UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

VIRTUALIZAÇÃO DE LABORATÓRIOS EDUCACIONAIS COM DISTRIBUIÇÃO EFICIENTE DE IMAGENS DE DISCO

MARCOS LAERTE GOMES DE LIMA

ORIENTADOR: PROF. DR. PAULO MATIAS

São Carlos – SP

Julho de 2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

VIRTUALIZAÇÃO DE LABORATÓRIOS EDUCACIONAIS COM DISTRIBUIÇÃO EFICIENTE DE IMAGENS DE DISCO

MARCOS LAERTE GOMES DE LIMA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas Distribuídos e Redes de Computadores.
Orientador: Prof. Dr. Paulo Matias

São Carlos – SP

Julho de 2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato Marcos Laerte Gomes de Lima, realizada em 01/06/2021.

Comissão Julgadora:

Prof. Dr. Paulo Matias (UFSCar)

Prof. Dr. Helio Crestana Guardia (UFSCar)

Prof. Dr. Paulo Licio de Geus (UNICAMP)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Dedico este trabalho a todo aquele
que não teve oportunidade de estudar.

AGRADECIMENTOS

Primeiramente agradeço a Deus por me conceder mais uma oportunidade de aprender e por todas as outras bênçãos que eu não sou capaz de enumerar.

Agradeço ao meu orientador Prof. Dr. Paulo Matias e ao meu coorientador Pedro Henrique Lara Campos por toda orientação, dedicação e paciência oferecidas a mim neste projeto; e principalmente, agradeço a amizade de vocês.

Aos meus pais Leonardo e Maria do Carmo, aos meus irmãos José Leandro e Carlos Laércio e à minha tia Severina, por terem me apoiado e me incentivado enquanto eu cursava a graduação em período integral.

Ao Erick Melo por incentivar a realização deste projeto, junto com todos os membros da Secretaria Geral de Informática, e em especial Mario Trench, Maria do Carmo e Mesailde Matias.

Agradeço a todos os membros do laboratório Asgard pelos momentos de descontração e ao Emerson Barea pelas orientações voluntárias sobre o mestrado. Sou grato aos professores do Departamento de Computação pelos conhecimentos transmitidos, em especial ao Prof. Dr. Hélio Crestana Guardia pelas sugestões dadas neste projeto e pelas aulas na graduação.

Por fim, agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e à UFSCar pela concessão de afastamento parcial para a realização de mestrado.

"De tudo o que se tem ouvido, a suma é:
Teme a Deus e guarda os seus mandamentos;
porque isto é o dever de todo homem."
(Eclesiastes 12:13)

RESUMO

Devido à transversalidade da Computação como área de pesquisa interdisciplinar e à adoção do computador como ferramenta pelas mais diversas áreas do conhecimento, laboratórios informatizados tornaram-se infraestrutura essencial para a realização de aulas de diversas disciplinas em instituições de ensino. Contudo, assim como qualquer organização ou empresa que possua dezenas ou centenas de máquinas, essas instituições têm enormes problemas para realizar o gerenciamento do parque computacional no que diz respeito à manutenção, atualização e inventário desses sistemas. O presente projeto pretende contribuir para a resolução desses problemas, criando uma solução para laboratórios educacionais informatizados que engloba autenticação de usuários, virtualização de sistemas de maneira transparente para o usuário (por meio do *passthrough* de dispositivos para a máquina virtual), gerência remota dos sistemas e distribuição de imagens de disco utilizando BitTorrent (de forma a aproveitar toda a largura de banda disponibilizada pela infraestrutura de rede). A solução proposta foi instalada nas salas de aula informatizadas mantidas pela Secretaria Geral de Informática da UFSCar. Os objetivos propostos de implantação de autenticação de usuários, virtualização de sistemas, gerência remota e distribuição de imagens foram alcançados. Espera-se que esta solução também seja adotada por outras universidades, e que o conhecimento gerado por este trabalho possa ser aplicado ao gerenciamento de grandes parques de máquinas em outros contextos e outras áreas da indústria.

Palavras-chave: Autenticação. Virtualização. Passagem de dispositivos. BitTorrent.

ABSTRACT

Due to the transversality of Computer Science and Engineering as an area of interdisciplinary research and to the adoption of the computer as a tool by the most diverse areas of knowledge, computer labs have become essential infrastructure for conducting various classes in educational institutions. However, just like any organization or company that has dozens or hundreds of machines, these institutions have huge problems managing their computer assets concerning maintaining, upgrading, and inventorying these systems. This project aims to contribute to solving these problems by creating a solution for computer labs that encompasses user authentication, transparent system virtualization (by passing through devices to the virtual machine), remote system management and disk image distribution using BitTorrent (to fully take advantage of the bandwidth provided by the network infrastructure). The proposed solution was installed in the computerized classrooms maintained by the Secretaria Geral de Informática at UFSCar. The proposed objectives of implementing user authentication, systems virtualization, remote management and image distribution have been achieved. It is expected that this solution will also be adopted by other universities, and that the knowledge generated by this work can be applied to the management of large machinery parks in other contexts and other areas of the industry.

Keywords: Authentication. Virtualization. Device passthrough. BitTorrent.

LISTA DE SIGLAS

ALM	*Application Layer Multicast*
AMD-V	*Advanced Micro Device – Virtualization*
API	*Application Programming Interface*
ATLB	*Arrival Time and Location-Based*
BIOS	*Basic Input/Output System*
CAT	*Configuration Assistant Tool*
CCTK	*Client Configuration Toolkit*
COW	*Copy On Write*
CPU	*Central Processing Unit*
Cron	*Command Run On*
D-Bus	*Desktop Bus*
DRM	*Direct Rendering Manager*
Ext4	*Fourth Extended Filesystem*
EAP	*Extensible Authentication Protocol*
Eduroam	*Education Roaming*
EGLFS	*Embedded-system Graphics Library – Full Screen*
GPT	*GUID Partition Table*
GPU	*Graphics Processing Unit*
GTK	*GIMP Toolkit*
GUID	*Globally Unique Identifier*
GRUB	*Grand Unified Bootloader*

HAXM	*Hardware Accelerated Execution Manager*
HD	*Hard Disk*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
ICP	Infraestrutura de Chaves Públicas
IME	*Intel Management Engine*
INI	*Initialization File*
Intel VT	*Intel Virtualization Technology*
Intel VT-d	*Intel Virtualization Technology for Directed I/O*
I/O	*Input/Output*
IOMMU	*Input/Output Memory Management Unit*
IP	*Internet Protocol*
IRC	*Internet Relay Chat*
ISP	*Internet Service Provider*
KVM	*Kernel-based Virtual Machine*
LAN	*Local Area Network*
LKML	*Linux Kernel Mailing List*
MAC	*Media Access Control*
MBR	*Master Boot Record*
MPMC	*Multipath-Multicast*
MSR	*Model-Specific Register*
OOM	*Out Of Memory*
PAM	*Pluggable Authentication Modules*
PCI	*Peripheral Component Interconnect*
P2P	*Peer-to-peer*
RADIUS	*Remote Authentication Dial-In User Service*

RAM	*Random Access Memory*
SDDM	*Simple Desktop Display Manager*
SDL	*Simple DirectMedia Layer*
SDVM	*Simple Desktop Virtualization Manager*
SIn	Secretaria Geral de Informática
SSD	*Solid State Drive*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*
TDP	*Thermal Design Power*
TTLS	*Tunneled Transport Layer Security*
UFSCar	Universidade Federal de São Carlos
USB	*Universal Serial Bus*
UUID	*Universally Unique Identifier*
VBR	*Volume Boot Record*
vBIOS	*Video BIOS*
VFIO	*Virtual Function Input/Output*
Virtio	*Virtual Input/Output*
VM	*Virtual Machine*
WinDbg	*Windows Debugger*
ZFS	*Zettabyte File System*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1
INTRODUÇÃO

1.1 Contexto

Laboratórios informatizados são essenciais em instituições de ensino para a realização de atividades em diversas áreas do conhecimento. Constantemente são criados *softwares* que servem como base para experimentos científicos e simulações, desempenhando um papel fundamental como instrumentos de ensino para professores e pesquisadores dessas instituições.

Apesar dos inúmeros benefícios dos laboratórios, é difícil gerenciar o parque computacional necessário para a realização dessas atividades. Grandes instituições e até mesmo empresas com dezenas ou centenas de máquinas têm enormes problemas para realizar atualizações e manutenções nos sistemas, muitas vezes agravados por infraestrutura inadequada, restrições devido ao horário de funcionamento do ambiente, custo para a instituição, dentre outros fatores.

