vez que o código é alterado ao invés de ser traduzido o que gera uma alteração na memória do sistema visitante.



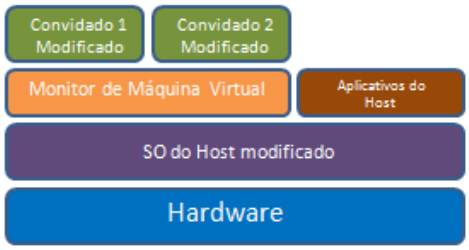
(a) Virtualização baseada em Hardware.



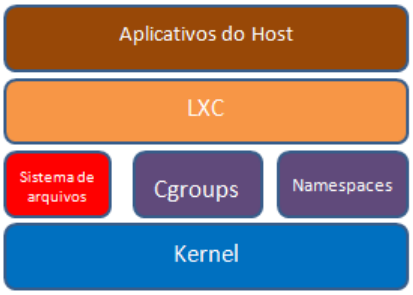
(b) Virtualização baseada em Software.



(c) Virtualização híbrida.



(d) Paravirtualização.



(e) Virtualização leve.

Figura 2.2: As diferentes arquiteturas de virtualização, desde a tradicional até a virtualização leve.

2.2 Paravirtualização

A paravirtualização foi feita para superar as restrições da arquitetura de processadores baseada em modo de proteção. Para melhorar a comunicação entre o sistema operacional do visitante e o hipervisor, a proposta de paravirtualização modifica o *kernel* (núcleo) do sistema operacional visitante substituindo as instruções não-virtualizáveis por *hypercalls* que se comunicam diretamente com uma camada de virtualização no hipervisor, como ilustrado na Figura 2.2(d). O hipervisor também fornece interfaces de *hypercalls* para outras operações críticas do núcleo do sistema

operacional como gerenciamento de memória, manipulação de interrupções e manutenção de tempo.

A paravirtualização diferencia-se da virtualização tradicional, onde o sistema operacional visitante sem modificações não sabe que é virtualizado e as chamadas sensíveis do sistema operacional são tratadas utilizando a técnica de tradução binária (*binary translation*). Uma vantagem da paravirtualização é a diminuição dos gargalos decorrentes da virtualização completa, entretanto a diferença de desempenho entre ambas pode variar de acordo com a carga aplicada. Como a paravirtualização não suporta sistemas operacionais sem modificação, sua compatibilidade e portabilidade é fraca. A ferramenta mais conhecida desse tipo de virtualização é o Xen [4].

O Xen é um hipervisor *open source* desenvolvido para executar em servidores *Commercial off-the-shelf* (COTS). O Xen permite executar múltiplas VMs simultaneamente em uma única máquina física. Além disso, ele implementa a técnica de paravirtualização que melhora o desempenho do sistema visitante através da troca do seu comportamento a fim de acionar o hipervisor quando necessário, evitando a necessidade da tradução binária das instruções do sistema. A arquitetura do Xen é composta por um hipervisor localizado acima do hardware físico e máquinas virtuais sobre o hipervisor.

Cada máquina virtual possui seu próprio sistema operacional e aplicações quando executadas no Xen. O hipervisor controla o acesso ao hardware e também gerencia os recursos compartilhados disponíveis para as máquinas virtuais. Adicionalmente, os drivers do dispostivo são colocados em uma máquina virtual isolada, chamada *Domain 0*(dom0), a fim de fornecer confiabilidade e eficiência ao hardware.Devido ao dom0 possuir acesso total ao hardware físico, ele possui privilégios especiais comparado a outras máquinas virtuais, referenciadas como *user domains*(domUs). Por outro lado, os *user domains* possui drivers virtuais, chamados de *front-end drivers*(fe), os quais se comunicam com os *back-end drivers*(be) localizados no dom0 para acesso o hardware físico.

2.3 Virtualização assistida por hardware

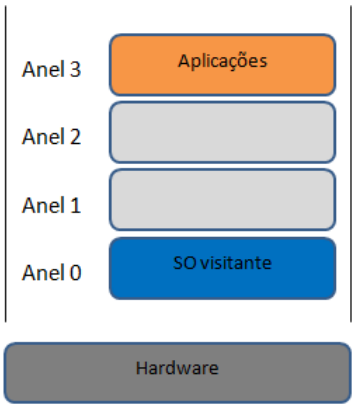
A virtualização assistida por hardware foi criada para superar as restrições da paravirtualização e da virtualização completa através da criação de um mecanismo de controle robusto no CPU. Este dispensa o uso de técnicas como *ring compression* que retira privilégios do sistema operacional visitante colocando-o para executar no anel 1 e também dispensa a modificação de código no sistema operacional visitante.

A tecnologia de virtualização assistida por hardware foi desenvolvida para possibilitar alto desempenho do hipervisor sem necessitar de mudanças decorrentes da paravirtualização ou de técnicas como tradução binária. Com isso, possibilita uma

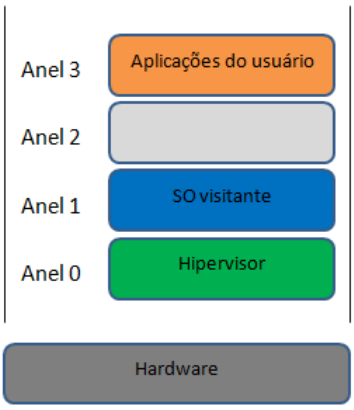
implementação de hipervisor que pode suportar uma vasta gama de sistemas operacionais visitantes sem modificação.

Para um hipervisor usufruir das vantagens de tecnologias como a Intel VT-x, ele é executado em um novo modo de CPU chamado de *VMX Root* e o sistema operacional visitante é executado no modo *VMX Non-root*. Assim, o *Virtual Machine Monitor* (VMM) gerencia as VMs através de um mecanismo de entrada e saída, conforme mostra a Figura 2.3(d).

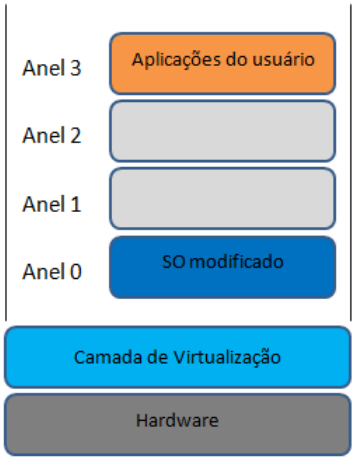
A tecnologia Intel VT-x inclui duas formas de operação para que o hipervisor possa ter um controle granular sobre o comportamento da CPU, essa instrução do ponto de vista do processador é o estado VMX-root que significa a possibilidade de execução de qualquer instrução no processador sem nenhuma restrição, inclusive somente neste contexto se pode alterar o estado do sistema.



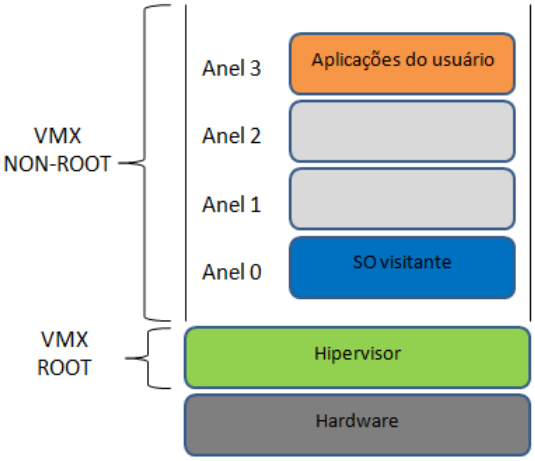
(a) Sem virtualização.



(b) Virtualização completa.



(c) Paravirtualização.



(d) Virtualização assistida por hardware.

Figura 2.3: Os diferentes tipos de acesso ao recurso de hardware.

Adicionalmente, quando o estado de execução é VMX-non root significa que o processador está configurado para entender que a execução neste contexto é de

uma máquina virtualizada. Com isso, algumas instruções, as mais privilegiadas, são desabilitadas e os ponteiros do processador deste contexto serão armazenados em cache dentro do próprio processador a fim de acelerar o compartilhamento do tempo de processamento de outras máquinas virtuais, já que a gravação e recuperação no cache do processador são muito mais velozes do que se fosse na memória principal.

A operação do estado VMX non-root e as transições entre os estados são controladas pela estrutura de dados conhecida como *Virtual Machine Control Structure* (VMCS). O acesso ao VMCS é controlado através de um componente do processador chamado VMCS *pointer* e existe um para cada processador lógico. O VMCS gerencia os modos de operação anteriormente citados e contém campos para lidar com os estados das transições, para informar quais são as exceções permitidas ou proibidas, para informar a motivo das transições e *flags* para alterar o comportamento das transições.

2.4 Virtualização Leve

A virtualização leve, também chamada de virtualização baseada em contêineres ou de virtualização a nível de sistema operacional, fornece um nível diferente de abstração em termos de virtualização e isolamento quando comparada aos hipervisores. Conforme mencionado anteriormente, os hipervisores fornecem uma abstração de hardware que resulta em uma sobrecarga em termos da própria virtualização do hardware e dos *drivers* virtuais do dispositivo. Isso significa que cada instância de VM é totalmente implementada com hardware virtual para suportar um sistema operacional visitante não modificado, executado sobre o hipervisor.

Diferente da virtualização completa, na virtualização leve é fornecido um isolamento de processos no nível do sistema operacional evitando sobrecarga da camada de abstração do hardware virtual para cada VM instanciada, como pode ser visto na Figura 2.2(e). Com isso, as imagens são executadas sobre o mesmo *kernel* do sistema operacional compartilhado no servidor físico, sendo que um ou mais processos podem executar dentro de cada imagem.

Sendo assim, a virtualização leve é considerada mais leve e com melhor desempenho que a virtualização tradicional. Apesar de fornecer menor capacidade de isolamento que a virtualização completa ou a paravirtualização, ela é considerada com um nível de isolamento suficiente para cenários de gerência de recursos em IoT devido a sua capacidade de ser executada em ambiente com baixo recursos computacionais [5].

Uma desvantagem da técnica de implementação da virtualização leve ao utilizar o mesmo *kernel* quando comparada as VMs é relacionada a questões de segurança. Diferente da virtualização tradicional, a virtualização leve expõe as tabelas de cha-

mada do sistema do hospedeiro para cada visitante e confia em ponteiros para redirecionar essas chamadas de sistemas para estruturas de dados isoladas, chamadas *namespaces*. Com isso, é aberta a possibilidade de explorar vulnerabilidades desse ponteiro a fim de obter informações sigilosas ou obter um privilégio maior.

Outra desvantagem na utilização do mesmo *kernel* é que uma determinada vulnerabilidade no *kernel* hospedeiro é compartilhada com todas as instâncias, sendo assim possuem o mesmos problemas de sistemas monolíticos que foram escritos com linguagens inseguras, as quais fizeram os usuário optarem por VM para um isolamento seguro mesmo considerando o compromisso entre desempenho e segurança.

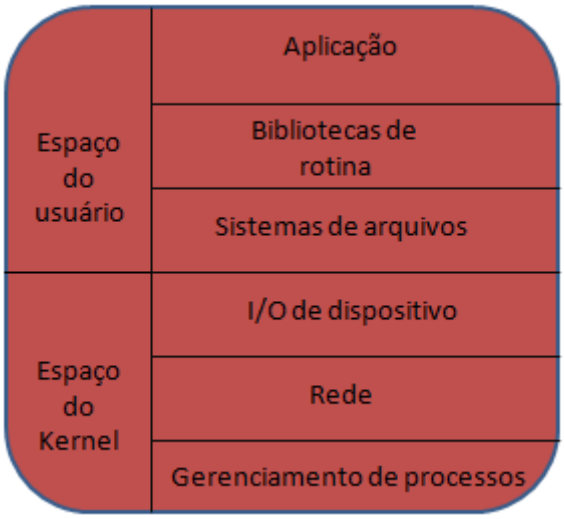
2.4.1 Unikernel

Um sistema operacional pode ser decomposto em dois espaços de endereçamento: o espaço do *kernel* e o espaço do usuário. O espaço do *kernel* contém as funções utilizadas pelo sistema operacional e bibliotecas compartilhadas incluindo funções de baixo nível como I/O de disco, acesso ao sistema de arquivos, gerenciamento de memória, bibliotecas compartilhadas, etc. Ele também fornece um isolamento de processos, catálogo de processos e outras funções necessárias para sistemas operacionais multiusuário. O espaço do usuário, por outro lado, contém o código da aplicação como é visto na Figura 2.4(a).

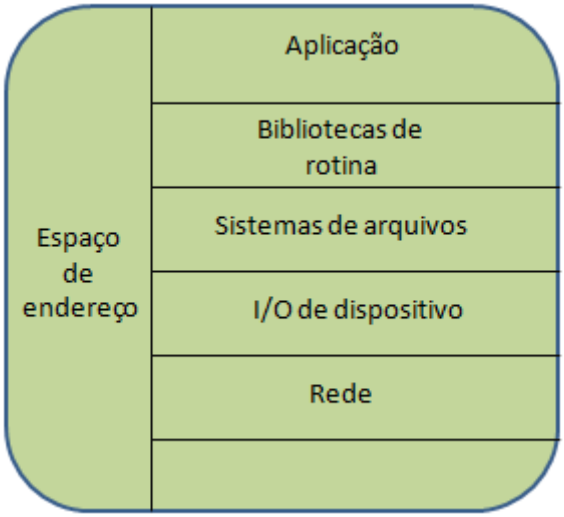
Ao utilizar Unikernel, não há divisão entre o espaço do usuário e o espaço do kernel, como pode ser visto na Figura 2.4(b). Apesar disso parecer uma pequena diferença, é de fato o oposto, uma vez que a pilha de protocolos do sistema operacional é uma combinação do kernel, bibliotecas compartilhadas e uma aplicação para obter seu objetivo final. Só existe um programa sendo executado, e ele contém desde o código em alto nível da aplicação até as rotinas em baixo nível de I/O do dispositivo, sendo uma imagem singular e autocontida que não requer nada adicional para que seja inicializada.