Uma alternativa que poderia ser adotada seria migrar os laboratórios para uma nuvem computacional, como abordado em (ARORA et al., 2014). Nesse modelo, os autores apresentam um modelo de virtualização de laboratórios utilizando Openstack (SEFRAOUI et al., 2012), no qual máquinas virtuais são provisionadas na nuvem e réplicas são feitas para os alunos, permitindo que eles consigam acessá-las mesmo fora da universidade. Entretanto, nem todas as universidades têm condições financeiras para manter esses ambientes e, ainda mais, para uma grande quantidade de alunos.

Apesar disso, ter os computadores em um laboratório não é o suficiente. É necessário gerenciá-los com eficiência para atender às demandas como: instalação de *software*, atualização de sistema operacional, criação de área para os usuários, possíveis manutenções de *hardware* da máquina ou da rede local e outras tarefas semelhantes.

Todas essas intervenções mencionadas podem ser necessárias a qualquer momento e nem sempre são efetuadas com agilidade ou preventivamente. Muitas delas ocorrem após surgir um evento inesperado que, posteriormente, é reportado para a equipe técnica e em um terceiro momento, o problema é solucionado com a visita do técnico no laboratório. Ou seja, com um monitoramento em tempo real dos computadores, muitos desses problemas poderiam ser notados

e solucionados antes mesmo de um usuário formalizar uma reclamação.

Além da administração dos laboratórios, a gerência dos usuários que fazem uso dos computadores é essencial para se manter a integridade, privacidade e segurança dos arquivos dos usuários. Em muitos casos, as áreas de computadores públicos são compartilhadas com todos e não é feito um registro de quem usou o computador e nem o período em que ele foi utilizado por cada pessoa, pois o sistema foi configurado com um usuário padrão ou sem nenhuma forma autenticação.

Esse modelo de configuração pode ocasionar uma série de problemas como: acesso indevido de arquivos alheios, acesso a contas de *email* e perfis de redes sociais, no caso de algum usuário esquecer de encerrar as sessões desses ambientes, intervenções prejudiciais ao sistema por um usuário mal intencionado, entre outras. Um modo de contornar essas questões seria utilizar um sistema para cada usuário, e isso é possível por meio criação de áreas de usuário dentro do sistema operacional ou mesmo de virtualização de ambientes por completo, dando a cada usuário um sistema somente seu.

O ideal para um laboratório seria autenticar os usuários e separar as áreas destes; monitorar os computadores remotamente; manter um parque computacional homogêneo; realizar intervenções remotamente e de maneira atômica; manter os sistemas sempre atualizados, com as mesmas configurações e com a mesma versão; prover mais de um sistema operacional; realizar manutenções preventivas para evitar o impedimento de uso dos computadores e agilizar o processo de restauração de máquinas; automatizar a instalação de sistemas para futuras ampliações do parque computacional.

1.2 Objetivos

O objetivo geral deste projeto é criar uma solução que permita a gestão remota dos laboratórios informatizados de uma instituição, promovendo agilidade nas intervenções e reduzindo ao máximo a necessidade de deslocamento físico de técnicos até as salas de aula.

Os objetivos específicos são:

a) Autenticar os usuários simultaneamente no sistema das máquinas e na rede.

b) Virtualizar os principais sistemas operacionais utilizados em laboratórios informatizados. A fim de prover facilidade de uso e desempenho similar ao de um sistema não virtualizado, fazer a passagem de periféricos (vídeo, áudio e USB) do sistema hospedeiro para a máquina virtual. A fim de prover privacidade e flexibilidade no controle do ambiente, expor um disco virtual distinto para cada usuário.

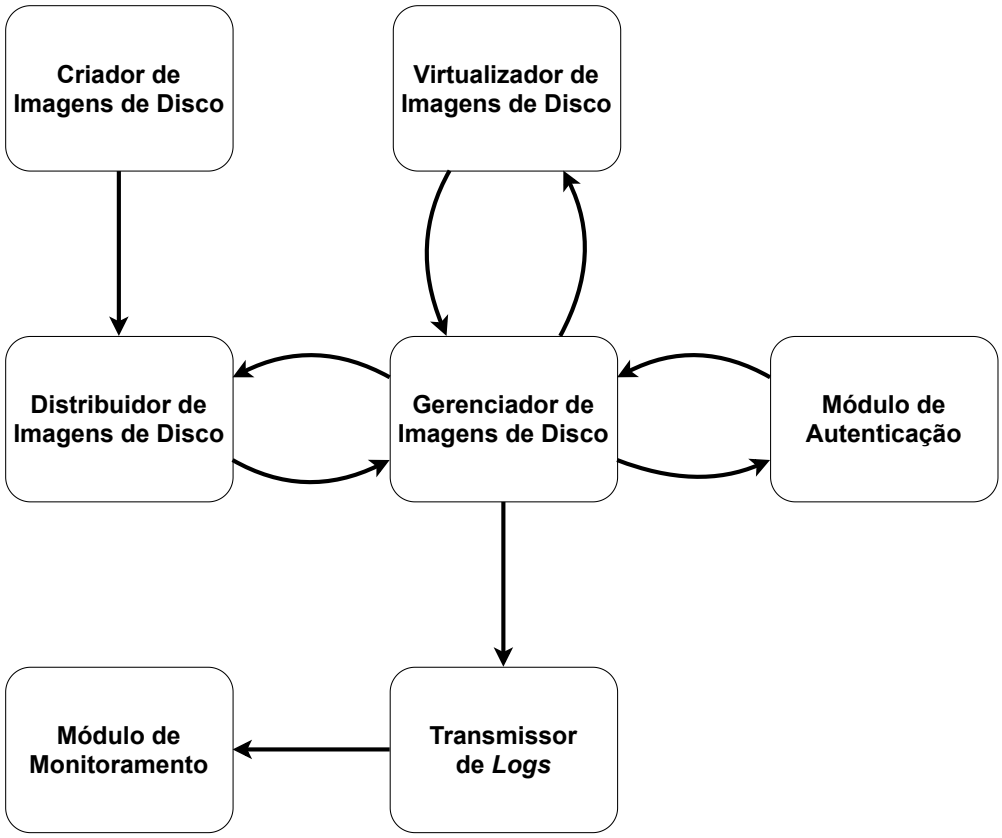
c) Disponibilizar meios para gerenciar remotamente as máquinas, permitindo atualizar o sistema hospedeiro de forma atômica, coletar informações de inventário e monitorar indicadores de qualidade de funcionamento das máquinas.

d) Descentralizar a distribuição de imagens de disco para o parque de máquinas de forma a aproveitar toda a largura de banda fornecida pela infraestrutura de rede local, mesmo que se trate de uma rede heterogênea.

1.3 Síntese dos métodos

Com intuito de atingir os objetivos citados, um modelo para gestão de laboratórios é proposto neste trabalho e exibido na Figura 1.

Figura 1 – Modelo para gestão de laboratórios informatizados



Fonte: Produzido pelo autor

Esse modelo foi criado com o propósito de alcançar uma gestão de laboratórios eficiente e contém os seguintes elementos: *(i)* Criador de Imagens de Disco, *(ii)* Distribuidor de Imagens de Disco, *(iii)* Gerenciador de Imagens de Disco, *(iv)* Virtualizador de Imagens de Disco, *(v)* Transmissor de *Logs*, *(vi)* Módulo de Monitoramento e o *(vii)* Módulo de Autenticação.

(i) **Criador de Imagens de Disco:** é o módulo responsável por criar e armazenar as imagens de disco que serão replicadas em todos os laboratórios. É imprescindível que ele crie imagens compatíveis com o *hardware* dos computadores existentes.

Outra tarefa que o Criador de Imagens deve desempenhar é instalar todos os programas necessários para atender às demandas da instituição. Após a finalização dessa atividade, o Criador de Imagens deve fornecer ao Distribuidor de Imagens de Disco a imagem pronta.

O processo de criação de imagem pode se tornar uma tarefa sujeita a falhas se realizada manualmente e, principalmente, à medida que mais instalações e configurações precisam ser efetuadas. Por isso, ter um criador de imagens automatizado é o mais indicado.

Com um criador de imagens automatizado, garante-se que nenhuma etapa será esquecida, facilita-se a identificação e reprodução de erros caso ocorram, produz-se uma imagem limpa e sem dados em *cache* que uma criação manual poderia gerar. Além disso, permite-se criar vários outros tipos de imagens somente modificando partes de um *script* automatizado, por exemplo.

(ii) **Distribuidor de Imagens de Disco:** é o módulo encarregado de transmitir as imagens para todos os computadores. Ele deve garantir que as imagens sejam transmitidas em sua totalidade e sem corrupção de dados.

É necessário que o Distribuidor seja eficiente, realizando as transferências à taxas de velocidade tão altas quanto à infraestrutura da rede permitir, pois imagens de disco costumam ser arquivos grandes e a demora na transferência pode congestionar a rede. Também é importante que ele execute de forma mais descentralizada possível, diminuindo a sobrecarga do servidor que contém as imagens que serão transmitidas.

O Distribuidor deve fazer a substituição das imagens de disco antigas de cada computador e informar ao Gerenciador de Imagens de Disco da existência das novas. Nesse processo, as imagens de disco dos usuários também são removidas. Desta forma, eles terão acesso às novas imagens na próxima vez que utilizarem um dos computadores.

(iii) **Gerenciador de Imagens de Disco:** faz o controle das imagens de disco principais transmitidas pelo Distribuidor, como também das imagens de disco de cada usuário que apontam para as imagens principais. O Gerenciador recebe as imagens do Distribuidor e oferece aos usuários como opções disponíveis para utilização, como por exemplo, sistemas operacionais distintos.

Além de gerenciar as imagens, após o usuário se autenticar no sistema, o Gerenciador informa ao Virtualizador a imagem associada ao usuário. Caso ainda não exista, o Gerenciador deve criá-la antes de iniciar a virtualização.

Por fim, o Gerenciador de Imagens de Disco registra em arquivos de *logs* todos os dados e eventos do sistema como um todo enquanto ele está em funcionamento. Esses registros são coletados pelo Transmissor de *Logs* para serem enviados ao Módulo de Monitoramento.

(iv) **Virtualizador de Imagens de Disco:** tecnologia capaz de virtualizar sistemas operacionais gravados em imagens de disco. O Virtualizador recebe a imagem de usuário que deve ser instanciada através do Gerenciador de Imagens de Disco após o usuário se autenticar.

O Virtualizador deve ser capaz de permitir a utilização do sistema operacional instanciado, efetuando leituras e gravações na imagem do usuário. Todas as alterações que o usuário fizer devem ser gravadas naquela imagem específica, e mantê-la isolada das demais.

A imagem precisa ser compatível com o *hardware* do computador disponível no laboratório. O Virtualizador deve ser customizável, concedendo possibilidade de ser configurado para aprimorar o desempenho da virtualização de cada tipo de sistema.

(v) **Transmissor de *Logs*:** é responsável por coletar registros de *logs* gerados desde o momento de inicialização do sistema até o seu encerramento. Também é responsável por coletar informações de *hardware* e *software* que podem variar no decorrer do tempo. Esses registros são coletados periodicamente e enviados ao Módulo de Monitoramento.

Diversos dados são gerados durante a utilização dos computadores, por isso, é importante selecionar as informações mais relevantes de cada *log*. Essas informações devem fornecer aos administradores o real estado de cada computador. Com informações precisas, como por exemplo, do que está sendo executado nas máquinas, quais componentes estão funcionando, quais dispositivos estão sendo utilizados e etc., é possível inferir se alguma intervenção precisa ser feita ou não e, além disso, se é possível efetuá-la remotamente.

Como o Transmissor de *logs* precisa atuar durante todo o período de funcionamento do sistema, ele precisa consumir o mínimo de recurso possível e não prejudicar o desempenho do sistema como um todo, tornando-se assim uma ferramenta imperceptível para o usuário, mas essencial para os administradores dos laboratórios.

(vi) **Módulo de Monitoramento:** é o módulo responsável por receber os *logs* de todos os computadores dos laboratórios, organizar as informações e disponibilizar para consulta. Diferentemente dos computadores, esse módulo precisa estar sempre ativo para que dados de *logs* não sejam perdidos.

O Módulo de Monitoramento precisa dispor de funcionalidades capazes de permitir aos administradores dos laboratórios consultar as informações de maneira simples, clara e objetiva. A possibilidade de realizar consultas com diferentes parâmetros, geração de gráficos por período e extração de relatórios são funcionalidades extremamente relevantes e, em muitos casos, indispensáveis para se ter uma boa noção da situação real dos laboratórios.

Além de monitorar, é muito importante que esse módulo permita configurar alertas quando determinados eventos forem identificados nos *logs* recebidos. Por exemplo, a abertura de um gabinete e remoção de dispositivos internos.

Por meio desse módulo muitas ações imediatas ou preventivas podem ser realizadas. Além disso, com uma boa noção de como e quando os computadores estão sendo utilizados, é possível elaborar estratégias de atualização do parque computacional por completo ou em partes, sem impactar nas atividades da instituição.

(vii) **Módulo de Autenticação:** é responsável por receber as credenciais dos usuários e verificar se são válidas junto à base de dados de usuários local ou remota. Esse módulo garante que o usuário forneceu dados válidos e pertence ao grupo de usuários com permissão de acesso aos computadores dos laboratórios.

Quanto mais abrangente forem os perfis de usuários que podem se conectar nos computadores e, quanto mais formas de autenticação existirem, melhor; pois desta forma, solicitações de acesso excepcionais serão cada vez mais pontuais.

Esse mesmo módulo permite identificar o usuário e fornecer ao Gerenciador de Imagens de Disco qual imagem de disco deve ser inicializada, já que o modelo proposto prevê imagens individuais para cada usuário. Assim, todas as ações feitas pelo usuário durante o período em que utilizou o computador podem ser atreladas às credenciais fornecidas.

Protótipo

O sistema hospedeiro (que contém o Gerenciador e o Virtualizador de Imagens de Disco e o Módulo de Autenticação) foi desenvolvido sobre o Alpine, uma distribuição Linux minimalista com enfoque em segurança. Toda a construção do sistema hospedeiro é realizada por um *script* em *shell*, permitindo reprodutibilidade e facilitando o versionamento. O produto do processo de construção é uma imagem de sistema de arquivos. Foram desenvolvidos mecanismos para que essa imagem possa ser instalada por meio de um dispositivo de armazenamento USB (*pendrive*) sem a necessidade de intervenção manual após o *boot*, ou aplicada via rede como uma atualização sobre uma instalação preexistente, de forma atômica.

Para exibir uma tela de *login* gráfica para o usuário, foi desenvolvido e incluído no sistema hospedeiro o SDVM, um gerenciador de *login* compatível com temas do *Simple Desktop Display Manager* (SDDM) (EDMUNDSON et al., 2020). Ao contrário do SDDM, que depende de um sistema gráfico X11 completo, o SDVM depende somente das bibliotecas do Qt, permitindo que a tela de *login* seja renderizada diretamente por meio do subsistema *Direct Rendering Manager* (DRM) do *kernel*.

O SDVM foi integrado ao gerenciador de conexões de rede *NetworkManager* e forma o Módulo de Autenticação. Quando um usuário fornece suas credenciais, cria-se uma conexão de rede com autenticação 802.1X. As configurações da conexão são escolhidas com base no domínio do nome de usuário, permitindo que a ferramenta funcione em federações como o Eduroam. Para prover redundância, tenta-se conectar simultaneamente pela interface de rede cabeada e pela interface de rede sem fio.

Se a autenticação na rede obtiver sucesso, o SDVM clona um disco virtual a partir de uma imagem base de um sistema operacional de escolha do usuário (por exemplo, Windows 10 ou Ubuntu) utilizando a técnica de *copy-on-write* (COW). Foi desenvolvida uma versão da solução que utiliza o sistema de arquivos ZFS e seus volumes ZVol para prover os discos virtuais, e outra

que emprega arquivos no formato *qcow2*. Uma vez preparado o disco, o SDVM dispara um *script* que configura todos os pré-requisitos para virtualização com o QEMU.

A todo momento, o sistema hospedeiro pode ser gerenciado via SSH. O sistema envia *logs* e relatórios periódicos contendo métricas e informações de configuração para um servidor de coleta, neste caso, o Módulo de Monitoramento. Os *logs* são enviados utilizando Fluent Bit (Transmissor de *Logs*), e os relatórios são gerados por um *script* de coleta e transmitidos para um serviço HTTP usando a ferramenta curl. Essas informações podem ser reunidas para gerar um inventário de máquinas, detectar mudanças no *hardware* ou indicar a necessidade de intervenções físicas (por exemplo, ajustes de cabos de rede).

As configurações de BIOS são mantidas sob controle, padronizando-as e permitindo, por exemplo, garantir que todas as máquinas estejam com o recurso de *Wake on LAN* ativado, para que possam ser ligadas remotamente nos horários em que estiverem fora de uso e a distribuição de imagens ou atualização do sistema ocorra. Como Distribuidor de Imagens de Disco foi escolhida a ferramenta Aria2c,[1] que suporta, por exemplo, *downloads* via *Metalink* ou *BitTorrent*. Para criar arquivos .torrent foi utilizado o Mktorrent.[2]

Scripts foram desenvolvidos para a geração de imagens dos sistemas convidados juntamente com a instalação de *softwares*, compondo os Criadores de Imagens de Disco. Para o sistema operacional Windows 10, somente a instalação de *softwares* e configurações de ambiente foram automatizadas com o gerenciador de pacotes Chocolatey,[3] o instalador, atualizador e removedor de programas via rede automatizado WPKG[4] e o Win10-Initial-Setup-Script.[5] Já a geração da imagem do Arch Linux, instalação de programas e compactação da imagem foram automatizadas por completo por meio de um *script*.[6]

Uma versão preliminar da solução proposta por este trabalho, incluindo as funcionalidades de autenticação, virtualização e gerência, foi implantada como projeto piloto em laboratórios informatizados da Universidade Federal de São Carlos (UFSCar). Os modelos de máquinas disponíveis nesse ambiente são listados na Tabela 1.