O Unikernel pode ser considerado um tipo de virtualização leve, porém implementado de uma maneira diferente da visão compartilhada no mesmo sistema operacional anteriormente citada. Sendo assim, é um sistema operacional que utiliza um conceito verdadeiramente minimalista para criação da pilha de software, ou seja, preocupa-se somente com as necessidades específicas para cada aplicação. No design do sistema operacional, as bibliotecas utilizadas para construí-lo são as estritamente necessárias para funcionamento da aplicação, com isso tornando-o mais leve do que as tradicionais VMs.

O Unikernel é conhecido como sistema operacional de bibliotecas, não é uma ideia nova, porém não era amplamente implementado devido à incapacidade de tratar com inúmeros hardwares diferentes. Entretanto, com a popularização dos



(a) Sistema operacional tradicional.



(b) Sistema operacional com Unikernel.

Figura 2.4: Diferentes projetos de sistemas operacionais.

hipervisores, essa preocupação de lidar diretamente com o hardware foi superada tornando possível um sistema operacional com essas características ser executado em um ambiente de nuvem.

Uma das vantagens do Unikernel em relação a outras ferramentas é que ele melhora a segurança por ser um sistema operacional muito pequeno e específico a ponto de carecer de ferramentas internas adicionais para serem utilizadas a fim de ajudar em uma possível invasão. Ademais, seu desempenho é elevado devido a sua leveza, a inexistência de processos múltiplos com necessidade de gerenciamento e a exclusão de mudanças de contextos de usuário na sua implementação, entretanto essa ferramenta não foi utilizada no presente trabalho.

2.5 Virtualização Híbrida

Além dos casos relatados anteriormente, existe a virtualização híbrida que mistura os tipos de virtualização anteriores, sendo ilustrada na Figura 2.2(c). A virtualização híbrida não é limitada a um tipo de virtualização específica e sim é uma prática na qual mistura-se mais de um tipo de virtualização para possivelmente melhorar algum aspecto específico, como por exemplo segurança. Um exemplo é o ClearContainers [6] da Intel agora integrado ao KataContainers [7] da comunidade Openstack que mistura virtualização leve, paravirtualização e virtualização assistida por hardware para melhorar o desempenho e a segurança.

O Clearcontainers é um projeto que nasceu em 2015 para lidar com uma falha de segurança chamada *DirtCow* dentro dos contêineres que explorava a técnica *Copy-on-Write* (CoW) para obter privilégios indevidos no sistema. A solução foi criada através da utilização da tecnologia Intel VT-x que é um tipo de virtualização assistida por hardware explicada anteriormente a fim de fornecer mais segurança ao ambiente executando os contêineres como se fossem máquinas virtuais leves e compatíveis com ferramentas de orquestração já existentes como Kubernetes [8].

No final de 2017, o projeto ClearContainers foi incorporado ao KataContainers que acrescentou um ambiente de execução chamada *runV* [9], o qual é compatível com as normas da *Open Container Initiative* (OCI) fazendo com que os contêineres possam ser executados em diferentes plataformas e possuam um sistema operacional leve e um *kernel* dedicado.

Adicionalmente a iniciativas de execução de contêineres juntamente com virtualização assistida por hardware, há também outras frentes de trabalho mais antigas como a do projeto Xenserver [10], o qual utiliza a paravirtualização, que adaptam sua solução para utilizar a virtualização assistida por hardware que já está presente em grande parte dos hardwares atuais, contêineres e Unikernel como no projeto MirageOS [11] e no *plugin xscontainer* [12].

Uma outra iniciativa famosa que pode utilizar mais de um tipo de virtualização é o *Kernel-based Virtual Machine* (KVM) [13] que é baseado na ferramenta *Quick Emulator* (QEMU) [14], uma vez que nele os dispositivos podem ser paravirtualizados através da *Application Programming Interface* (API) Virtio [15] que fornece uma camada de abstração evitando que os dispositivos sejam emulados, o que acarreta em uma sobrecarga maior.

O QEMU é um emulador open source que pode emular desde dispositivos até aplicações. Ele é executado no Linux, Windows e OS X e possibilita que sistemas hospedeiros executem programas escritos para arquiteturas diferentes. Entretanto, há sobrecarga em relação a desempenho devido ao uso da técnica *dynamic binary translation* mencionada anteriormente.

O hipervisor KVM é uma extensão do *kernel* do Linux e trabalha expondo uma API, a qual pode ser utilizada para interagir com o *kernel* e com o hardware. Essa API, por sua vez, é então utilizada por um *client* tal como o QEMU. Isso permite a uma instância de software hospedada no QEMU ser executada mais rápido, uma vez que a virtualização assistida por hardware é um requisito.

Recentemente uma outra estratégia adotada foi a mistura da virtualização leve utilizando contêineres com a virtualização assistida por hardware e com o projeto Unikernel para fornecer a ferramenta Docker nas plataformas Windows e Mac evidenciando a importância da virtualização híbrida nos dias atuais.

Capítulo 3

Contêineres

Apesar de o termo ter se tornado conhecido nos últimos anos, durante décadas já se utilizava o conceito de contêineres em sistemas Unix, através do comando `chroot` que era apenas uma forma de isolar o sistema de arquivos (*filesystem*). Em computação, um *filesystem* é um método para guardar e organizar arquivos e dados contidos para tornar fáceis a procura e o acesso. Um sistema de arquivos é utilizado sobre um dispositivo de armazenamento de dados como um disco rígido a fim de preservar a localização física dos arquivos.

Em seguida, vieram os `jails` do sistema operacional FreeBSD que, além do isolamento do sistema de arquivos, permitiam também o isolamento de processos. O `chroot` era limitado e ao longo do anos foram descobertas formas de burlar sua segurança, com isso os `jails` vieram para melhorar a segurança e expandir o isolamento para um conjunto determinado de usuários e subsistemas de rede.

Já a Sun Microsystems desenvolveu o Solaris Zones, mais uma solução baseada em contêineres, porém somente para sistemas Solaris. O grande passo rumo ao cenário dos dias de hoje foi a criação de um painel que permitia o fácil gerenciamento de contêineres e a disponibilização do núcleo de um sistema, chamado Virtuozzo como código aberto com o nome de OpenVZ [16].

O OpenVZ foi criado pela empresa Parallels e é outra tecnologia baseada em virtualização leve. É considerado o antecessor do *Linux Containers* (LXC) tanto que uma parte do código do LXC é derivado do OpenVZ ou com contribuição do time do OpenVZ. A diferença do OpenVZ para os demais é que o seu kernel era proprietário e derivado do *kernel* Linux 2.6, porém após um certo tempo ele foi desenvolvido para o núcleo Linux podendo ser executado em ambos. Além disso, ele possui funções como *live migration* e foi muito utilizado em cenários de produção para *Virtual Private Server* (VPS).

Uma desvantagem do OpenVZ era a necessidade de aplicar um *patch* ao *kernel* Linux. Após o surgimento do OpenVZ, o Google iniciou o desenvolvimento dos *Control Groups* (cgroups) para o *kernel* do Linux e iniciou a utilização de contêineres

em seus *datacenters*. Logo depois, surgiu o LXC que trazia consigo os cgroups para limitar os recursos computacionais, os *namespaces* para isolar os processos e o *chroot* fornecendo uma solução completa para a criação e gerenciamento de contêineres.

A virtualização baseada em contêineres é considerada uma alternativa leve à virtualização baseada em hipervisor. Contêineres implementam isolamento de processos a nível do sistema operacional da máquina hospedeira. Assim, evita-se a sobrecarga devido ao hardware virtualizado e aos *drivers* virtualizados de dispositivos. Um contêiner pode ser considerado um ambiente virtual pequeno e isolado, o qual inclui um conjunto de dependências específicas necessárias para executar determinada aplicação.

Os processos dentro dos contêineres não conseguem ver outros processos externos e cada um dos contêineres tem a sua própria árvore de diretórios. Sendo assim, na perspectiva da aplicação o contêiner não difere da virtualização tradicional porque em ambos os casos a aplicação não sabe que está sendo executada em um ambiente virtualizado. O mecanismo para proporcionar a construção desse ambiente isolado chamado contêiner é constituído por ferramentas como os *namespaces* e os cgroups.

3.1 Namespaces

Os *namespaces* são umas das principais ferramentas do *kernel* Linux para permitir a virtualização leve. A proposta de cada *namespace* é separar uma parte dos recursos de sistema global em uma abstração que faz com que pareça para os processos dentro de um *namespace* que eles possuem sua própria instância isolada de recursos do sistema hospedeiro.

O primeiro *namespace*, de montagem, foi introduzido no Linux kernel versão 2.4.19 em 2002 e atualmente o Linux possui *namespaces* para identificadores de processos, comunicação inter-processos, rede, hostname, ponto de montagem, identificadores de usuário e cgroups.

Os *namespaces* são utilizados para criar múltiplos espaços de usuário isolados no kernel hospedeiro. Além disso, os namespaces permitem que processos enxerguem um diretório arbitrário como sendo seu diretório raiz. Processos podem trabalhar com privilégio de superusuário dentro de um *namespace* sem colocar em perigo o sistema hospedeiro fora dele. Adicionalmente, os *namespaces* expandem o isolamento para outras partes e funções do *kernel*.

Tradicionalmente, o sistema operacional Linux pode somente lidar com uma única árvore de processos, a qual consiste em processos pais e filhos. Diferentes processos podem possuir diferentes privilégios podendo inspecionar uns aos outros na árvore e cada um deles possui um único identificador de processo (*Process Identifier* (PID)). Quando o Linux inicia, ao primeiro processo é atribuído o PID 1 e esse

processo é considerado como um processo *init* do sistema. Além disso, o processo de inicialização executa e mantém todos os outros *daemons* e serviços necessários.

Os *namespaces* de PID são utilizados para virtualizar as árvores de processos. Estes, por sua vez, dentro do *namespace* de PID, são isolados dos outros processos do hospedeiro assim como dos processos em diferentes *namespaces* de PID. Os *namespaces* de PID são hierárquicos, o que significa que cada processo também é capaz de enxergar todos os processos dentro de seus *namespaces* filhos. Por exemplo, o hospedeiro pode ver os processos de todos os *namespaces* na sua árvore de processos original. Processos dentro de um *namespace* de PID tem um espaço para um PID isolado e eles podem ter seu próprio processo *init* com PID 1. Isso significa que os processos podem possuir múltiplos processos atribuídos a eles, um para cada árvore de processo hospedeira e outro processo que é visível dentro do *namespace*.

Processos executando dentro de um *namespace* podem precisar se comunicar com outros processos de fora para funcionarem corretamente. Essa comunicação inter-processos é feita por *namespaces* de *Interprocess Communication* (IPC) que permitem comunicações somente entre os processos pertencentes ao mesmo *namespace* de IPC. Isso previne a interferência de processos em *namespaces* distintos.

Os *namespaces* de rede fornecem isolamento de recursos do sistema relacionados a rede. Então, cada *namespace* de rede possui seus próprios dispositivos de rede, endereços de *Internet Protocol* (IP), tabela de roteamento, diretórios `/proc/net`, números de porta e etc. Através do uso de *namespaces* de rede, processos diferentes podem enxergar diferentes interfaces de rede. Os *namespaces* de rede são capazes de fornecer novas interfaces de rede e endereços virtuais para seus processos. Processos hospedeiros e processos em diferentes *namespaces* podem se comunicar uns com os outros utilizando essas interfaces virtuais como se estivessem se comunicando com usuários externos.

Já os *namespace*s de montagem são úteis para determinar quais pontos de montagem podem ser vistos dentro dos *namespaces*. Um processo consegue inicialmente ver os mesmos pontos de montagem que o hospedeiro, contudo, com *namespaces* de montagem, os processos podem montar e desmontar terminais sem afetar os pontos de montagem do hospedeiro.

Adicionalmente, os *namespaces* de usuário são utilizados para executar os processos que possuem identidades de usuário ou de grupo diferentes. O *namespace* de usuário também permite que um processo tenha privilégios de superusuário dentro do *namespace* sem ter os mesmos privilégios fora deste *namespace*.

3.2 Control Groups

Os *Control Groups* (cgroups) servem para gerenciar o uso do hardware e do software, como por exemplo CPU, memória, disco e rede de grupos de processos específicos. Isso é feito pela atribuição de um conjunto de processos dentro de grupos hierárquicos que possuem regras específicas para seu comportamento tendo como exemplo o limite de quanto de CPU os processos podem consumir.

O projeto de cgroups iniciou através de dois engenheiros do Google em 2006, no ano seguinte foi incorporado ao *kernel* do Linux. Entretanto, a comunidade alterou o conceito de cgroups e ele foi redesenhado e incluído na versão 4.5 do *kernel* do Linux em 2016.

Com os cgroups é possível garantir que certos grupos de processos não possam consumir muitos recursos computacionais. Além disso, são utilizados para garantir que um grupo específico de processos é capaz de obter determinada quantidade de recursos de computação. Os cgroups se tornaram realmente úteis quando eles foram combinados com os *namespaces*, servindo como base para a implementação de contêineres.

A partir do momento em que os cgroups foram combinados com os *namespaces* foi possível determinar a quantidade de recursos computacionais que cada contêiner tem permissão de utilizar. Isso aproximou a virtualização a nível de sistema operacional da virtualização tradicional, onde as aplicações de recursos computacionais podem ser atribuídas para máquinas virtuais diferentes. Os cgroups também fornecem métricas para utilização de recursos computacionais e essas métricas, por sua vez, podem também ser utilizadas para outros fins, a título de exemplo para tarifação (*billing*).

3.3 LXC

O LXC é considerado o primeiro método de conteinerização real que foi incluído no *kernel* do Linux. Apesar de haver outras soluções que também fazem uso da virtualização a nível de sistema operacional, elas não eram tão ativas na integração com o *kernel* do Linux.

Ademais, o LXC utiliza o mesmo tipo de isolamento que o *chroot*, contudo o LXC utiliza os namespaces do Linux para expandir seu isolamento tendo em conta processos e rede. O LXC também utiliza os cgroups para gerenciar recursos e para utilizar algumas funções de segurança, como políticas *seccomp* que são oferecidas pelo *kernel* do Linux para limitar a quantidade de *system calls* que determinado processo pode realizar.

O LXC oferece um isolamento lógico completo para os processos executados nos

contêineres. Os contêineres são isolados do *host*, assim como dos outros contêineres. Os contêineres LXC compartilham o mesmo *kernel* subjacente do hospedeiro e portanto não é necessário um sistema operacional adicional dentro dos contêineres. Através da utilização do LXC é possível compartilhar os dispositivos do hospedeiro e as imagens com os contêineres.

Além disso, o LXC oferece um conjunto de ferramentas que podem ser utilizadas para gerenciar os contêineres. Isso inclui ferramentas para criação, destruição, inicialização, pausa de contêineres, clonagem e monitoração dos contêineres. Contudo, o LXC é um tecnologia mais antiga quando comparada a outros gerenciadores de contêineres, como por exemplo o Docker que foi feito originalmente utilizando o LXC como *driver* principal. O LXC não tem uma comunidade tão grande para suporte como o Docker, o suporte do LXC geralmente é mais focado na distribuição Ubuntu.

O objetivo do LXC é criar um ambiente o mais próximo possível de uma distribuição Linux padrão sem precisar separar o *kernel*. A sua arquitetura pode ser vista na Figura 2.2(e), desconsiderando a camada superior adicionada pelo Docker [17]. Algumas características como leveza e versatilidade tem facilitado o uso dos contêineres em diferentes contextos abrangendo desde computação na nuvem até cenários de IoT, passando por NFV.

3.4 Docker

Conforme dito anteriormente, o conceito de contêineres não é novo no mundo da virtualização, mas tem adquirido maior relevância com a crescente adoção do sistema Docker. O Docker introduz um mecanismo de camadas subjacentes junto a uma API funcional que permite facilmente criar, gerenciar e remover a aplicação dentro do contêiner. Devido à pequena sobrecarga adicionada, vários contêineres podem ser executados mesmo em dispositivos com recursos computacionais limitados, tais como plataformas *Single Board Computer* (SBC).

O Docker é baseado no sistema operacional Linux e permite utilizar o modelo *Representational State Transfer* (REST) para gerenciar os contêineres através das suas imagens Docker. Um contêiner Docker é semelhante a uma máquina virtual, contudo o contêiner Docker é talvez mais eficiente que a VM porque ele elimina o hipervisor e o sistema operacional do visitante completamente enquanto mantém os contêineres portáveis através do *kernel* do sistema operacional Linux.

A significativa melhoria do contêiner em relação à VM não é limitada a desempenho, mas também a densidade de desenvolvimento. A natureza leve do contêiner torna a densidade de desenvolvimento mais rápida que a de VM. Alta densidade de desenvolvimento em computação na nuvem significa que menos máquinas físicas são

necessárias para atender à mesma quantidade de clientes, com isso há uma redução de custos de CAPEX e OPEX para as operadoras de *cloud*.

Uma desvantagem do Docker é a falta de maturidade da tecnologia em comparação a VMs já conhecidas há décadas, além disso há uma desconfiança em relação a capacidade do Docker de fornecer segurança uma vez que o mesmo *kernel* do sistema hospedeiro é compartilhado com as instâncias visitantes. Ademais, outra desvantagem para implantação do Docker em ambientes de produção é o esforço que as companhias terão que fazer para alterar suas práticas comuns de gerenciamento, desenvolver políticas de automação para o Docker, alterar a governança de áreas técnicas, obter treinamento para equipes se qualificarem confiando em uma empresa que há muito pouco tempo desenvolveu uma versão de software com suporte garantido conhecida como Docker Enterprises Edition.

Além do Docker e LXC, o Rocket [18], o Kurma [19] e o Jetpack [20] são ferramentas para execução de contêineres de aplicações no Linux. O mais completo é o Rocket que é integrado com ferramentas de orquestração de *clusters* e totalmente compatível com outras ferramentas para contêineres linux, como por exemplo o Docker. Isso significa que o Rocket pode executar imagens Docker. Ele foi construído utilizando uma especificação chamada `appc` feita pela CoreOS que tornou-se obsoleta com a criação dos padrões da *Open Container Initiative* (OCI), portanto a ferramenta não foi utilizada no presente trabalho devido a sua base ser uma especificação obsoleta.

A escolha do Docker é devido a ser atualmente a ferramenta mais utilizada relacionada a virtualização leve e a que possui a maior comunidade para suporte e integração com diversas ferramentas de orquestração. Além disso, é a ferramenta que possui o maior ecossistema em torno das suas funções desde a contribuição para a criação de um padrão para contêineres com a OCI, eventos globais de incentivo ao uso da tecnologia, aquisições de empresas para impulsionar a cultura de utilização da ferramenta e parcerias com *Over the Top*s (OTTs) como Google e Amazon para tornar a ferramenta global em ambientes de produção de grandes empresas.

Já a escolha do LXC é devido a ele ter sido considerado o primeiro sistema de contêineres que entregava uma solução completa. Além disso, até a versão 1.11 o Docker ainda utilizava ele em uma camada subjacente do seu código servindo como plataforma base para execução das imagens de contêineres.

3.4.1 Arquitetura do Docker

O Docker foi desenvolvido com técnicas de *Copy-on-Write* (CoW). A ideia básica é que um novo recurso, seja ele um bloco de disco ou uma área de memória, só é alocado quando for modificado. O Docker utiliza um esquema de camadas (*layers*) e para

montar essas camadas são usadas técnicas de CoW. Um contêiner é basicamente uma pilha composta por N camadas *read-only* e uma camada, a superior, conhecida como *read-write*, conforme é visto na Figura 3.1.

As camadas da Figura 3.1 representam uma aplicação web Java e elas são construídas através de uma ferramenta chamada *Dockerfile* que é simplesmente um arquivo de texto com uma sintaxe específica para criação de imagens Docker. A técnica de CoW faz com que caso o usuário precise fazer o download e configurar um Apache Tomcat dentro de um diretório `/opt/tomcat`, esse procedimento não afetará a imagem original do Java na camada abaixo. Ao invés disso, o Docker iniciará escrevendo uma camada de sistema de arquivos totalmente nova. Quando o contêiner iniciar, ele agrupará esses sistemas de arquivos juntos. Isso permite ler o diretório `/usr/bin/java` de uma camada e um `/opt/tomcat/bin` de outra. De fato, cada passo em um *Dockerfile* produz uma camada nova de um sistema de arquivos, mesmo se somente um arquivo é alterado. Por fim, essas imagens customizadas, construídas muitas vezes em função de outras imagens já existentes podem ser salvas e replicadas com facilidade.

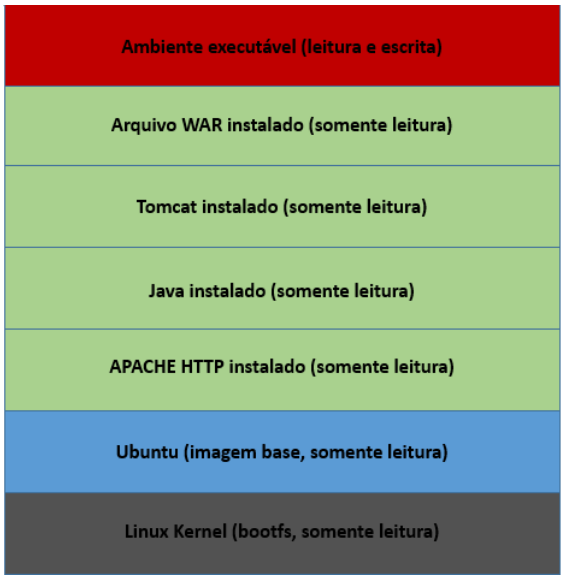


Figura 3.1: Exemplo da imagem Docker de uma aplicação em Java.

O Docker *daemon* escuta as requisições das APIs Docker e gerencia os objetos Docker tais como imagens, contêineres, redes e volumes. Um daemon também pode se comunicar com outros para gerenciar serviços Docker. Já o cliente Docker é a principal maneira dos usuários interagirem com a ferramenta Docker e com isso enviarem comandos para o daemon através de uma API Docker.

Além dos elementos acima, há o registro Docker, onde são armazenadas as imagens. Por padrão, o Docker utiliza o Docker Hub, mas também há a Docker Cloud. Além desses anteriores, há a Docker Store que possibilita a compra e venda de ima-

gens, uma vez que as imagens são customizáveis e geralmente baseadas em outras imagens.

A Docker Cloud fornece um serviço de registro hospedado com construção e teste de facilidade para imagens de aplicações Docker, ferramentas para auxiliar a construção de imagens, gerenciamento de infraestrutura e aplicações com funções de ciclo de vida dos contêineres para automatizar o desenvolvimento e atualização de serviços criados com imagens Docker, conforme é visto na Figura 3.2.

Conforme dito anteriormente, a Docker Store é a loja onde podem ser publicados códigos e procurar *plugins* ou imagens que podem agregar valor a solução de negócio de cada empresa ou para desenvolvedores que pensam em empreender. Além disso, há uma versão *open source* do Docker que é a *Community Edition* e outra paga para empresas que buscam suporte e contratos de SLA que é a *Enterprise Edition*. Com isso, o Docker criou um ecossistema para aproveitar ao máximo suas ferramentas e difundi-las para ambientes de produção de empresas a nível global.

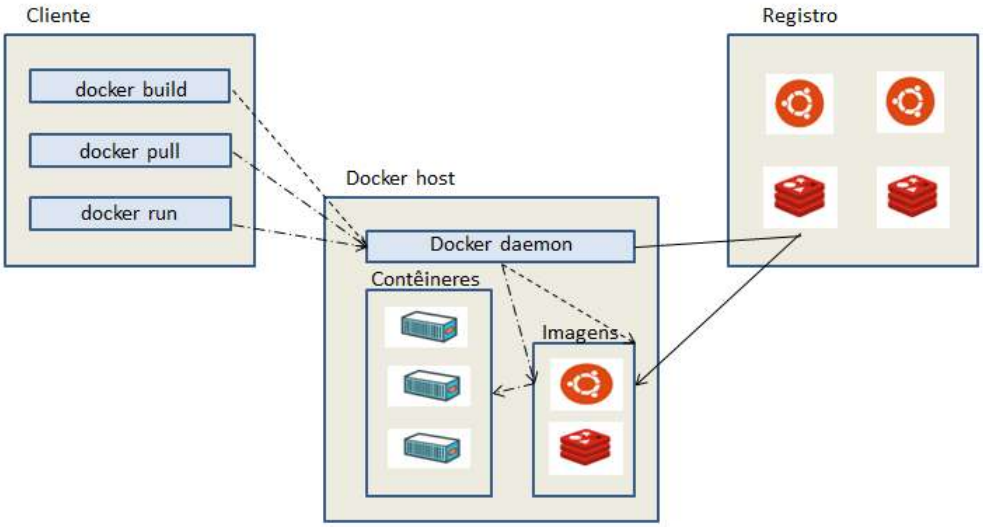


Figura 3.2: O ecossistema do Docker.

A eficiência das facilidades do Docker no armazenamento e distribuição de imagens dos contêineres é baseada em um *backend* de armazenamento flexível e permutável. Por padrão, o Docker utiliza o sistema de arquivo sobreposto *AnotherUnionFS* (AUFS). Este combina diversos diretórios em um sistema de arquivo virtual com múltiplas camadas chamadas *branches*. As camadas inferiores são somente de leitura, porém os arquivos dessas camadas podem ser logicamente trocados reescrevendo um arquivo de um *branch* superior, conforme pode ser observado na Figura 3.1 para uma aplicação web em Java.

Capítulo 4

Virtualização de Funções de Rede

O trabalho colaborativo em NFV nasceu em outubro de 2012, quando alguns provedores de serviços de telecomunicações ao redor do mundo publicaram um documento pedindo uma ação industrial e acadêmica. Em novembro de 2012, sete desses provedores selecionaram o *European Telecommunication Standards Institute* (ETSI) para ser o grupo de especificação industrial para NFV. Atualmente, mais de 5 anos depois, uma grande comunidade de *experts* está trabalhando intensamente para desenvolver os padrões necessários para NFV e compartilhando suas experiências no seu desenvolvimento e implantação [21].

4.1 Objetivo do NFV

A demanda dos usuários por novos serviços, muitas vezes de curta duração e com banda passante elevada, continua a aumentar. Por exemplo, frequentemente os clientes solicitam um aumento programado de banda por um tempo limitado ou o bloqueio de portas em um *firewall* para todas as filiais de uma empresa, para alguma manutenção rápida.

Com isso, as operadoras continuamente compram, guardam e operam novos equipamentos físicos, os quais possuem custos muito elevados. A principal ideia do NFV é desacoplar o equipamento de rede físico das funções que são executadas nele. Isso significa que uma função de rede tal como um *firewall*, pode ser vista pela operadora como uma instância de um software simples, adotando o conceito de NFV.

Dessa maneira, um dado serviço pode ser decomposto em um conjunto de VNFs, as quais podem então ser implementadas em um software executado em um ou mais servidores de prateleira (COTS). As VNFs podem então ser realocadas e instanciadas em diferentes localizações, acrescentando flexibilidade à configuração da rede da operadora [22].

4.2 Diferenças para soluções legadas

O desacoplamento entre o software e o hardware faz com que um elemento de rede não seja mais uma composição de entidades de hardware e software integradas, frequentemente chamadas de soluções verticais. Isso possibilita que as evoluções de ambos sejam independentes uma da outra e também permite a separação das linhas de desenvolvimento e manutenção [23].

Outra diferença é o desenvolvimento flexível de funções de rede, uma vez que a separação entre o software e o hardware ajuda a reatribuir e compartilhar os recursos de infraestrutura, facilitando a realização de diferentes funções em diferentes períodos. Isso ajuda as operadoras de rede a implantarem novos serviços de forma mais rápida utilizando a mesma plataforma física e com isso, componentes podem ser instanciados em qualquer dispositivo que suporta NFV na rede e suas conexões configuradas de maneira flexível.