[1] https://aria2.github.io/
[2] https://github.com/pobrn/mktorrent
[3] https://chocolatey.org/
[4] https://wpkg.org/
[5] https://github.com/Disassembler0/Win10-Initial-Setup-Script
[6] https://github.com/chaotic-aur/infra/blob/main/home/main-builder/chaotic-mkrootfs/ufscar

Tabela 1 – Modelos de computadores utilizados nos experimentos

Características	Optiplex 7010	Optiplex 9020	Opitplex 7040
Processador Intel	Core i7-3770	Core i7-4790	Core i5-6500
Geração	3ª geração	4ª geração	6ª geração
Nº de núcleos	4	4	4
Nº de threads	8	8	4
Frequência	3,40 GHz	3,60 GHz	3,20 GHz
Frequência Turbo máxima	3,90 GHz	4,00 GHz	3,60 GHz
Cache	8 MB	8 MB	6 MB
Velocidade do barramento	5 GT/s DMI	5 GT/s DMI2	8 GT/s DMI3
Thermal Design Power (TDP)	77 W	84 W	65 W
Tipo de memória	DDR3	DDR3	DDR4
Quantidade de memória	8 GB	8 GB	8 GB
Frequência da memória	1600 MHz	1333 MHz	2133 MHz
Versão da BIOS	A16	A07	1.14.0

Fonte: Produzido pelo autor

1.4 Síntese dos resultados

A autenticação está funcionando tanto por meio da rede cabeada como pela rede sem fio. A virtualização foi testada com passagem de periféricos (*passthrough*) nos três modelos distintos de computadores que compõem os laboratórios informatizados da UFSCar.

Foram realizados *benchmarks* de disco para verificar o desempenho das máquinas virtuais, variando a quantidade de memória RAM disponibilizada para os sistemas convidados (*guests*). Em alguns experimentos, a máquina virtual travou devido à forma como o ZFS do sistema hospedeiro gerencia a gravação em disco, impondo grande pressão sobre a memória RAM.

Para dar maior estabilidade ao sistema, optou-se por armazenar os discos virtuais em arquivos no formato *qcow2*. Esse formato oferece, por si só, o recurso de clonagem de discos virtuais com a técnica de *copy-on-write* (COW), sem depender para isso de suporte específico do sistema de arquivos do hospedeiro. O *qcow2* pode, portanto, ser utilizado em conjunto com qualquer sistema de arquivos convencional (como Ext4 ou XFS). Os *benchmarks* realizados dessa versão da solução demonstraram perda de desempenho com relação à versão baseada em ZFS, em troca do menor uso de memória RAM e estabilidade do sistema.

A distribuição de imagens via BitTorrent foi realizada em um ambiente real, sendo possível observar diferentes eventos por meio das ferramentas de monitoramento adotadas.

1.5 Justificativa

Este trabalho tem por objetivo solucionar um problema enfrentado por diversas instituições com laboratórios de ensino informatizados e por empresas com grande parque computacional. Com a arquitetura proposta, toda nova instalação de *software* ou atualização de sistema poderá ser executada remotamente após modificar a imagem base e distribuí-la para as demais máquinas.

Esse modelo de gerenciamento agilizará significativamente a atualização do parque em instituições nas quais esse tipo de intervenção hoje é realizado manualmente em cada máquina. Será possível, também, registrar a entrada de todos os usuários, além de isolar o espaço de cada usuário no sistema, permitindo que mais de um usuário utilize o mesmo computador com permissões de administrador (no sistema convidado) sem afetar os demais.

1.6 Organização do trabalho

Este trabalho está organizado em seis capítulos, sendo o primeiro este capítulo de introdução. Os quatro capítulos seguintes explicam cada uma das facetas do projeto — autenticação, virtualização, gerência e distribuição de imagens. Cada capítulo apresenta, sempre que aplicável, revisão de literatura, métodos, decisões de projeto e experimentos referentes a cada aspecto da solução e trabalhos futuros. O último capítulo expõe as considerações finais.

Capítulo 2
AUTENTICAÇÃO

A autenticação de usuários tem sido um dos requisitos prioritários para os administradores de rede desde o Marco Civil da Internet no Brasil (Lei n° 12965/2014), que obriga os provedores de acesso à Internet a registrarem os usuários que acessam sua rede (BRASIL, 2014).

Para cumprir a legislação no que diz respeito a registrar os acessos à rede sem fio, diversas instituições brasileiras já aderiram à federação Eduroam que, além de proporcionar uma forma padronizada para a instituição autenticar seus próprios usuários utilizando 802.1X, permite que ela receba facilmente visitantes de outras instituições federadas.

No entanto, o principal caso de uso para o qual a rede Eduroam foi inicialmente pensada é o de um usuário que configura suas credenciais em algum dispositivo móvel e deseja utilizá-lo tanto na sua instituição de origem como em outras instituições que visitar. Apesar de ser natural propor que um usuário empregue as mesmas credenciais para se autenticar em qualquer serviço relacionado à rede, a utilização das credenciais do Eduroam em terminais públicos de acesso, tais como computadores de salas de aula informatizadas, ainda é um tema pouco explorado.

Um dos poucos trabalhos que aborda o tema é um projeto de *software* publicado por Bello (2014), que é capaz de criar automaticamente uma conexão de rede com autenticação 802.1X no *NetworkManager* quando o usuário entra com suas credenciais em um gerenciador gráfico de *login*. No entanto, essa ferramenta é incapaz de funcionar de forma federada, além de não isolar a área pessoal de cada usuário nem verificar se a autenticação ocorreu com sucesso antes de liberar o acesso à máquina.

Para sanar essas limitações, desenvolvemos e publicamos uma ferramenta chamada `pam_eduroam` (LIMA et al., 2019), que vem sendo utilizada há mais de três anos nos terminais públicos de acesso da Sala 24h da UFSCar. Essa ferramenta serviu como embrião para as ideias adotadas pelo presente projeto, que serão apresentadas a seguir.

As Seções 2.1, 2.2 e 2.3 introduzem os componentes envolvidos no mecanismo de autenticação. Por fim, o processo de autenticação é sumarizado na Seção 2.4.

2.1 SDVM

O primeiro componente da solução de autenticação é a tela de *login*. A ideia inicial era utilizar o SDDM[1], um gerenciador gráfico de *login* customizável por temas escritos com a *Qt Modeling Language* (QML) e que atualmente é adotado como gerenciador de *login* oficial pelos projetos KDE e LXQt.

No entanto, verificou-se que os temas de SDDM eram suficientemente autocontidos para serem renderizados por qualquer programa baseado na biblioteca gráfica Qt. Como o uso do SDDM exigiria mapear alguns conceitos da solução de virtualização para conceitos tradicionais de *desktops* baseados em Linux (por exemplo, diferentes máquinas virtuais corresponderiam a diferentes sessões do X11), pareceu mais simples criar nosso próprio *software* de gerenciamento de *login*, que apelidamos de SDVM.

O SDVM é compatível com os temas do SDDM, no entanto em vez de utilizar o *Pluggable Authentication Modules* (PAM) para verificar as credenciais, ele embute dentro de seu próprio executável o código necessário para conversar por meio do barramento D-Bus com o *NetworkManager*, ao qual é delegada a tarefa de autenticação. Em vez de listar as sessões do sistema gráfico X11, o SDVM trabalha diretamente com o conceito de disco virtual, encarregando-se de listar as imagens base disponíveis e de iniciar o processo de clonagem quando um novo usuário acessa a máquina. Quando a autenticação ocorre com sucesso, o SDVM dispara um *script* em *shell* que faz a preparação do ambiente para o processo de virtualização com passagem de periféricos (*passthrough*), que será explicado no Capítulo 3.

Por ser um *software* com o propósito específico de gerenciar *login* em máquinas virtuais que assumem o controle dos periféricos, o SDVM pode sujeitar-se a diversas simplificações que não seriam possíveis no SDDM. O executável do SDVM depende somente das bibliotecas Qt, permitindo que ele execute sem a necessidade do X11, usando o *backend* EGLFS, que renderiza diretamente no dispositivo DRM fornecido pelo *kernel*. Além disso, o SDVM torna desnecessário criar dinamicamente novos usuários no sistema Linux hospedeiro, tarefa que o `pam_eduroam` precisa realizar nas instalações baseadas no SDDM ou em outros gerenciadores de *login* tradicionais. Essa simplificação é possível pois, no presente projeto, o próprio virtualizador atua como barreira de isolamento entre diferentes usuários (assim como em uma nuvem computacional).