Além disso, o NFV proporciona uma escalabilidade dinâmica, uma vez que a separação da funcionalidade das funções de rede em componentes de software instanciáveis fornece uma alta flexibilidade para escalar o atual desempenho das VNFs dinamicamente [24] e com granularidade mais fina, por exemplo, de acordo com o tráfego atual para o qual o operador precisa prover capacidade.

4.3 Edge Computing

A Edge Computing, ou computação na borda, é uma tecnologia emergente que fornece serviços de nuvem e de Tecnologia da Informação (TI) com uma maior proximidade dos assinantes, com isso a inteligência da rede é expandida para a borda aumentando a responsabilidade da operadora ao lidar com plataformas distribuídas e consequentemente com o aumento na complexidade para gerenciamento delas.

Uma plataforma de computação na borda reduz a latência das aplicações através de sua capacidade de computação e armazenamento na borda da rede, ou seja, mais próximo do usuário final [25]. Dispositivos móveis e de IoT são os grandes beneficiados desse novo conceito, uma vez que possuem uma grande descarga de tráfego oriundo de aplicações com uso computacional intensivo, tais como processamento de imagens, jogos móveis e etc.

A computação na borda não implica porém a substituição dos *datacenters* tradicionais ou da computação em nuvem que conhecemos. A computação na borda é uma computação adicional que pode coexistir com a computação na nuvem. Assim, uma carga computacional é distribuída onde é mais conveniente, entre a borda e a nuvem. Por exemplo, dados de sensores serão coletados e processados em *gateways* na borda onde soluções de tratamento de grandes massas de dados (*Big Data*) po-

dem ser aplicadas utilizando algoritmos baseados em regras e inclusive podendo ser aplicadas em tempo real.

Os dados filtrados podem ser enviados para uma nuvem centralizada visando o enriquecimento e agregação destes com outras fontes de informação, tais como banco de dados de *Customer Relationship Management* (CRM) e então colocados em mecanismos de análise da rede para gerar modelos que podem ser novamente enviados para análise na borda.

Em relação à telefonia móvel, a *Mobile Edge Computing* (MEC) oferece informação da RAN em tempo real para os desenvolvedores de aplicação e conteúdo [26]. Essas informações, tais como utilização da rede e localização do usuário na rede, são utilizadas para fornecer serviços personalizados para assinantes móveis, elevando o nível de flexibilidade dos serviços e consequentemente de satisfação do usuário e *Quality of Experience* (QoE). Além disso, as operadoras podem permitir que os rádios na borda da rede sejam manipulados por parceiros e com isso permitir o rápido desenvolvimento de novas aplicações e serviços na ponta para assinantes móveis e empresas.

Já no nicho de IoT, a *Fog Computing* é proposta para possibilitar a computação diretamente na borda da rede que pode entregar novas aplicações e serviços [27] especialmente para Internet do Futuro. Na *Fog Computing*, facilidades ou infraestruturas que podem fornecer recursos para serviços na borda da rede são chamados *fog nodes* [28]. Estes podem ser dispositivos com poucos recursos como *set-top-boxes*, pontos de acesso, roteadores, concentradores, *base stations* ou terminais. Um exemplo disso é a plataforma IOx [29] da Cisco que é executado em um hipervisor e permite que desenvolvedores executem códigos python podendo assim executar seus próprios códigos.

Apesar de terem nomes distintos em algumas áreas, o cerne dos conceitos acima é o mesmo, uma vez que a camada de Edge Computing será executada perto das fontes de dados e cada unidade de Edge Computing terá seu próprio conjunto de recursos no que diz respeito a computação, armazenamento e rede. Essas unidades serão configuradas para funções específicas as quais serão a função principal do dispositivo. Estes, por sua vez, serão os primeiros responsáveis pela manipulação de funções de rede como switching, routing, load balancing, segurança. O *cluster* de dispositivos de Edge computing é o ponto de agregação local para dados originados de uma variedade de fontes.

Ademais, cada um desses dados obtidos é analisado por um mecanismo complexo de processamento de eventos que decide o caminho pelo qual eles seguirão baseado em políticas e regras predefinidas, com isso os dados são processados localmente ou são encaminhados para a nuvem tradicional visando um processamento mais completo. O *hot data*, o qual é crítico para a operação da infraestrutura lo-

cal será analisado, armazenado e processado imediatamente pela camada de Edge Computing. O *cold data* que contribui para uma análise histórica a longo prazo será enviado para a nuvem tradicional para um processamento em lote [30].

4.4 Desempenho em NFV

O conceito de NFV consiste em executar *Network Function*s (NFs) em servidores COTS, o que significa que os fornecedores de servidores poderiam produzir o equipamento sem o conhecimento das características das funções que, por sua vez, poderiam ser executadas neles no futuro. Da mesma forma, os fornecedores de VNFs poderiam garantir que essas funções seriam capazes de serem executadas nos servidores anteriores. Isso levanta a questão de que se as funções executadas nos servidores COTS poderiam realmente alcançar um desempenho comparável àquelas que são executadas em hardware especializado e se essas funções poderiam ser migráveis entre esses servidores.

O desempenho em altas taxas de transmissão é um problema mesmo em equipamentos que executam funções de rede não virtualizadas. Já em ambientes virtualizados, técnicas tais como aceleração de hardware serão também importantes para NFV, uma vez que o desempenho é impactado por uma série de fatores como a virtualização em si, o uso de redes sobrepostas e o uso de protocolos para transporte como *Transmission Control Protocol* (TCP)/IP.

Alguns pesquisadores [31] [32] [33] em NFV buscam como alcançar alto desempenho em uma plataforma de software *middlebox* virtualizada. Para Edge Computing com NFV, o desempenho em ambientes virtualizados ainda é a primeira preocupação, uma vez que esse desafio possui dois aspectos: o primeiro é a vazão e latência das *middleboxes* virtualizadas na rede e o outro é como obter uma instalação eficiente, colocação e migração das instâncias virtuais em uma rede dinâmica cumprindo requisitos de latência baixa e vazão elevada.

Existem algumas abordagens para aceleração de software como as bibliotecas *Data Plane Development Kit* (DPDK) que utilizam *polling* para superar os gargalos das interrupções geradas pelo hipervisor e utiliza tecnologias como *Core pinning* que fixa os processadores que a VM irá utilizar. Já em relação a hardware existe o *Single Root I/O Virtualization* (SR-IOv) que permite que a *Network Interface Card* (NIC) crie diversas funções virtuais com acesso direto a VM realizando um *bypass* dos gargalos do sistema hospedeiro. Entretanto, o presente trabalho não considerou essas técnicas de aceleração nos cenários em análise.

Ademais, VNFs relacionadas a serviços de voz manipulam uma grande quantidade de pacotes com tamanhos pequenos, os quais exigem um grande esforço ao sistema, sendo assim, a garantia contra perda de pacotes é muito mais complexa

quando se tratam de pacotes pequenos. Além disso, alguns subsistemas são compostos por diversos componentes, como por exemplo *IP Multimedia Subsystem* (IMS), *Evolved Packet Core* (EPC) e *Virtual Customer Premises Equipment* (vCPE), o que contribui para penalizar o desempenho. Adicionalmente aos desafios decorrentes da virtualização, há o impacto devido ao *Network Service Chaining* (NSC) para permitir o encadeamento de funções em determinados fluxos de carga gerando assim a flexibilidade lógica e automação prometidas pelo NFV.

Capítulo 5

Trabalhos Relacionados

A avaliação de desempenho em soluções virtualizadas está presente em diversos trabalhos na literatura com alguns deles abordando aplicações específicas como em [34] que faz uma análise de desempenho de duas ferramentas de código aberto de virtualização, KVM e Docker, em um cenário com *Hypertext Transfer Protocol* (HTTP) *proxy*. Adicionalmente, em [35] foram apresentadas técnicas leves de benchmarkings na nuvem que são executadas rapidamente e podem ser utilizadas em tempo real.

Em [36] é realizada uma investigação sobre os impactos de vários hipervisores no que diz respeito ao desempenho do sistema e com o objetivo de identificar qual é o hipervisor ótimo para satisfazer as necessidades do mercado. Os hipervisores VMWare [37], VirtualBox e VirtualPC foram comparados adotando um benchmarking de desempenho que analisa I/O de disco, memória, CPU e consumo de energia. Entretanto, o trabalho considera como cenário nativo um sistema operacional diferente do cenário virtualizado podendo ter seus resultados mascarados devido à natureza das diferentes pilhas de sistemas operacionais hospedeiros. Já o presente trabalho leva em consideração o desempenho de rede, aplicação e utiliza o mesmo sistema operacional hospedeiro para todos os cenários avaliados trazendo mais confiabilidade às amostras.

Já em [38] é descrita uma abordagem fim a fim para avaliar plataformas de virtualização. Introduz-se um benchmarking com virtualização tradicional que envolve uma comparação de desempenho em sistemas reais versus desempenho em sistemas virtualizados. Foram descritos três benchmarkings específicos para ambientes virtualizados. Para nível de sistema foi utilizado o VMMark [39], o qual incorpora rotinas que avaliam simultaneamente sobrecarga de diversos recursos como CPU, memória e I/O. As outras duas ferramentas foram desenhadas para medição de gerenciamento de virtualização e desempenho da aplicação: VCBench e ViewPlanner [40]. Houve também uma preocupação com *design* de fluxos de trabalho, entretanto diferentemente do presente trabalho, o artigo focou em um único tipo de virtualização utilizando um único hipervisor para análise o que gera um resultado muito específico

podendo não ser aplicado em ambientes reais.

Ademais, em [41] foi feita uma análise de desempenho em 3 diferentes sistemas de arquivos no Linux que são `ext4`, `xfs` e `btrfs` utilizando hipervisores do tipo 2. Foram utilizadas as ferramentas de benckmarking Postmark [42] e Bonnie++ para geração de testes no ambiente com ênfase na vazão de leitura e escrita de disco sob diferentes condições de carga. Entretanto, a análise foi específica para disco que talvez não seja o aspecto mais importante em termos de desempenho de aplicações no ecossistema de NFV, uma vez que processamento e memória são cruciais e também são uma das contribuições do presente trabalho.

Diversas técnicas de benchmarking são tipicamente pesadas consumindo processos os quais necessitam testar a VM inteira a fim de obter dados corretos. Tais benchmarkings não podem ser feitos em tempo real na nuvem causando custos extras quando a aplicação é implementada. Em [35] , foram apresentadas técnicas leves de *benchmarking* na nuvem que são executadas rapidamente e podem ser utilizadas em tempo real. A utilização de técnicas de *benchmarking* leves são facilitadas pelo desenvolvimento do *Docker Container-bases technology Lightweight Benchmarking* (DocLite). Este é construído com tecnologia Docker, a qual permite que um pedaço definido pelo usuário seja testado. DocLite opera em dois modos, no primeiro, os contêineres são utilizados para avaliar uma pequena parte da VM para gerar classificações de desempenho. Já no segundo modo de execução, os dados históricos do *benchmarking* são utilizados através do primeiro modo de maneira híbrida para gerar classificações de VMs.

Nos trabalhos [43] e [44], visando avaliar como serão os cenários e protocolos para Internet do Futuro, tendo como premissa que virtualização será a técnica principal para acomodar os elementos dessa nova Internet, os autores avaliaram o desempenho de três ferramentas de virtualização conhecidas que são Xen, VMWare e OpenVZ considerando a utilização para virtualização de roteador. Os autores conduziram o experimento com ferramentas de benchmarking para medir a sobrecarga introduzida pela virtualização em termos de memória, processador, rede e desempenho de disco dos roteadores virtuais executando em servidores COTS.

Adicionalmente, avaliaram os efeitos de escalabilidade de máquinas virtuais na ferramenta de virtualização Xen e os resultados mostraram que ele é quem melhor atende os requisitos de roteadores virtuais. Entretanto, a análise está defasada devido a diversas melhorias nas ferramentas utilizadas diferentemente do presente trabalho. Além disso, o trabalho presente analisa explicitamente a virtualização assistida por hardware e a virtualização leve que é foco em ambientes operacionais nas principais empresas que lidam com aplicações na nuvem evidenciando um avanço no trabalho atual em relação aos citados acima.

O NFV é um novo paradigma no qual serviços de rede são virtualizados e podem

ser executados em hardware genérico com melhor agilidade do serviços e redução de custos. Há muitos fatores que encorajam o emprego de NFV, mas alguns requisitos particulares devem ser cuidadosamente avaliados, como por exemplo qual tipo de virtualização escolher. Em [34] foi feita uma análise de desempenho de duas soluções de código aberto (*open source*) de virtualização, KVM e o Docker, em um cenário de HTTP *proxy*. Os resultados mostraram que o Docker processa as requisições HTTP em um tempo menor que o KVM, devido a sua virtualização leve. Com isso, baseado nos resultados, forneceram uma ampla discussão sobre cada solução de virtualização aplicada em ambientes NFV.

Nenhum dos trabalhos anteriores mistura diversos tipos de virtualização e ferramentas atuais com análise de aplicações de missão crítica, ou seja, aquelas cruciais para a empresa como banco de dados, *Enterprise Resource Planning* (ERP), *Customer Relationship Management* (CRM) e plataformas de colaboração. Além disso, nenhum dos anteriores utiliza mais de uma ferramenta em uma única frente de análise, conforme é feito no caso de processamento para dupla validação do comportamento garantindo maior confiabilidade a análise. Assim como os outros trabalhos, o presente trabalho se preocupa com tratamento estatístico, uma vez que em todos os casos foram consideradas entre 50 e 100 iterações para cada ponto em análise.

Capítulo 6

Metodologia de Análise

Este capítulo descreve a metodologia utilizada para analisar o desempenho de diferentes tipos de virtualização na implementação de *Edge computing* baseada em NFV, ou seja, a computação próxima ao usuário final. A computação na borda é passível de ser aplicada em diferentes casos como IoT, SD-WAN, *Customer Premises Equipment*s (CPEs) virtuais, Redes Móveis, *Content Delivery Networks* (CDN), *Big Data*, entre outros.

Cabe ressaltar que quando é medido o desempenho de um dispositivo de rede virtual, avalia-se o desempenho cumulativo de várias entidades envolvidas, seja da VM, do contêiner, do hipervisor e dos recursos de NFVI. Qualquer comparação entre desempenho de alguma entidade de diferentes fornecedores deve ser feita somente quando as variáveis envolvidas se mantiverem constantes para um cenário justo. Por exemplo, quando o desempenho de VNFs fornecidas por diferentes *Independent Software Vendor*s (ISVs) é comparado, deve-se utilizar o mesmo hipervisor e os mesmos recursos de NFVI no momento de execução do *benchmarking* de desempenho.

Foram analisadas algumas das ferramentas de virtualização mais difundidas atualmente: KVM, VirtualBox [45], LXC e Docker. As ferramentas escolhidas representam diferentes tipos de virtualização, conforme explicado anteriormente. Uma das maiores dificuldades em avaliar ferramentas de virtualização é que algumas ferramentas comuns de *benchmarking* têm suas medidas distorcidas pelo tempo interno ao ambiente virtualizado, porém essa dificuldade não prejudicou a confiabilidade da análise devido ao ajuste de *timestamp* no KVM e ao controle de *drift* temporal no Virtualbox.

Conforme dito anteriormente, as ferramentas de virtualização de diferentes fornecedores devem ser avaliadas utilizando os mesmos benchmarkings, além das mesmas VNFs e recursos de NFVI a fim de manter a mesma referência na análise dos resultados obtidos. Entretanto, uma consideração importante é que não basta avaliar somente o número máximo de VMs suportadas, por exemplo. Em vez disso, deve ser testado o número máximo de VM suportadas no hipervisor sob tráfego avaliando

a QoE do usuário que o serviço é capaz de oferecer dentro de limites aceitáveis e satisfazendo o SLA contratado.

Foram realizados benchmarkings avaliando desempenho de CPU, de memória, de entrada e saída em disco, de desempenho de rede, e de aplicações visando a verificação da qualidade de experiência do usuário que é uma tendência nas operadoras de telecomunicações em face da real ameaça das chamadas OTTs.

Além disso, é válido informar que o presente trabalho não leva em consideração as técnicas que alguns hipervisores e sistemas operacionais utilizam relacionadas à otimização de CPU diminuindo o número de interrupções no controle dos elementos virtualizados através de técnicas como SR-IOv, DPDK e *Core pinning*. Essa escolha foi feita na tentativa de simplificar os cenários, uma vez que o objetivo do presente trabalho é mostrar a diferença de desempenho das tecnologias de virtualização conforme elas são disponibilizadas na instalação básica e padrão.

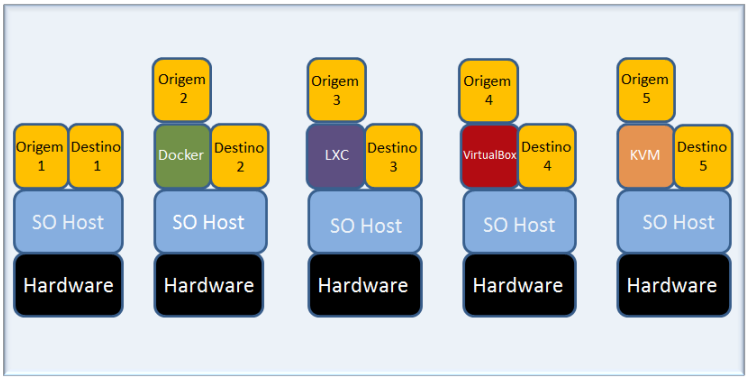


Figura 6.1: Cenários para avaliação do desempenho de rede.

Para realizar a análise de desempenho, foi utilizado um cenário composto por um computador pessoal com processador Core i7, 8 GB de *Random Access Memory* (RAM) e disco rígido de 1 TB. O computador executa um sistema operacional Linux, distribuição Ubuntu 16.4 com *kernel* versão 4.4.0-31 e placa de rede com velocidade de 100 Mbps.

Tabela 6.1: Cenários de análise.

Cenários	**Plataformas**
1º cenário	Nativo
2º cenário	Nativo + Docker
3º cenário	Nativo + LXC
4º cenário	VirtualBox + VM
5º cenário	KVM + VM

A configuração dos testes permitiu comparar o desempenho dos diferentes tipos de virtualização em cinco cenários distintos, ilustrados na Tabela 6.1. Ademais, a

topologia de rede está ilustrada na Figura 6.1, com origem e destino de acordo com o cenário definido na Tabela 6.1.

Os cenários e ferramentas escolhidas foram o Docker por ser a ferramenta mais utizada relacionada a virtualização leve, o LXC que deu origem ao Docker e é considerado a primeira tecnologia a oferecer um ambiente completo para execução de contêineres. Além dessas duas, há o Virtualbox que é uma ferramenta muito conhecida por representar os hipervisores do tipo II e o KVM que é *open source* e mundialmente utilizado em ambientes de produção representando os hipervisores tipo I com virtualização assistida por hardware. Ademais, há o cenário nativo que representa o legado e serve como referência para comparação dos gargalos inseridos pelas diferentes tecnologias de virtualização.

Além disso, em alguns cenários foram utilizadas mais de uma ferramenta de análise para verificar a correlação dos resultados e evidenciar um comportamento semelhante em ambas a título de comparação. Com isso, é possível realizar uma dupla validação do resultado e assim evitar que alguma ferramenta tenha uma conclusão equivocada nos resultados do benchmarking devido a alguma característica peculiar da sua implementação.

6.1 Processamento

Um programa de computador consiste em uma série de passos chamados instruções, as quais informam aos computadores o que fazer e o processador de um computador busca, decodifica e executa as instruções dos programas. Cada instrução pode ser um cálculo aritmético básico ou uma operação lógica e antes que o programa possa ser executado ela é lida dentro da memória.

Essa basicamente é a função do processador, o qual controla o computador para buscar uma instrução de programa na memória, decodificar a instrução e então executar as ações necessárias no processo de execução. É de responsabilidade do processador executar o ciclo busca-decodificação-execução inúmeras vezes com as instruções obtidas na memória. Essa ciclo de busca, decodificação e execução é frequentemente chamado de ciclo de execução.

Os principais componentes do processador são a Unidade Lógica e Aritmética (*Arithmetic and Logic Unit* (ALU)), a Unidade de Controle (*Control Unit* (CU)) e os registradores. A primeira é o cérebro do processador, realiza os cálculos aritméticos básicos como adição, substração, multiplicação e divisão e também as operações lógicas como comparação. Já a segunda, como o nome sugere é quem controla todas as funções executadas dentro do próprio processador. Também indica para a ALU qual operação lógica ou aritmética precisa ser feita, estas disparadas a cada *clock* do sistema e escolhendo todos os caminhos internos do processador para ter certeza de

que o dado seja obtido no lugar certo e vá para o destino correto.

Os registradores também são componentes chave dos processadores e eles ocupam a posição de um armazenador temporário dos dados vindos da memória RAM em direção ao processador para execução e dos dados vindos do processador após o processamento ser feito. Portanto, um registrador é um local de armazenamento dentro do processador que é utilizado para manter dados e instruções que estão sendo trabalhados pelo processador.

Sendo o processador peça essencial para um sistema de computação, sua utilização foi considerada como uma das frentes de análise nessa dissertação para determinar seu desempenho nos 5 cenários, a fim de identificar possíveis gargalos em cada um e como esses impactam a velocidade de processamento. As ferramentas utilizadas para essa análise foram o Y-cruncher [46] e o Sysbench [47], o primeiro calcula o valor de π com um número de casas decimais determinado pelo usuário e o segundo calcula os números primos existentes até um limite superior.

6.2 Disco

O disco rígido (*Hard Disk* (HD)) é o local onde se encontra a maior parte da memória secundária, onde o computador armazena os programas e as informações que estão sendo usadas, modificadas ou processadas. Uma característica desse tipo de disco é a persistência de dados, ou seja, estes dados gravados podem ser recuperados inúmeras vezes pela memória RAM, modificados e novamente gravados.

Em se tratando de disco, o principal gargalo é o tempo de acesso que é o período necessário para um computador processar o dado pedido pelo processador e então obter o dado pedido no dispositivo de armazenamento. Uma vez que os discos rígidos convencionais são mecânicos, é necessário esperar que o disco rotacione até determinado setor específico do disco. Sendo assim, a latência dessa leitura é da ordem de grandeza de milissegundos, enquanto a da memória RAM é de nanosegundos evidenciando uma grande diferença em desempenho.

Por outro lado, há também o *Solid-State Drive* (SSD) que é uma nova tecnologia de armazenamento considerada a evolução do HD. Ele não possui partes móveis e é construído em torno de um circuito integrado semicondutor, o qual é responsável pelo armazenamento, diferentemente dos sistemas magnéticos. Duas vantagens são a redução de vibrações devido a eliminação de partes mecânicas e o tempo de acesso reduzido à memória flash persistente presente no seu interior. Entretanto, os SSDs não foram utilizados no presente trabalho.

Gargalos no desempenho de disco podem afetar o desempenho de todo o sistema. Sendo assim, problemas relacionados a disco criam lentidão nas VMs, nos sistemas operacionais visitantes e nas aplicações executadas dentro das VMs. Problemas com

I/O de disco causam experiências como *timeouts* podendo chegar ao congelamento das VMs e até mesmo sua interrupção.

Devido ao disco ser um potencial gargalo a sistemas que utilizam virtualização, ele foi utilizado como frente de análise no presente trabalho. A ferramenta utilizada para avaliar o desempenho do disco foi o Bonnie++ [48], um software livre para teste de disco que simula algumas operações como criação, leitura, exclusão e buscas aleatórias de arquivos.

6.3 Memória

Nos computadores, a memória é a responsável pelo armazenamento de dados e instruções em forma de sinais digitais. Com isso, para que o processador possa executar suas tarefas, ele busca na memória RAM algumas as informações necessárias ao processamento.

Além da memória RAM que é do tipo *dynamic* RAM, existe uma memória mais rápida ainda que é a memória *cache*, do tipo *static* RAM. Esta é utilizada em pequenas quantidades posicionada entre a memória RAM e o CPU para diminuir o impacto de diferença de velocidade entre eles. Com isso, a memória cache copia os dados frequentemente acessados da RAM para ela e tenta prever o que o CPU solicitará na próxima requisição.

O tipo de memória abordado nesse trabalho é o RAM, uma vez que é para ela que os programas, ou parte deles, e os dados que estão sendo trabalhados no momento da execução do programa são transferidos. É principalmente nela em que são executadas a maioria das operações, portanto é nesta memória que ocorrem as operações de CPU.

A razão da existência e importância da memória RAM está na sua velocidade de leitura de dados que é muito grande. Todas informações que estão contidas nela podem ser acessadas de maneira mais rápida do que as informações que estão no disco rígido, o qual é considerado um tipo de memória secundária. Estas, apesar de terem acesso mais lento são permanentes, ou seja, as informações nelas gravadas ficam armazenadas mesmo quando o dispositivo está desligado.

A memória RAM é parte essencial de um sistema com virtualização, além disso a quantidade de memória disponível em uma plataforma impacta o número de VMs que podem ser executadas sem compartilhar memória evitando a degradação do desempenho. Devido a sua importância, a memória foi considerada uma das frentes de análise nessa dissertação para determinar seu desempenho nos cenários propostos, a fim de identificar possíveis gargalos. A ferramenta utilizada para essa análise foi o STREAM [49] que mede o desempenho da memória utilizando algumas operações triviais no *kernel*.

6.4 Rede

Uma rede de computadores é um conjunto de dois ou mais dispositivos, também chamados de nós, que utilizam um conjunto de regras em comum para compartilhar recursos entre si através de uma determinada conexão.

Rede virtual é um tipo de rede configurada para permitir que sub-redes sejam criadas de forma lógica, isto é, adicionar, mover e alterar um ambiente virtual são processos executados por software economizando assim o tempo e o custo consumidos nas reinstalações físicas.

Dois modos de abordagem comuns para análise de desempenho da rede são o ativo e o passivo. Esses dois são complementares entre si e podem ser utilizados em conjunto para uma medição eficaz. No escopo dessa dissertação utilizamos ambos os modos para a análise.

O modo ativo injeta tráfego de teste na rede ou envia pacotes para servidores e aplicações a fim de medir o desempenho obtido pela rede no momento da execução. A criação de tráfegos ajustáveis e medidas significativas podem ser obtidas com uma pequena quantidade de tráfego em certos casos. Além disso, o modo ativo fornece uma maior flexibilidade e controle da natureza da geração de tráfego, técnicas de amostragem, o tempo, frequência, tamanho e tipo de pacotes, qualidade estatística, caminho e função a ser monitorada.

Ademais, o modo ativo significa liberdade para testar o que quiser e do jeito que quiser. Com isso, facilita a reprodução de cenários reais e a verificação das garantias de *Quality of Service* (QoS) e os requisitos de SLA de maneira correta. Além disso, fornece um estado de rede fim a fim entre dois *hosts* evidenciando o desempenho da rede através da medição de *Round Trip Time* (RTT), média das perda de pacotes e banda disponível ao longo da rede.

A fim de avaliar a sobrecarga de virtualização no desempenho de rede, foi adotada a ferramenta Iperf [50] para medir a vazão com fluxos partindo da tecnologia sob teste com destino para o *host* considerando apenas o protocolo de transporte *User Datagram Protocol* (UDP) por ser amplamente utilizado e para simplificação de cenário, uma vez que o protocolo UDP é mais simples em termos de funcionalidades que outros protocolos como o TCP.

6.5 Aplicação

Desempenho de aplicação e qualidade de experiência do usuário são cruciais para o sucesso dos provedores de telecomunicações que visam concorrer com as OTTs na venda de conteúdo. Além disso, um dos objetivos da tecnologia da informação é tornar os usuários e consequentemente as empresas mais produtivos.

A possibilidade do usuário ser o árbitro final do valor do serviço justifica a importância do foco de TI na experiência do usuário. Usuários finais preocupam-se com utilidade dos serviços e com a sua facilidade de utilização, ou seja, se uma aplicação ou serviço não agrega valor ou não funciona conforme o esperado, os usuários não irão utilizá-la.