A Figura 2 mostra a tela de *login* apresentada ao usuário pelo SDVM.

2.2 D-Bus

D-Bus é um barramento de mensagens para comunicação entre processos que pode ser utilizado de forma mais simples que as primitivas baseadas em compartilhamento de memória geralmente disponibilizadas pelo sistema operacional (PENNINGTON et al., 2016).

[1] <https://github.com/sddm/sddm>

Figura 2 – Tela de *login*

Fonte: Produzido pelo autor

Como o D-Bus já está consolidado em sistemas baseados em Unix, diversos aplicativos têm criado interfaces de comunicação que suportam D-Bus. Assim, cada aplicativo pode configurar uma interface de comunicação exibindo os métodos e os atributos que podem ser solicitados. Os métodos são ações que um aplicativo requisitante pode solicitar que outro aplicativo execute. Os atributos são informações que um aplicativo pode requisitar de outro, por exemplo para consultar o estado de um recurso. Para implementar o processo de autenticação, foi criado um canal de comunicação entre o SDVM e o *NetworkManager*, intermediado pelo D-Bus, utilizando a biblioteca QtDBus.

2.3 NetworkManager

NetworkManager é um serviço controlador de rede dinâmico que gerencia as conexões de rede, procurando manter as interfaces e as conexões ativas sempre que possível. O serviço *NetworkManager* inicializa junto com o sistema hospedeiro e executa como um *daemon*. Na solução proposta, o sistema inicia sem conexão ativa ou com conexão a uma rede confinada (para permitir que o computador seja acessado remotamente pelo administrador), e permanece assim até que o usuário forneça suas credenciais.

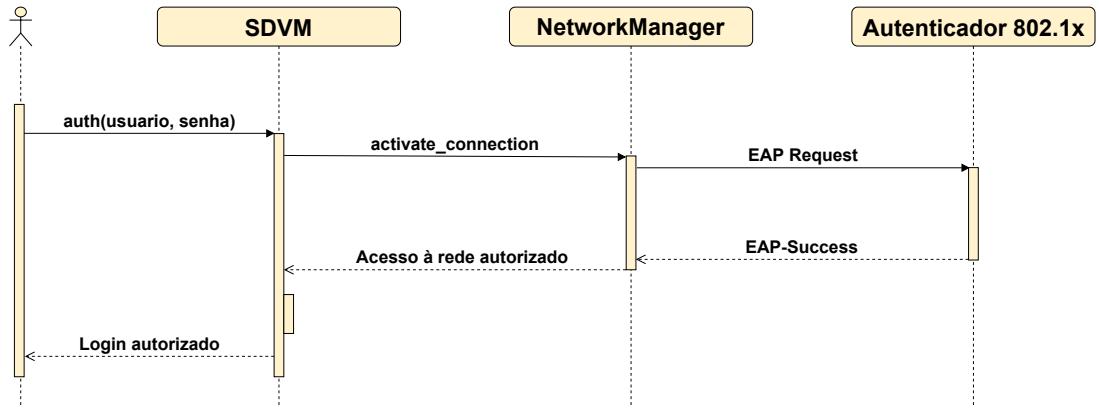
Quando o usuário fornece os dados para autenticação, o *NetworkManager* é acionado para criar uma nova conexão na rede cabeada e outra na rede sem fio. O *NetworkManager* configura as

métricas da tabela de rotas do sistema operacional para que apenas uma das conexões seja de fato utilizada, priorizando a conexão cabeada. Ambos os tipos de conexão são estabelecidos com base no padrão de autenticação 802.1X e utilizam os protocolos *Extensible Authentication Protocol* e *Tunneled Transport Layer Security* (EAP-TTLS) para que os dados não sejam transmitidos em texto claro (CONGDON et al., 2003).

Após a autenticação ser validada pelo servidor *Remote Authentication Dial-In User Service* (RADIUS) associado ao autenticador, o *NetworkManager* define alguma das novas conexões como primária, permitindo o acesso à Internet.

2.4 Resumo do processo de configuração da rede com *NetworkManager*

Figura 3 – Diagrama de sequência para autenticação efetuada com sucesso



Fonte: Produzido pelo autor

Na Figura 3, é possível observar o fluxo de acionamento dos módulos para que a autenticação de um usuário seja efetuada. Primeiramente, o usuário fornece nome de usuário e senha na tela de *login* do sistema. O SDVM recebe esses dados e utiliza o protocolo 802.1X para configurar a conexão de rede em conjunto com os protocolos EAP-TTLS, permitindo que as credenciais sejam comunicadas ao autenticador.

Após o autenticador aceitar as credenciais do usuário, o *NetworkManager* habilita o acesso à Internet e o SDVM repassa o controle ao *script* que prepara a máquina virtual para o usuário. Caso os dados não sejam validados, o *NetworkManager* permanece sem conexão ou conectado a uma rede confinada.

A principal limitação da abordagem proposta é que as especificações de autenticação e os certificados raiz de Infraestrutura de Chaves Públicas (ICP) de cada instituição federada cujos usuários possam vir a acessar a máquina precisam ser previamente instalados e mantidos atualizados. Como trabalho futuro, pretende-se carregar no sistema hospedeiro todos os certificados e

especificações disponíveis na base de dados do Eduroam CAT[2], permitindo que visitantes de variadas instituições possam acessar as máquinas sem a necessidade de configurações adicionais.

[2] <https://cat.eduroam.org>

Capítulo 3
VIRTUALIZAÇÃO

Virtualização é a prática de apresentar a um usuário ou aplicativo uma versão virtual de algum recurso computacional de forma que este mantenha a funcionalidade do recurso real. Trata-se de uma técnica essencial para promover o compartilhamento de recursos.

A abordagem tradicionalmente adotada em laboratórios informatizados não emprega o conceito de virtualização. Em vez disso, restringe-se ao máximo as ações do usuário, na tentativa de impedir que este possa prejudicar o funcionamento do sistema para os próximos que o utilizarão. Por exemplo, é comum encontrar sistemas nos quais os estudantes não podem instalar programas por iniciativa própria, ou que restauram o disco para o estado original sempre que a máquina é reiniciada.

Fazendo um paralelo com serviços *web*, esses laboratórios assemelham-se à antiga prática de hospedagem compartilhada (*shared hosting*), na qual o usuário é forçado a utilizar as linguagens de programação instaladas no servidor (por exemplo, PHP) e o isolamento entre diferentes usuários se dá pela configuração de permissões e pela proibição do acesso como administrador (*root*). Há alguns anos, houve um declínio na procura por esse tipo de serviço, com a popularização das VPS (*Virtual Private Servers*) e, posteriormente, das nuvens computacionais, que dão maior flexibilidade para o provisionamento de serviços (BUYYA et al., 2009).

A proposta de empregar virtualização em salas de aula informatizadas visa dar ao usuário a percepção de controle completo da máquina, mantendo sempre que possível as funcionalidades às quais o usuário teria acesso se aquela fosse sua máquina pessoal. Neste sentido, procura-se uma ampliação da flexibilidade para o usuário, similar à que ocorreu com a introdução de VPS e nuvens computacionais no cenário de serviços de hospedagem (HARDAWAY et al., 2005).

Mais que desburocratizar o ambiente de ensino, pretende-se abrir portas para novos tipos de experimento em sala de aula (GASPAR et al., 2008). Por exemplo, em uma aula de Redes de Computadores, os estudantes poderão escrever sua própria pilha TCP/IP e interagir com os colegas usando *raw sockets*. Em uma aula de Sistemas Operacionais, os estudantes poderão desenvolver e carregar seu próprio módulo de *kernel* no mesmo sistema que estão usando.

Além disso, empregar virtualização traz vantagens no que diz respeito à gerência das

máquinas, uma vez que o sistema com o qual o usuário tem contato (convidado) permanece isolado do sistema que será utilizado para as tarefas administrativas (hospedeiro), facilitando a atualização de ambos os tipos de sistema e a persistência dos serviços de gerência.

Para o presente trabalho, optou-se por adotar o virtualizador *Kernel-based Virtual Machine* (KVM) em conjunto com o QEMU. A escolha deu-se devido à maturidade dessa pilha de *software*, que é a opção padrão em soluções conceituadas de computação em nuvem, como o OpenStack (SEFRAOUI et al., 2012).

Este capítulo introduz os principais desafios técnicos enfrentados para prover uma solução de virtualização adequada aos laboratórios de ensino. A Seção 3.1 apresenta a técnica de passagem de periféricos. A Seção 3.2 explica a abordagem de provisionamento de discos virtuais distintos para cada usuário.