Não basta simplesmente analisar diversas frentes com variáveis isoladas para determinar um resultado real de *benchmarking* que avalia a QoE, uma vez que problemas podem acontecer com a necessidade de interação dessas variáveis em ambientes reais. Portanto, é evidente a importância de um teste com cargas reais de aplicações de uso intensivo como um banco de dados.

Tendo em vista a necessidade de ter um desempenho das aplicações previsível, o presente trabalho escolheu aplicação como uma frente de análise no *benchmarking* proposto através da ferramenta sysbench [47] que é uma ferramenta muito utilizada, criada há 9 anos e em constante atualização para analisar o desempenho de uma aplicação de banco de dados relacional conhecida como MySQL [51], a qual é mundialmente utilizada [52].

Capítulo 7

Resultados Experimentais

Ferramentas de *benchmarking* sintético permitem gerar diferentes tipos de carga para avaliar o desempenho de algum subsistema específico do hardware. Assim, CPU, memória, I/O de disco, rede e qualidade da experiência do usuário são os principais componentes testados nesse trabalho, uma vez que eles são métricas importantes para ambientes de utilização do NFV. Para isso, as ferramentas utilizadas na avaliação foram o Y cruncher [46], o Sysbench [47], o STREAM [49], o Bonnie++ [48] e o Iperf [50].

7.1 Desempenho de CPU

A fim de avaliar a sobrecarga da virtualização em termos do desempenho de CPU, foi utilizado o Y-cruncher que é uma ferramenta de *benchmarking* com múltiplas *threads* desenvolvida para sistemas *multicore* que calcula o valor de π. Há outras aplicações para *stress test* de CPU que realizam o cálculo de π, porém o Y-cruncher tem a vantagem de ser *multi-thread*. Uma outra ferramenta *multi-thread* é o wPrime [53], porém este só funciona para o sistema operacional Windows e por isso não foi utilizada no presente trabalho.

A métrica levada em consideração na ferramenta é o tempo total de cálculo de π. Por mais que exista também o tempo total de execução que consiste no tempo total de computação somado ao tempo que é necessário para o processamento do resultado, este não reflete uma diferença de desempenho somente de CPU e sim de velocidade de escrita no disco já que uma das saídas desse software é a quantidade escolhida de algarismos de π escrita em um arquivo.

O tamanho do valor de π considerado na ferramenta Y-cruncher foi 500 milhões de algarismos. Conforme observa-se na Figura 7.1, o desempenho da virtualização leve foi próxima à do ambiente nativo, entretanto o desempenho da virtualização baseada em software foi aquém das demais evidenciando um gargalo de CPU na utilização do VirtualBox.

Os desempenhos obtidos tanto pelo Docker quanto pelo LXC aproximam-se do ambiente nativo devido a ambos compartilharem recursos do *kernel* hospedeiro utilizando o *Completely Fair Scheduler* (CFS) como escalonador de processos. Este, por sua vez, sempre busca normalizar as fatias de tempo para acesso pelo processador em relação ao número total de tarefas para se aproximar do acesso multitarefa ideal do processador, ou seja, executar os processos de maneira justa.

Ademais, os já mencionados cgroups e *namespaces* se encarregam do isolamento dos contêineres proporcionando a cada um deles um escalonamento de processos em grupo através do CFS que utiliza um mecanismo de árvore rubro-negra ordenado no tempo para construir uma linha de execução de tarefas futuras.

Apesar de também utilizar o CFS como escalonador de processos, o KVM obteve um desempenho inferior aos cenários com virtualização leve devido ao tratamento dado às interrupções pela sua arquitetura que depende da virtualização baseada em hardware. Esta ainda possui uma sobrecarga nas trocas de contexto entre visitante e hospedeiro fazendo com que algumas instruções causem um número elevado ciclos de processamento.

Além do Y-cruncher, também foi utilizada a ferramenta Sysbench que realiza o cálculo dos N números primos existentes até um determinado valor superior como critério de parada. O valor escolhido foi 200.000 porque é o valor máximo sugerido pela ferramenta dada uma análise interna de hardware. Como pode ser visto na Figura 7.2, o desempenho do cenário que utiliza o Virtualbox foi bem inferior aos demais devido à tentativa do sistema operacional visitante de executar código de anel 0 em anel 1 causando uma série de falhas de instrução, uma vez que o código no anel 1 não possui permissão de executar instruções privilegiadas.

Em cada uma dessas falhas, o Virtualbox intervém através da emulação do código para obter o comportamento desejado e isso gera um sobrecarga elevada que degrada o desempenho nitidamente. Apesar de utilizar técnicas para melhoria de desempenho como o componente chamado *Code Scanning and Analysis Manager* (CSAM) que desmembra o código executado pelo visitante e aciona o *Patch Manager* (PATM) que substitui esse código em tempo de execução, o desempenho do Virtualbox permaneceu abaixo dos demais.

O motivo da utilização de mais de uma ferramenta nesse cenário de processamento é evidenciar a correlação dos resultados em ambas as ferramentas. Com isso, foi confirmado que o comportamento observado é semelhante em ambas com um desempenho perto do nativo pelas ferramentas com virtualização leve com um desempenho inferior pelas ferramentas que utilizam virtualização completa.

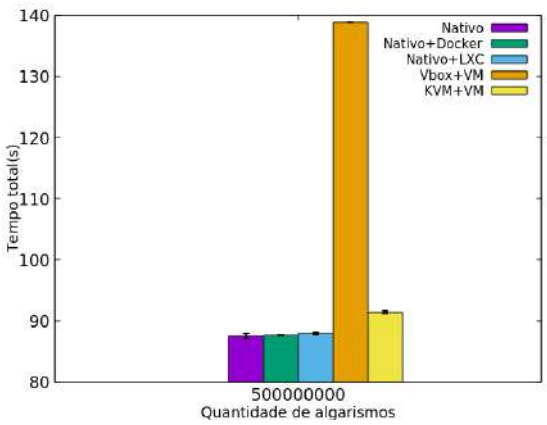


Figura 7.1: Desempenho de CPU: tempo de cálculo do número π com x algarismos utilizando o Y-cruncher.

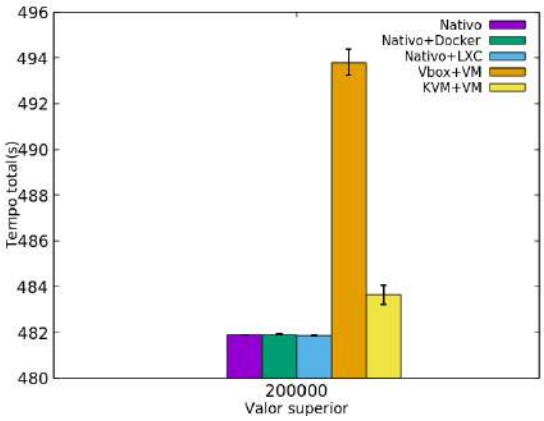


Figura 7.2: Desempenho de CPU: tempo de cálculo de N números primos até o valor superior x utilizando o Sysbench.

7.2 Desempenho de Memória

A fim de testar a velocidade de acesso à memória foi utilizada a ferramenta STREAM [49]. Esta mede o desempenho da memória utilizando operações triviais de vetores no *kernel*, ou seja, gerando resultados para quatro operações diferentes sobre vetores que são a Cópia, Escala, Adição e Tríade que podem ser vistas na Tabela 7.1. Conforme ilustrado na Figura 7.3, para as quatro operações, o desempenho do VirtualBox que representa a virtualização completa baseada em software é inferior aos demais sendo aproximadamente 30% inferior a partir de 2.000 *threads*.

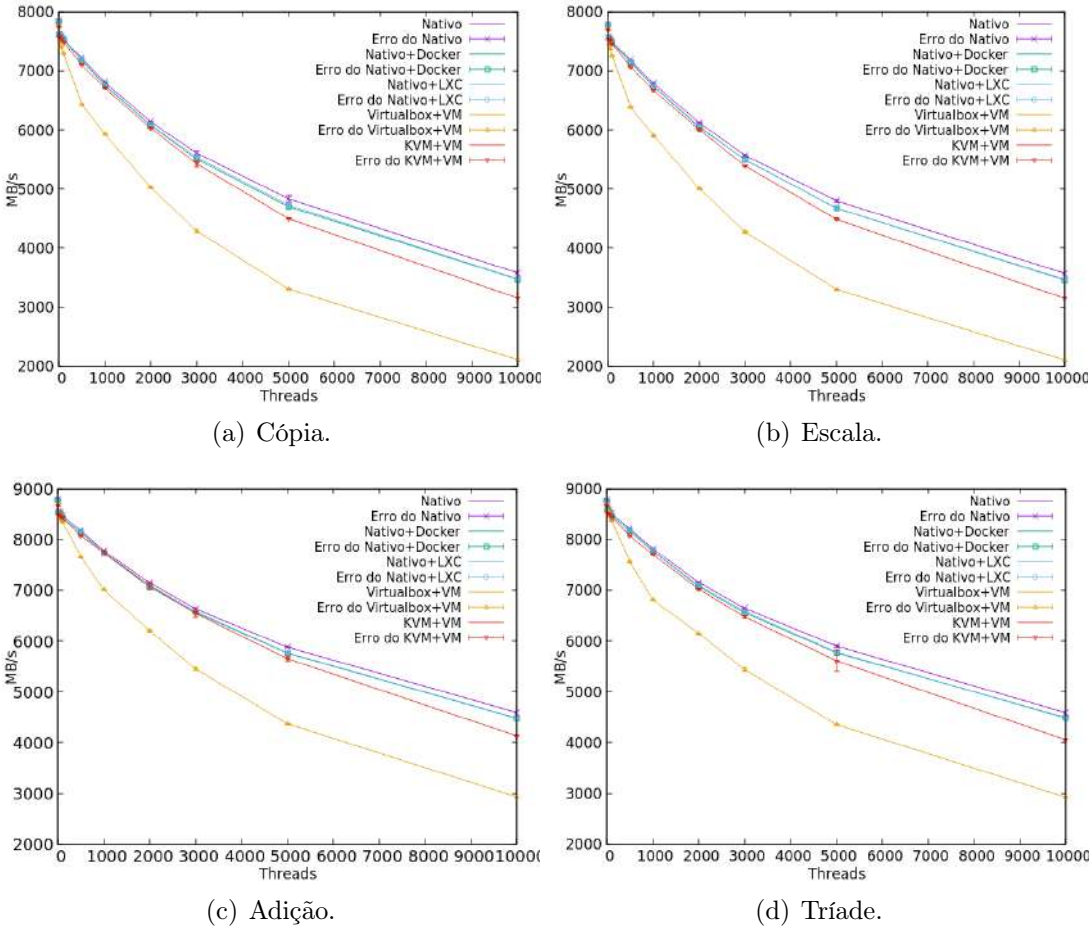
Tabela 7.1: Tabela de operações STREAM.

Operação	**Fórmula**	**Bytes por iteração**
Cópia	x[i] = y[i]	16
Escala	x[i] = q * y[i]	16
Adição	x[i] = y[i] + z[i]	24
Tríade	x[i] = y[i] + q * z[i]	24

A tradução de endereços de memória virtuais para físicos pelos hipervisores é uma operação que utiliza a memória intensamente, uma vez que o mapeamento deve ser feito de maneira hierárquica diversas vezes. Para reduzir essa sobrecarga, os processadores automaticamente guardam as traduções recentes em tabelas, chamadas *Translation Look-aside Buffer* (TLB). Com isso, a cada referência a memória, o processador checa antes o TLB para determinar se a tradução desejada já está em *cache*.

Ademais, o sistema operacional atualiza o *Control Register 3* (CR3) durante uma mudança de contexto. Uma mudança no registrador CR3 estabelece um novo conjunto de traduções e então o processador invalida as entradas do TLB associadas ao contexto anterior.

O VirtualBox obteve um desempenho inferior aos demais devido à sua técnica de tradução de memória chamada *shadow pages*, a qual mantém uma tabela de blocos de memória de sombra derivada da tabela de memória do sistema operacional visitante. Quando o visitante está ativo, o hipervisor força o processador a utilizar essa tabela de sombra para realizar a tradução de endereços de memória e isso gera uma sobrecarga devido ao esforço de manter essa tabela de sombra válida através de rastreamento do estado da tabela de memória do visitante.



(a) Cópia. (b) Escala.

(c) Adição. (d) Tríade.

Figura 7.3: Desempenho de Memória: Vazão de operações em memória utilizando diferentes operações com a ferramenta STREAM.

Os cenários com virtualização leve utilizam o cgroups de memória, o `memcg`, para alocar endereços físicos de memória, inclusive com seu próprio mapeamento de endereços mais utilizados recentemente conhecido como *Least Recently Used* (LRU) gerando uma sobrecarga baixa e fazendo com que seu desempenho seja semelhante ao nativo.

Além disso, no cenário com KVM é construída uma tabela de endereços de memória para mapear os endereços físicos do visitante para os endereços físicos do hospedeiro como se o visitante acessasse seu próprio endereço físico. Essa técnica, conhecida pela Intel como *Extended Page Table* (EPT), faz com que o controle do

mapeamento de memória fique no sistema operacional visitante fazendo com que o hipervisor não fique sempre rastreando o estado da tabela de memória do visitante para emular alguma alteração, reduzindo assim a sobrecarga como pode ser visto na Figura 7.3.

7.3 Desempenho de Disco

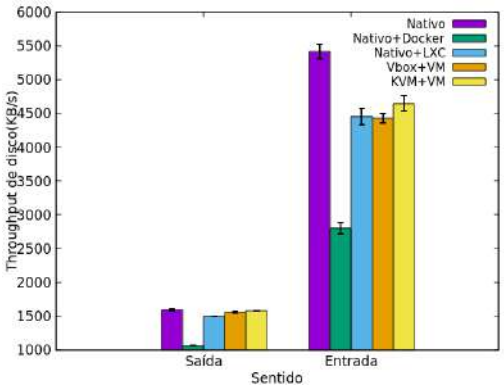
Nesse trabalho, também foi avaliado o desempenho de acesso ao disco rígido. Para isso foi medida a sobrecarga de virtualização do disco nas tarefas de escrita e leitura comparando a sobrecarga produzida por cada tecnologia de virtualização. Foi utilizada a ferramenta Bonnie++, um software livre de teste de disco que simula algumas operações como criação, leitura e exclusão de pequenos arquivos. O Bonnie++ também testa o desempenho de acesso a diferentes regiões do disco rígido através do acesso aos setores no começo, meio e fim do disco.

As métricas levadas em consideração para a análise foram a velocidade de escrita em disco representada como entrada e de leitura representada como saída do disco. Além disso, foi avaliada a quantidade de buscas aleatórias feitas com sucesso no disco por meio da criação de quatro processos em paralelo pela ferramenta Bonnie++ e variando a quantidade de arquivos com os valores 25, 50 e 100.

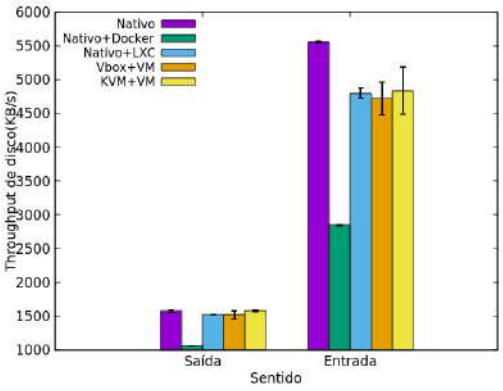
Como pode ser observado na Figura 7.4, tanto para 25, 50 e 100 arquivos, o pior desempenho é o da ferramenta Docker, a qual utiliza virtualização leve frente às demais e a discrepância é grande até mesmo com o LXC que utiliza o mesmo tipo de virtualização. O Docker versão 1.12.6 executado no Ubuntu 16.04 utiliza o sistema de arquivos AUFS em conjunto com o ext4 como sistema de arquivos subjacente.

O AUFS agrupa diversas camadas em um único hospedeiro e as apresenta como sendo uma única. Essas camadas são unificadas através de um processo que é chamado de *union mount*. Uma vez que o AUFS faz uma manipulação a nível de arquivos, basta uma pequena escrita em um arquivo que ele tem que ser copiado por inteiro. Essa operação é chamada de *copy up* e gera uma grande sobrecarga na escrita e leitura de arquivos conforme é observado na Figura 7.4.

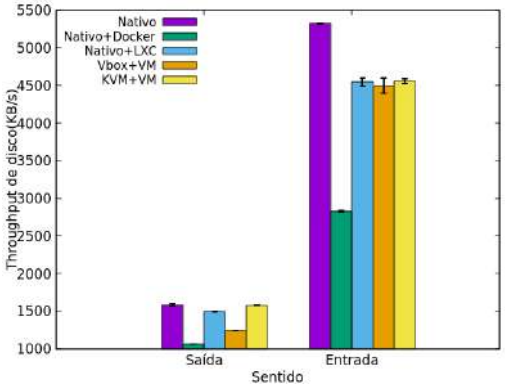
Tanto o desempenho do Virtualbox quanto do KVM foram próximos ao desempenho do cenário nativo. O primeiro emulou um controlador *Serial ATA* (SATA) no modo *Advanced Host Controller Interface* (AHCI) juntamente com uma imagem de disco com extensão *Virtual Disk Image* (VDI), já o segundo obteve um bom resultado devido à baixa sobrecarga inserida pelo KVM que manipula arquivos através de entidades chamadas *storage pools* e *volumes*. O *storage pool* padrão utilizado foi o baseado em diretórios com uma técnica de *cache* chamada *writeback* na qual estão habilitados para o sistema operacional visitante tanto o *cache* do hospedeiro para melhorar I/O de disco quanto o *cache* de operações de escrita no disco.



(a) 25 arquivos.



(b) 50 arquivos.



(c) 100 arquivos.

Figura 7.4: Desempenho de Disco: Vazão de acesso ao disco para diferentes quantidades de arquivos criados utilizando o Bonnie++.

Para buscas aleatórias no disco feitas com sucesso após o Bonnie++ criar 4 processos filhos, o desempenho do Docker foi bom como é mostrado na Figura 7.5, devido as buscas não necessitarem de operações de cópia em diversas camadas, uma vez que basta achar a posição no disco para medir o *Input Output per Second* (IOPS).

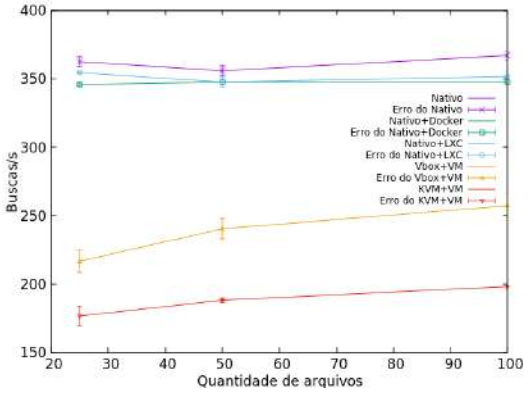


Figura 7.5: Desempenho de Disco: Taxa de buscas aleatórias em disco feitas com sucesso utilizando o Bonnie++ e 4 processos em paralelo.

Por outro lado, tanto o desempenho do cenário com KVM quanto o do cenário com Virtualbox foram ruins devido a gargalos inseridos pelos sistemas de arquivos e controladores de I/O, no KVM através do `virtio-blk` e no VirtualBox com o SATA. Esses gargalos das diferentes técnicas de virtualização no que diz respeito a tempo de acesso a disco impactam diretamente a quantidades de buscas aleatórias no disco.

7.4 Desempenho de Rede

Uma função virtual de rede, como por exemplo um roteador, tem que ser capaz de receber corretamente pacotes e encaminhá-los para a interface de saída. Diante disso, a camada de virtualização deve possuir obviamente a menor sobrecarga possível para não afetar o desempenho de rede. A fim de avaliar a sobrecarga de virtualização no desempenho de rede, foi adotada a ferramenta Iperf [50] para medir a vazão de fluxos partindo da tecnologia sob teste com destino ao *host*, conforme observado na Figura 6.1.

Nesta frente de análise foi realizada uma transferência de um arquivo com tamanho aproximado de 6GB durante 10 segundos com uma banda nominal declarada na ferramenta Iperf de 1Gbps e com tamanho de pacote padrão da ferramenta que é de 1470 bytes.

Tabela 7.2: Desempenho de rede: Vazão total utilizando o Iperf.

Plataforma	**Fluxo UDP (Mbps)**
Nativo	$815 \pm 0,15$
Nativo+Docker	$814 \pm 0,08$
Nativo+LXC	814 ± 0.2
VirtualBox+VM	$511 \pm 31,59$
KVM+VM	$672 \pm 33,63$

A Tabela 7.2 mostra que o pior desempenho para o protocolo de transporte UDP foi o do VirtualBox. O desempenho do KVM para UDP ficou bem próximo tanto do ambiente nativo quanto das ferramentas que utilizam virtualização leve, que mais uma vez tiveram um bom desempenho.

Apesar do Docker e LXC utilizarem uma *bridge* com o uso de *Network Address Translation* (NAT), eles compartilham o mesmo *kernel* do sistema hospedeiro e acessam os recursos computacionais sem a necessidade de percorrer pilhas adicionais e assim ambos alcançaram um desempenho semelhante ao cenário nativo.

O desempenho do Virtualbox foi inferior aos demais devido à sobrecarga inserida pela emulação da placa que foi a Intel PRO/1000, a qual habilita funções como a *checksum offloading* e *segmentation offloading* que contribuíram para a degradação

do desempenho. Além disso, há o esforço para utilização do NAT na interface degrada o desempenho do cenário como podemos observar na Tabela 7.2.

Por outro lado, o KVM obteve um desempenho melhor que o cenário com Virtualbox devido à utilização de paravirtualização para o dispositivo de rede através da interface *virtio-net* que apesar de diminuir as trocas de contexto entre hospedeiro e visitante, ainda possui sobrecarga elevada em comparação tanto ao ambiente nativo e ao cenário com virtualização leve.

7.5 Desempenho de Aplicação

Diversas operadoras de serviços de telecomunicações estão preocupadas com conteúdo para atrair clientes e assim a qualidade de experiência do usuário está em foco, conforme mencionado anteriormente. Como a motivação principal deste trabalho é avaliar a tecnologia com melhor desempenho para inserção em um ambiente com virtualização de funções de rede aliado ao conceito de *Edge computing*, foi avaliado o desempenho de uma aplicação em cada tipo de virtualização.

A aplicação escolhida para análise foi um banco de dados MySQL, uma vez que esta é uma aplicação muito utilizada em diversas áreas tecnológicas. Para avaliar o desempenho das plataformas foi criado em cada uma delas um banco de dados com uma tabela contendo 1 milhão de registros e o *benchmarking* consiste em realizar consultas a valores contidos nessa tabela.

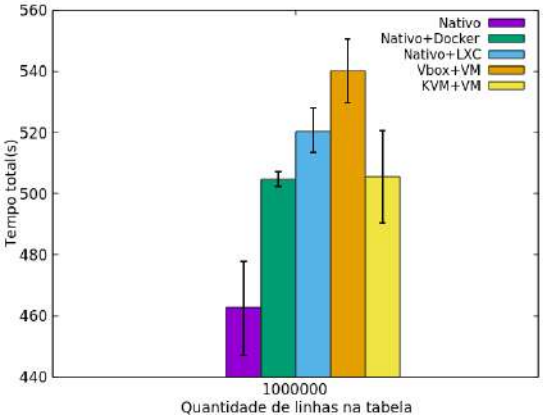


Figura 7.6: Desempenho de aplicação: Tempo de execução para consultas em uma tabela MySQL com 1 milhão de linhas utilizando o Sysbench.

A métrica considerada para avaliação é o tempo total de execução das operações citadas. Conforme se observa na Figura 7.6, o desempenho do VirtualBox foi pior, evidenciando uma sobrecarga criada pela virtualização tradicional na execução de aplicações. Esse fraco desempenho poderia degradar uma aplicação em produção,

o que torna a virtualização baseada em software uma escolha ruim para aplicações que exijam um alto desempenho de I/O.

O desempenho ruim tanto no cenário com LXC quanto o do cenário com Virtualbox evidenciam os gargalos inseridos pelas diferentes técnicas de virtualização no que diz respeito a tempo de acesso a disco, interação com *drivers*, velocidade de CPU e de acesso a memória em ambientes que vão além de *benchmarkings* sintéticos, ou seja, com aplicações reais.

Capítulo 8

Conclusão e Trabalhos Futuros

Soluções baseadas em contêineres e outros sistemas emergentes desafiam os tradicionais hipervisores, baseados em máquinas virtuais, na computação na nuvem. Essas novas tecnologias são mais leves e isso facilita um desenvolvimento mais denso de serviços. Nesse trabalho, foi apresentada uma avaliação de desempenho para diferentes tecnologias de virtualização no sistema operacional Linux.

Além do nível de sobrecarga baixo, elevado grau de versatilidade e facilidade de gerenciamento das ferramentas de virtualização serem pontos a favor, ficou evidente através do *benchmarking* das ferramentas que o desempenho das ferramentas de virtualização leve se aproxima da plataforma nativa na maioria dos casos. Este aspecto é importante para um ambiente de *Edge computing*, no qual a arquitetura de microsserviços tende a ganhar cada vez mais força e exigir um alto desempenho e flexibilidade das plataformas subjacentes.

As contribuições do trabalho são a demonstração de uma comparação atualizada entre um cenário nativo, outros com utilização de contêineres e outros com máquinas virtuais em ambientes utilizando virtualização através de *benchmarkings* relevantes para essa nova tendência em telecomunicações. Adicionalmente, a identificação do potencial da utilização de virtualização leve como alternativa à virtualização tradicional e seu impacto no desempenho de aplicações.

A análise do presente trabalho é revelante uma vez que com o novo objetivo de flexibilização das redes para geração de novos serviços de forma mais rápida e com a difusão de arquiteturas de microsserviços em ambiente virtualizados, há a possibilidade das operadoras de telecomunicações mudarem o cenário atual e modificarem totalmente a arquitetura das redes desde o acesso, transporte e agregação até o core. Além disso, há a oportunidade de alterarem características das suas redes, como por exemplo o *vendor lock-in* que muitas vezes gera uma estagnação tecnológica.

Os principais resultados foram um desempenho superior da virtualização leve em relação a CPU semelhante ao nativo, com uma diferença de quase 50 segundos para o Virtualbox que representa os hipervisor tipo II com virtualização completa quando

utilizada a ferramenta Y-cruncher. Além disso, foi confirmado com a ferramenta Sysbench um desempenho semelhante chegando a uma diferença de aproximadamente 102 segundos no cálculo dos números primos até o valor 200000 entre os mesmos representantes citados acima.

Em relação a memória, a virtualização leve alcançou um desempenho semelhante ao nativo para as 4 operações executadas na ferramenta STREAM, estas são Cópia, Escala, Soma e Tríade demonstrando ser eficaz em uma das principais fontes de gargalo em máquinas virtuais e alcançando diferenças que chegaram a 40% em relação a virtualização completa e a 10% em relação a virtualização assistida por hardware representada pela ferramenta KVM.

No que diz respeito a disco o desempenho o desempenho da virtualização leve foi inferior as demais para a ferramenta Docker e satisfatório para a ferramenta LXC em relação as demais, porém em todos os casos não se aproximou do ambiente nativo com uma diferença em torno de 20% evidenciando que em relação a disco a virtualização ainda necessita evoluir.

Para gargalos internos à plataforma relacionados a rede e gerados somente pelas camadas de virtualização com a ferramenta Iperf, somente a virtualização leve obtive um desempenho aproximadamente igual ao do ambiente nativo. Além disso, nessa frente de análise tanto a virtualização completa representando pelo Virtualbox quanto a virtualização assistida por hardware obtiveram um desempenho muito aquém do esperado sendo 40% inferior aos demais.