3.1 Passagem de periféricos

Por padrão, o QEMU virtualiza somente o processador da máquina virtual: os dispositivos periféricos são emulados[1]. Por exemplo, em vez de expor diretamente a controladora gráfica do sistema hospedeiro, o QEMU emula uma placa de vídeo PCI Cirrus CLGD 5446, imitando por meio de *software* o comportamento da placa e refazendo as operações gráficas solicitadas pelo sistema convidado com alguma biblioteca gráfica (SDL ou GTK) ou com o dispositivo DRM. De maneira similar, dispositivos como mouse, teclado, interface de áudio, interface de rede e discos também são emulados, imitando algum modelo de dispositivo que, em algum momento do passado, foi lançado no mercado. Essa abordagem pode gerar dois problemas: um de usabilidade e outro de desempenho.

Quanto à usabilidade, espera-se que o fato de estar utilizando uma máquina virtual fique transparente para o usuário. Não seria aceitável, por exemplo, que o usuário precisasse interagir com o sistema hospedeiro para liberar o acesso a um *pendrive* que tenha sido conectado a uma porta USB da máquina, ou para que o sistema convidado fosse capaz de detectar e utilizar um novo dispositivo conectado em uma saída de vídeo ou de áudio. Apesar de ser possível preparar o sistema hospedeiro para lidar automaticamente com essas situações, passar ao sistema convidado o controle direto sobre os periféricos soluciona o problema de forma mais simples.

Quanto ao desempenho, a emulação gera uma sobrecarga de processamento desnecessária, pois exige imitar um dispositivo de *hardware* que foi projetado para um sistema físico real e que implementa protocolos que não necessariamente correspondem à maneira mais eficiente de trocar dados entre sistema hospedeiro e convidado.

É possível mitigar o problema de desempenho por meio de uma técnica denominada para-virtualização, na qual o QEMU expõe um dispositivo que nunca existiu no mercado como

[1] <https://qemu.weilnetz.de/doc/qemu-doc.html#QEMU-PC-System-emulator>

hardware real, mas que foi projetado desde o início para trocar dados de forma eficiente em um ambiente de virtualização. Para dar suporte a esses dispositivos no sistema convidado, é necessário instalar *drivers* da suíte Virtio (RUSSELL, 2008).

No entanto, para alguns tipos de dispositivo, é difícil resolver o problema de desempenho sem repassar o controle diretamente para a máquina virtual. O principal exemplo é o dispositivo de vídeo. Hoje, mesmo controladoras de vídeo integradas possuem GPUs relativamente poderosas. Esses dispositivos são complexos e as APIs gráficas ainda são pouco interoperáveis, portanto implementá-los por meio de para-virtualização é uma tarefa árdua.

Pelas razões expostas acima, este projeto optou por repassar o controle de praticamente todos os periféricos para a máquina virtual. Apenas disco e interface de rede, que são dispositivos sobre os quais o sistema hospedeiro precisa reter certo controle por questões de gerência, foram para-virtualizados utilizando Virtio.

A técnica de passagem de periféricos (*passthrough*) pode ser implementada em qualquer máquina que possua uma IOMMU (LIU et al., 2010). A IOMMU é uma unidade que permite aplicar o conceito de memória virtual à entrada e saída de dispositivos de *hardware* (AMIT et al., 2012). A Intel deu à sua implementação de IOMMU o nome Intel VT-d, já a AMD deu o nome AMD-Vi. Para passar um periférico para a máquina virtual, o sistema hospedeiro emprega a IOMMU para mapear as regiões de memória usadas pelo periférico em endereços que o sistema convidado consiga acessar.

No Linux, é possível aplicar essa técnica a dispositivos PCI por meio do módulo `vfio-pci` (WILLIAMSON, 2012). Para fazer a passagem de um dispositivo, é necessário primeiro fazer o *unbind* de qualquer *driver* que o esteja utilizando. Depois, o identificador do dispositivo precisa ser informado ao módulo `vfio-pci`, que passa a tomar o controle dos endereços do barramento pertencentes àquele dispositivo.

No entanto, aplicar a técnica de *passthrough* a controladoras de vídeo integradas exige alguns cuidados especiais. Os processadores da Intel só passaram a suportar[2] a passagem do vídeo integrado a partir da arquitetura de codinome *Sandy Bridge*, também conhecida como Intel Core de "segunda geração". Além disso, o dispositivo precisa ser exposto em um barramento, número de dispositivo e número de função predefinidos (`00:02.0`), pois a vBIOS tenta localizar a controladora em um endereço fixo (*hardcoded*). Por fim, é necessário atentar-se a possíveis falhas no *software* de virtualização que causem comportamentos inesperados para o *driver* de vídeo no sistema convidado, tema que será abordado nas Subseções 3.1.1 e 3.1.2.

A Figura 4 mostra o comando para inicializar o QEMU com *passthrough* de periféricos em uma máquina Dell Optiplex 7010. Foi construído um *script* denominado `run_guest.sh` que consulta o modelo da máquina na BIOS e constrói esse comando automaticamente a partir de uma lista de identificadores de dispositivos. Além do modelo Optiplex 7010, o *script* foi

[2] <https://forum.proxmox.com/threads/guide-intel-intergrated-graphic-passthrough.30451/>

configurado para lidar também com os modelos Optiplex 9020 e 7040 da Dell. Esses três modelos são os que podem ser atualmente encontrados em salas informatizadas mantidas pela Secretaria Geral de Informática (SIn) da UFSCar e que foram utilizados na fase de implantação do projeto.

Figura 4 – Comando de inicialização de máquina virtual na Dell Optiplex 7010

```
qemu-system-x86_64
-accel kvm
-cpu host
-smp cores=8
-machine pc
-nodefaults
-boot menu=off
--mem-path /hugepages
--mem-prealloc
-m 6910M
-device vfio-pci-nohotplug,host=00:02.0,bus=pci.0,addr=0x02
-device vfio-pci-nohotplug,host=00:1d.0,bus=pci.0,addr=0x1d
-device vfio-pci-nohotplug,host=00:1a.0,bus=pci.0,addr=0x1a
-device vfio-pci-nohotplug,host=00:14.0,bus=pci.0,addr=0x14
-device vfio-pci-nohotplug,host=00:1b.0,bus=pci.0,addr=0x1b
-device virtio-scsi-pci,id=scsi
-drive if=none,id=hd_sys,file=/dev/zvol/tank/user/xz8ZaIS-3
   Ri1Mqeu0oa7DMJgDGuZbdw9tOI5EA/windows,format=raw
-device scsi-hd,drive=hd_sys
-netdev user,hostfwd=tcp:0.0.0.0:3389-:3389,id=net_sys
-device virtio-net-pci,netdev=net_sys,mac=74:86:7a:fd:de:36
-rtc base=localtime,clock=vm
```

Fonte: Produzido pelo autor

3.1.1 *Patch* no QEMU para desabilitar *stolen memory*

Stolen memory é o nome dado à região de memória "roubada" da CPU pela controladora de vídeo integrada, tornando essa região inacessível para o *software* (INTEL, 2014). Trata-se de um recurso que não é padrão de dispositivos PCI em geral, sendo utilizado especificamente por controladoras de vídeo integradas. Por esse motivo, parece ainda ser controversa a forma como esse recurso deve ser exposto ao sistema convidado.

Em fevereiro de 2017, desenvolvedores da Intel enviaram um *patch*[3] ao projeto QEMU que impedia o *driver* que executa no sistema convidado de reservar *stolen memory*, ficando limitado a utilizar a quantidade que estiver pre-alocada.

No entanto, em março de 2017, os mesmos desenvolvedores reverteram o *patch*[4], alegando que ele causava problemas com o *driver* para Windows das controladoras de vídeo

[3] <https://github.com/qemu/qemu/commit/93587e3af3a259deac89c12863d93653d69d22b8>
[4] <https://github.com/qemu/qemu/commit/c2b2e158cc7b1cb431bd6039824ec13c3184a775>

integradas em processadores Intel Core a partir da nona geração (*Cannon Lake*).

O problema é que o *patch* é necessário para que o *driver* de controladoras de vídeo mais antigas que as de nona geração funcione. A documentação sobre *passthrough* da própria Intel[5] sugere reaplicar o *patch* de fevereiro de 2017.

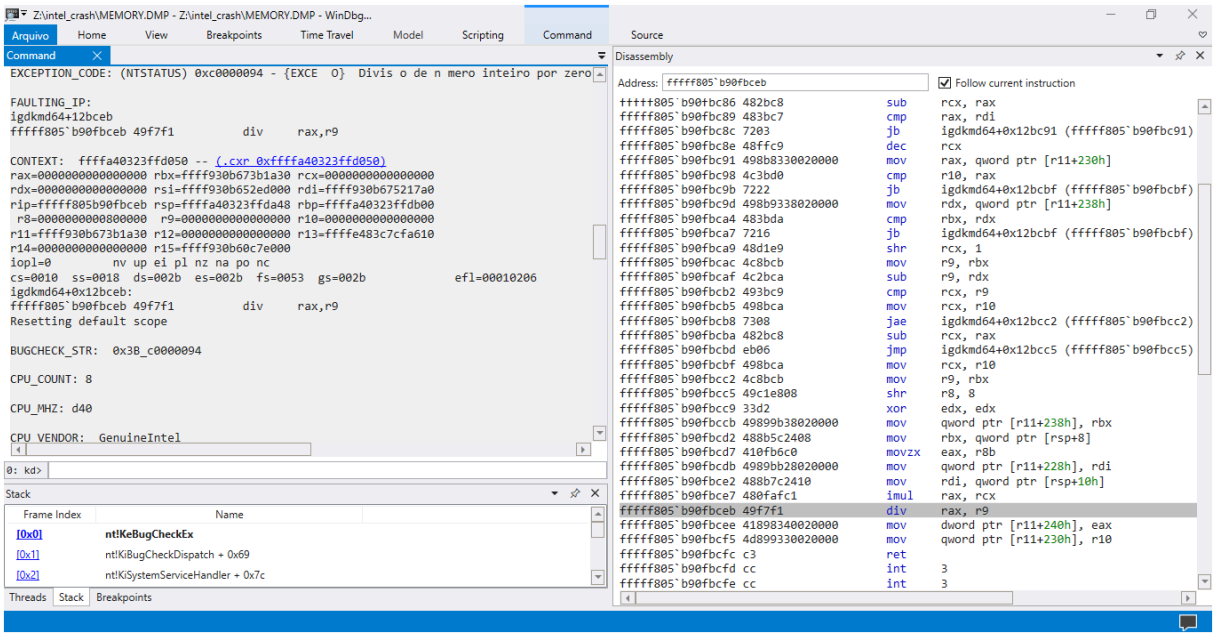
Portanto, foi necessário compilar um pacote de QEMU contendo esse *patch*. No futuro, para permitir suporte às controladoras de vídeo mais recentes, pretende-se sugerir ao projeto QEMU uma forma de habilitar ou desabilitar a disponibilização da *stolen memory* por meio de uma opção de linha de comando.

3.1.2 *Patch* no KVM para implementar os MSRs necessários

Para testar a estabilidade e o desempenho da GPU, foram executados *benchmarks* e jogos de computador. Com o Linux como sistema convidado, todos os testes obtiveram sucesso. No entanto, notou-se que qualquer aplicativo gráfico que operasse em tela cheia causava uma pane quando o sistema convidado era o Windows.

Para investigar a situação, configurou-se o Windows para gerar um arquivo de despejo de memória (*memory dump*) em caso de pane. O arquivo de despejo produzido ao reproduzir a falha foi, então, analisado com a ferramenta WinDbg Preview[6], da própria Microsoft. Como ilustra a Figura 5, a pane era causada por um erro de divisão de inteiro por zero em espaço de *kernel*.

Figura 5 – Tela da ferramenta WinDbg Preview



Fonte: Produzido pelo autor

5 <https://github.com/intel/gvt-linux/wiki/GVTd_Setup_Guide#33-build-qemu>
6 <https://www.microsoft.com/pt-br/p/windbg-preview/9pgjgd53tn86>

O código ao redor da instrução de divisão foi estudado para entender de onde surgia o valor zero no denominador. Identificou-se que o código pertencia à função mostrada na Figura 6, que aparentemente é responsável por calcular a frequência atual do processador.

Para resolver o problema, foi criado um *patch* que faz o KVM informar sempre o valor 1 como conteúdo dos MSRs `APERF` e `MPERF`. Essa modificação fez com que o *driver* de vídeo para a máquina Dell Optiplex 7010 funcionasse sem causar panes.

No entanto, ainda ocorriam panes com programas de tela cheia na máquina Dell Optiplex 9020. Refazendo a análise, notou-se que o código do *driver* para a controladora de vídeo dessas máquinas era ligeiramente diferente, de forma que também surgia uma divisão por zero caso o segundo *byte* menos significativo do MSR `PLATFORM_INFO` fosse zero. O *patch* foi, então, complementado para preencher esse *byte* com o valor 30. Desta forma, o KVM simula estar executando em um processador com frequência fixa de 3 GHz. Essa modificação fez com que as panes cessassem em todos os três modelos de máquina testados.

3.2 Discos virtuais

Um disco virtual isolado é provisionado para cada usuário distinto que efetuar *login* no sistema. A proposta básica é aplicar a técnica de *copy-on-write* (COW), que permite criar rapidamente (geralmente em tempo inferior a um segundo) um novo disco virtual a partir de uma imagem base, além de ocupar no disco físico apenas o espaço necessário para armazenar diferenças com relação à imagem original.

Foram exploradas duas formas de aplicar a técnica de COW. Na primeira forma, os discos virtuais são armazenados em um sistema de arquivos que implementa essa técnica, como o ZFS ou o Btrfs. Para este projeto, foi escolhido o ZFS, devido à sua maturidade. Na segunda forma, os discos virtuais são armazenados em arquivos no formato *qcow2* (que implementa a funcionalidade de COW) dentro de um sistema de arquivos convencional, como Ext4, XFS ou F2FS. Para este projeto, foi escolhido o Ext4, novamente devido à sua maturidade.

A ideia do modelo proposto neste projeto é o computador do laboratório receber uma imagem base de cada sistema convidado que será virtualizado. Utiliza-se a técnica de COW para permitir que vários usuários tenham um disco virtual local próprio ocupando o mínimo de espaço possível no disco. Desta forma, não há necessidade de armazenar centenas de discos virtuais com o mesmo tamanho da imagem base, mas sim, manter apenas uma imagem base com vários discos virtuais menores de usuários fazendo leituras a partir dela.

O objetivo deste modelo não é guardar as imagens do usuário por muito tempo, mas somente durante um período; oferecendo privacidade dos seus dados, de arquivos, do histórico de acesso, além de manter a integridade dos arquivos salvos em casos de queda de energia. Ou seja, caso isso aconteça e o usuário tenha salvo seus arquivos antes, ele poderá reiniciar o computador

Figura 6 – Trecho descompilado de *driver* de vídeo que usa os MSRs `APERF` e `MPERF`

```
void __fastcall calc_cpufreq(unkcls01 *this)
{
  unsigned __int64 plat_info; // r8
  unsigned __int64 aperf; // r10
  unsigned __int64 mperf; // rbx
  unsigned __int64 ts; // rdi
  unsigned __int64 last_ts; // rax
  unsigned __int64 time_diff; // rcx
  unsigned __int64 last_aperf; // rax
  unsigned __int64 last_mperf; // rdx
  unsigned __int64 mperf_diff; // r9
  bool v11; // cf
  unsigned __int64 aperf_diff; // rcx

  plat_info = __readmsr(MSR_PLATFORM_INFO);
  aperf = __readmsr(MSR_IA32_APERF);
  mperf = __readmsr(MSR_IA32_MPERF);
  ts = __rdtsc();
  last_ts = this->last_ts;
  time_diff = ts - last_ts;
  if ( last_ts >= ts )
    --time_diff;
  last_aperf = this->last_aperf;
  if ( aperf < last_aperf || (last_mperf = this->last_mperf, mperf <
     last_mperf) )
  {
    aperf_diff = aperf;
    goto LABEL_8;
  }
  mperf_diff = mperf - last_mperf;
  v11 = time_diff >> 1 < mperf - last_mperf;
  aperf_diff = aperf;
  if ( !v11 )
  {
LABEL_8:
    mperf_diff = mperf;
    goto LABEL_9;
  }
  aperf_diff = aperf - last_aperf;
LABEL_9:
  this->last_mperf = mperf;
  this->last_ts = ts;
  // pane na operação de divisão da linha abaixo
  this->cpufreq100MHz = aperf_diff * BYTE1(plat_info) / mperf_diff;
     // BYTE1(plat_info) é o max_nonturbo_ratio
  this->last_aperf = aperf;
}
```

Fonte: Produzido pelo autor

e recuperá-los a partir da sua imagem salva.

A universidade tem cada vez menos condições de prover ou ser um repositório local. Por isso, esse modelo de virtualização possibilita formas do usuário salvar seus arquivos na nuvem ou de integrar uma unidade de armazenamento externa e ter disponibilidade dos dados em qualquer lugar.

3.2.1 Arquitetura baseada em ZFS

ZFS é um sistema de arquivos transacional, com fortes garantias quanto à integridade de dados em caso de quedas de energia ou panes no sistema operacional (BONWICK et al., 2003). Todos os blocos possuem um *checksum* com algoritmo configurável. Por padrão, utiliza-se o algoritmo Fletcher4 (FLETCHER, 1982). Os *checksums* são organizados em uma árvore de Merkle para permitir verificação eficiente de integridade ao longo de toda a hierarquia do sistema de arquivos (MERKLE, 1990).