Na frente de análise de aplicação foi utilizado a ferramenta Sysbench em conjunto com o banco de dados MySQL que visava um teste com aplicações reais a fim de uma verificação mais completa da QoE nas plataformas, a virtualização leve representada pela ferramenta Docker obteve um desempenho superior que as demais, porém mais uma vez muito distante do cenário nativo. A virtualização tradicional representada pelo VirtualBox mais uma vez obteve de longe o pior desempenho. Isso demonstra que as técnicas desenvolvidas pela Intel relacionadas a virtualização assistida por hardware são eficazes para cenários reais, tendo em vista que o KVM foi superior a virtualização completa.

O objetivo da avaliação foi alcançado ao proporcionar um direcionamento em uma possível implantação de plataformas virtuais, isto é, no momento em que houver um questionamento sobre qual tecnologia de virtualização será utilizada, tendo em vista a sensibilidade da aplicação no que diz respeito a desempenho. Esse *trade-off* entre desempenho, flexibilidade e custo tem que ser analisado com mais profundidade em um trabalho futuro.

8.1 Trabalhos Futuros

Apesar de considerar diversas frentes de análise, o presente trabalho limitou-se a avaliar as plataformas na configuração *default* em que são apresentadas. Cabe como análise futura uma abordagem na qual verifica diversas variáveis para cada frente de análise proposta. Além disso, como o presente trabalho não levou em consideração técnicas de aceleração de hardware como SR-IOv, DPDK e *Core Pinning*, essa análise complementar é totalmente válida para os cenários reais.

Um estudo futuro interessante é uma análise de segurança [54] comparando contêineres com VMs para desmistificar o compromisso necessário entre segurança e flexibilidade na escolha da tecnologia a ser utilizada. Ademais, em posse das evidências que a virtualização leve possui um melhor desempenho que a virtualização tradicional, cabe avaliar as ferramentas de orquestração para contêineres e seus principais benefícios. Por fim, cabe a verificação do resultado quando ambos são colocados lado a lado em um ambiente real realizando *Network Service Chaining* (NSC), conforme é feito pela ferramenta Contrail [55] da Juniper.

Referências Bibliográficas

[1] "Quagga Source Open Router". Disponível em: <"http://www.nongnu.org/quagga/">. Acessado em 06/02/2018.

[2] "strongSwan IPsec Gateway". . Disponível em: <"https://www.strongswan.org">. Acessado em 06/02/2018.

[3] "runc/libcontainer at master opencontainers/runc". . Disponível em: <"https://github.com/opencontainers/runc/tree/master/libcontainer">. Acessado em 06/02/2018.

[4] "Xen Project". . Disponível em: <"https://www.xenproject.org/">. Acessado em 06/02/2018.

[5] MORABITO, R. "Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation", *IEEE Access*, v. 5, pp. 8835–8850, 2017. ISSN: 2169-3536. doi: 10.1109/ACCESS.2017.2704444.

[6] "Intel Clear Containers". Disponível em: <"https://clearlinux.org/containers">. Acessado em 06/02/2018.

[7] "Kata Containers - The speed of containers, the security of VMs". Disponível em: <"https://katacontainers.io/">. Acessado em 06/12/2018.

[8] "Kubernetes | Production-grade Container Orchestration". Disponível em: <"https://kubernetes.io/">. Acessado em 06/02/2018.

[9] "runV - bring isolation to Docker". Disponível em: <"https://blog.hyper.sh/runv-bring-isolation-to-docker.html">. Acessado em 06/02/2018.

[10] "Xenserver | Open Source Server Virtualization". . Disponível em: <"https://xenserver.org/">. Acessado em 06/02/2018.

[11] "MirageOS". Disponível em: <"https://mirage.io/">. Acessado em 06/02/2018.

[12] “xenserver/xscontainer: Support for Docker and Container Management”. Disponível em: <"https://github.com/xenserver/xscontainer">. Acessado em 06/02/2018.

[13] “KVM”. Disponível em: <"https://www.linux-kvm.org/page/Main_Page">. Acessado em 06/02/2017.

[14] “QEMU”. Disponível em: <"https://www.qemu.org/">. Acessado em 06/02/2018.

[15] “Virtio Libvirt”. Disponível em: <"https://wiki.libvirt.org/page/Virtio">. Acessado em 06/02/2018.

[16] “OpenVZ Virtuozzo Containers”. Disponível em: <"https://openvz.org/Main_Page">. Acessado em 06/02/2018.

[17] “Docker - Build, Ship, and Run Any App, Anywhere”. Disponível em: <"https://www.docker.com">. Acessado em 06/02/2018.

[18] “rkt, a security-minded, standards-based container engine”. Disponível em: <"https://coreos.com/rkt/">. Acessado em 06/02/2018.

[19] “Kurma - Containers all the way down”. Disponível em: <"https://github.com/apcera/kurma">. Acessado em 06/02/2018.

[20] “FreeBSD Jail/ZFS based implementation of the Application Container Specification”. Disponível em: <"https://github.com/3ofcoins/jetpack">. Acessado em 06/02/2018.

[21] LYNCH, P., HAUGH, M., KURTZ, L., et al. *Demystifying NFV in Carrier Networks: A Definitive Guide to Successful Migrations*. 1 ed. Califórnia, IXIA, 2014. Disponível em: <https://www.ixiacom.com>.

[22] LI, Y., CHEN, M. “Software-defined network function virtualization: A survey”, *IEEE Access*, v. 3, pp. 2542–2553, 2015.

[23] MIJUMBI, R., SERRAT, J., GORRICHO, J.-L., et al. “Network Function Virtualization: State-of-the-Art and Research Challenges”, *IEEE Communications Surveys & Tutorials*, v. 18, n. 1, pp. 236–262, 2016. ISSN: 1553-877X. doi: 10.1109/COMST.2015.2477041. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7243304>.

[24] SAHHAF, S., TAVERNIER, W., CZENTYE, J., et al. "Scalable architecture for service function chain orchestration". In: *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*, pp. 19–24. IEEE, 2015.

[25] DOLUI, K., DATTA, S. K. "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing". In: *Global Internet of Things Summit (GIoTS), 2017*, pp. 1–6. IEEE, 2017.

[26] JAIN, A., SADAGOPAN, N., LOHANI, S. K., et al. "A Comparison of SDN and NFV for Re-designing the LTE Packet Core". In: *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*, pp. 74–80. IEEE, 2016.

[27] VAQUERO, L. M., RODERO-MERINO, L. "Finding your way in the fog: Towards a comprehensive definition of fog computing", *ACM SIGCOMM Computer Communication Review*, v. 44, n. 5, pp. 27–32, 2014.

[28] BONOMI, F., MILITO, R., ZHU, J., et al. "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16. ACM, 2012.

[29] "Cisco IOx Network Infrastructure". Disponível em: <"https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html">. Acessado em 06/02/2018.

[30] LEVANDOSKI, J. J., LARSON, P.-Å., STOICA, R. "Identifying hot and cold data in main-memory databases". In: *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pp. 26–37. IEEE, 2013.

[31] BONAFIGLIA, R., CERRATO, I., CIACCIA, F., et al. "Assessing the performance of virtualization technologies for nfv: a preliminary benchmarking". In: *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*, pp. 67–72. IEEE, 2015.

[32] RAHO, M., SPYRIDAKIS, A., PAOLINO, M., et al. "Kvm, xen and docker: A performance analysis for arm based nfv and cloud computing". In: *Information, Electronic and Electrical Engineering (AIEEE), 2015 IEEE 3rd Workshop on Advances in*, pp. 1–8. IEEE, 2015.

[33] HWANG, J., RAMAKRISHNAN, K. K., WOOD, T. "NetVM: high performance and flexible networking using virtualization on commodity platforms", *IEEE Transactions on Network and Service Management*, v. 12, n. 1, pp. 34–47, 2015.

[34] EIRAS, R. S., COUTO, R. S., RUBINSTEIN, M. G. "Performance evaluation of a virtualized HTTP proxy in KVM and Docker". In: *Network of the Future (NOF), 2016 7th International Conference on the*, pp. 1–5. IEEE, 2016. Disponível em: <http://ieeexplore.ieee.org/abstract/document/7810144/>.

[35] VARGHESE, B., SUBBA, L. T., THAI, L., et al. "Container-Based Cloud Virtual Machine Benchmarking". pp. 192–201. IEEE, abr. 2016. ISBN: 978-1-5090-1961-8. doi: 10.1109/IC2E.2016.28. Disponível em: <http://ieeexplore.ieee.org/document/7484184/>.

[36] AL JABRY, H., LIU, L., ZHU, Y., et al. "A critical evaluation of the performance of virtualization technologies". In: *Communications and Networking in China (CHINACOM), 2014 9th International Conference on*, pp. 606–611. IEEE, 2014. Disponível em: <http://ieeexplore.ieee.org/abstract/document/7054367/>.

[37] "Cisco IOx Network Infrastructure". Disponível em: <"https://www.vmware.com/company/foundation.html">. Acessado em 06/02/2018.

[38] SOUNDARARAJAN, V., AGRAWAL, B., HERNDON, B., et al. "Benchmarking a virtualization platform". In: *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pp. 99–109. IEEE, 2014. Disponível em: <http://ieeexplore.ieee.org/abstract/document/6983049/>.

[39] "VMmark Virtualization Benchmark". Disponível em: <"https://www.vmware.com/products/vmmark.html">. Acessado em 06/02/2018.

[40] "VMWare View Planner". Disponível em: <"https://www.vmware.com/br/products/view-planner.html">. Acessado em 06/02/2018.

[41] PESIC, D., DJORDJEVIC, B., TIMCENKO, V. "Competition of virtualized ext4, xfs and btrfs filesystems under type-2 hypervisor". In: *Telecommunications Forum, 2016 24th*, pp. 1–4. IEEE, 2016. Disponível em: <http://ieeexplore.ieee.org/abstract/document/7818895/>.

[42] "Postmark: A New File System Benchmark". Disponível em: <"https://private-communities.netapp.com/">. Acessado em 06/02/2018.

[43] MATTOS, D. M., FERRAZ, L. H. G., COSTA, L., et al. "Virtual network performance evaluation for future internet architectures", *Journal of Emerging Technologies in Web Intelligence*, v. 4, n. 4, pp. 304–314,

2012. Disponível em: <http://www.jetwi.us/uploadfile/2014/1222/20141222101024429.pdf>.

[44] MATTOS, D. M., FERRAZ, L. H. G., COSTA, L. H. M., et al. "Evaluating virtual router performance for a pluralist future internet". In: *Proceedings of the 3rd International Conference on Information and Communication Systems*, p. 4. ACM, 2012. Disponível em: <http://dl.acm.org/citation.cfm?id=2222448>.

[45] "Oracle VM Virtualbox". Disponível em: <"https://www.virtualbox.org/">. Acessado em 06/02/2018.

[46] "Y-cruncher - A Multi-Threaded Pi-Program". Disponível em: <"http://www.numberworld.org/y-cruncher/">. Acessado em 06/02/2018.

[47] "Sysbench". Disponível em: <"https://github.com/akopytov/sysbench/'>. Acessado em 06/02/2018.

[48] "Bonnie++". Disponível em: <"http://www.coker.com.au/bonnie++/">. Acessado em 06/02/2018.

[49] "STREAM: Sustainable Memory Bandwidth in High Performance Computers". . Disponível em: <"https://www.cs.virginia.edu/stream//">. Acessado em 06/02/2018.

[50] "Iperf2". Disponível em: <"https://sourceforge.net/projects/iperf2/">. Acessado em 06/02/2018.

[51] "MySQL". Disponível em: <"https://www.mysql.com/">. Acessado em 06/02/2018.

[52] "DB-Engine Rankings - popularity ranking of database management systems". Disponível em: <"https://db-engines.com/en/ranking">. Acessado em 06/02/2018.

[53] "Multi-threaded Computer Benchmark | wPrime". Disponível em: <"http://www.wprime.net/">. Acessado em 06/02/2018.

[54] ZHANG, M., MARINO, D., EFSTATHOPOULOS, P. "Harbormaster: Policy Enforcement for Containers". pp. 355–362. IEEE, 2015. ISBN: 978-1-4673-9560-1. doi: 10.1109/CloudCom.2015.96. Disponível em: <http://ieeexplore.ieee.org/document/7396177/>.

[55] "Contrail - Juniper Networks". . Disponível em: <"https://www.juniper.net/us/en/products-services/sdn/contrail/">. Acessado em 06/02/2018.

[56] NIU, Z., XU, H., TIAN, Y., et al. "Benchmarking NFV Software Dataplanes", *arXiv preprint arXiv:1605.05843*, 2016. Disponível em: <http://arxiv.org/abs/1605.05843>.

[57] LIVI, S., JACQUEMART, Q., PACHECO, D. L., et al. "Container-Based Service Chaining: A Performance Perspective". pp. 176–181. IEEE, out. 2016. ISBN: 978-1-5090-5093-2. doi: 10.1109/CloudNet.2016.51. Disponível em: <http://ieeexplore.ieee.org/document/7776597/>.

[58] ISMAIL, B. I., GOORTANI, E. M., AB KARIM, M. B., et al. "Evaluation of docker as edge computing platform". In: *Open Systems (ICOS), 2015 IEEE Confernece on*, pp. 130–135. IEEE, 2015. Disponível em: <http://ieeexplore.ieee.org/abstract/document/7377291/>.

[59] PREETH, E. N., MULERICKAL, F. J. P., PAUL, B., et al. "Evaluation of Docker containers based on hardware utilization". In: *Control Communication & Computing India (ICCC), 2015 International Conference on*, pp. 697–700. IEEE, 2015. Disponível em: <http://ieeexplore.ieee.org/abstract/document/7432984/>.

[60] JOY, A. M. "Performance comparison between linux containers and virtual machines". In: *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in*, pp. 342–346. IEEE, 2015. Disponível em: <http://ieeexplore.ieee.org/abstract/document/7164727/>.

[61] COSTA, L. H. M. K., TRINDADE, L. V. P. "Análise do Desempenho da Virtualização Leve para Ambientes com Edge Computing baseada em NFV", *XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2018*, maio 2018. Aceito em congresso.