Geralmente o ZFS é configurado para gerenciar por completo um ou mais discos, formando uma *pool* (conjunto de blocos). Esse esquema dispensa o uso de partições, substituindo-as pelo conceito mais flexível de *datasets* (conjuntos de dados). Cada *dataset* corresponde a uma árvore de arquivos e diretórios (*filesystem*), que pode ser mapeada em um diretório do sistema operacional, ou a um volume (ZVol), que se comporta como um dispositivo de blocos. Pode-se reservar uma certa quantidade de *bytes* da *pool* para cada *dataset*, ou pode-se deixar a expansão de todos os *datasets* livre até o espaço máximo disponível na *pool*.

Com o ZFS, pode-se aplicar a técnica de COW de duas maneiras distintas: efetuando a clonagem de um *dataset*; ou criando *snapshots*, que servem como um mecanismo de versionamento dos dados contidos naquele *dataset*. Em ambos os casos, conforme o novo *dataset* desvia-se do conteúdo do *dataset* original, apenas os blocos que forem modificados ou adicionados contribuem para aumentar o espaço fisicamente ocupado no disco.

A Tabela 2 mostra a organização em *datasets* implementada pelo presente projeto. Em `tank/ROOT/alpine`, armazena-se a raiz do sistema de arquivos do hospedeiro. Em `tank/base/<sistema>`, guardam-se as imagens base dos sistemas operacionais disponíveis para escolha pelo usuário. Em `tank/user/<usuário>/<sistema>`, são armazenados os volumes clonados para cada usuário a partir das imagens base.

Para evitar que um adversário possa utilizar seu nome de usuário para construir caminhos de *dataset* maliciosos, nomes de usuário são processados com a função SHA-3 (BERTONI et al., 2013), produzindo uma saída de tamanho padronizado que é, então, codificada com Base64Url (JOSEFSSON, 2006).

O ZFS suporta adicionar metadados arbitrários aos *datasets*, criando pares chave–valor denominados propriedades. Para cada *dataset* abaixo de `tank/base`, utilizam-se propriedades para definir:

Tabela 2 – Lista de *pools* do ZFS

NAME	USED	AVAIL	REFER	MOUNTPOINT
tank	104G	795G	96K	none
tank/ROOT	723M	795G	96K	none
tank/ROOT/alpine	723M	795G	609M	legacy
tank/base	103G	795G	96K	none
tank/base/linux	51.6G	847G	56K	-
tank/base/windows	51.6G	847G	56K	-
tank/user	288K	795G	96K	none
tank/user/xz8ZaIS-3...dw9tOI5EA	96K	795G	96K	none
tank/user/xz8ZaIS-3...dw9tOI5EA/linux	0B	795G	56K	-
tank/user/xz8ZaIS-3...dw9tOI5EA/windows	0B	795G	56K	-
tank/user/zon8ZkPC...PafOfSxQ	96K	795G	96K	none
tank/user/zon8ZkPC...PafOfSxQ/windows	0B	795G	56K	-

Fonte: Produzido pelo autor

a) o nome com o qual aquela opção deve ser exibida na tela de *login* do SDVM (`br.ufscar:name`);

b) o tipo de sistema operacional contido na imagem (`br.ufscar:ostype`);

c) se aquela opção deve aparecer selecionada por padrão (`br.ufscar:default`).

Para cada *dataset* abaixo de `tank/user`, adiciona-se uma propriedade que armazena a data e a hora em que o usuário correspondente efetuou *login* pela última vez naquele sistema convidado (`br.ufscar:lastlogin`). Essa informação é utilizada para permitir a remoção dos *datasets* mais antigos quando o disco estiver com nível de ocupação muito elevado.

O nível de ocupação do disco é considerado elevado quando:

$$\texttt{volsize} - \texttt{usedds} \geq \texttt{avail} \tag{3.1}$$

Na equação acima,

a) `volsize` (propriedade `ZFS_PROP_VOLSIZE`) corresponde ao tamanho em *bytes* do volume virtual enxergado pelo sistema convidado, que é também a quantidade máxima de *bytes* que o *dataset* pode vir a ocupar;

b) `usedds` (propriedade `ZFS_PROP_USEDDS`) corresponde a quanto o *dataset* do usuário expandiu-se desde que foi criado até o momento, ou seja, à quantidade de *bytes* usados pelo *dataset* do usuário que está efetuando *login*, desconsiderando os blocos que estiverem apenas fazendo referência à imagem base;

c) `avail` (propriedade `ZFS_PROP_AVAILABLE`) corresponde à quantidade de *bytes* que estão disponíveis em disco no momento, ou seja, que podem ser usados pelo *dataset* do usuário para expandir-se.

Para ilustrar melhor o significado da equação, consideremos duas situações hipotéticas: (1) o usuário está *logando* pela primeira vez em um sistema convidado com volume de 50GB –

nesse caso, `usedds` é zero e são necessários 50GB livres para garantir que o usuário consiga utilizar o disco do sistema convidado como desejar, sem correr o risco de estourar o espaço disponível no sistema hospedeiro; (2) o usuário está retornando a esse mesmo sistema convidado, depois de ter previamente modificado uma quantidade blocos de disco equivalente a 1GB de espaço – nesse caso, `usedds` é 1GB e são necessários "apenas" 49GB livres, pois o espaço de 1GB anteriormente modificado já está reservado para aquele usuário.

Diversos *benchmarks* foram realizados com a arquitetura baseada em ZFS. Os resultados são apresentados na Subseção 3.3.3.

3.2.2 Arquitetura baseada em *qcow2*

Nesta segunda proposta de arquitetura, prevê-se o uso do recurso de *backing files* do formato *qcow2*, que provê a funcionalidade de COW. Com esse recurso, é possível implementar o análogo à operação de clonagem de volumes do ZFS mesmo que os arquivos *qcow2* sejam armazenados em um sistema de arquivos convencional, como o Ext4.

Ao criar um novo volume com o comando `qemu-img create`, pode-se utilizar a opção `-b` para especificar o *backing file* – o arquivo *qcow2* referente à imagem base que se deseja clonar. O arquivo *qcow2* criado como resultado (que denominamos volume de usuário) não contém inicialmente nenhum bloco, mas faz referência ao *backing file*.

Ao utilizar o volume de usuário como disco de uma máquina virtual, todos os blocos que não forem encontrados são lidos do *backing file*. Nas operações de escrita, o *backing file* nunca é modificado. Todo bloco criado ou modificado é gravado sempre no volume de usuário.

Como os arquivos *qcow2* não suportam metadados customizados, as propriedades de cada imagem base (nome de exibição, tipo de sistema operacional e opção padrão) são configuradas em um arquivo separado no formato INI.

A única propriedade que precisaria ser armazenada para os volumes de usuário é a última data de *login*. No entanto, como nesta arquitetura os volumes são armazenados na forma de arquivos, o próprio sistema de arquivos dentro do qual eles estão contidos mantém controle da data da última alteração (*mtime*). Essa informação é usada para determinar quais volumes estão há mais tempo sem uso e que devem ser removidos caso o disco alcance um nível de ocupação muito elevado.

Novamente, utiliza-se a equação 3.1 para definir o nível de ocupação do disco como elevado. Mas, desta vez:

a) `avail` corresponde à quantidade de *bytes* livres na partição que contém os volumes de usuário;

b) `usedds` corresponde ao tamanho atual em *bytes* do arquivo *qcow2* que contém o volume do usuário que está efetuando *login*;

c) `volsize` corresponde ao tamanho em *bytes* do disco virtual assim como percebido pela máquina virtual.

Por simplicidade, a implementação atual utiliza um valor fixo `reserve` escolhido em configuração em vez da definição de `volsize` acima. Para que a implementação seguisse a definição acima (sendo, portanto, totalmente análoga à da arquitetura baseada em ZFS), seria necessário interpretar o texto produzido como saída pelo comando `qemu-img info`, já que não existe uma API padrão para consultar essa informação. Esse procedimento introduziria um potencial para quebra de compatibilidade em atualizações do QEMU. No futuro, pretende-se controlar esse risco com a introdução de testes automatizados dessa funcionalidade.

Na Seção 3.3 são apresentados os *benchmarks* de ambas as arquiteturas para efeito de comparação e escolha da mais adequada para implantação nos laboratórios.

3.3 Experimentos e Resultados

Nesta seção são apresentados testes gráficos e de disco efetuados nos três modelos de computador. Foram realizados experimentos em ambas as arquiteturas de virtualização propostas e também em cenários sem virtualização para efeito de comparação.

Na Subseção 3.3.1 são apresentados testes gráficos dos ambientes sem virtualização e com virtualização. Na Subseção 3.3.2 são apresentados testes de disco executados no sistema operacional Windows 10 instalado diretamente no disco físico dos três modelos. Na Subseção 3.3.3 os testes de disco foram executados no sistema convidado Windows 10, tendo o hospedeiro Alpine Linux instalado em discos rígidos. Nessas três subseções mencionadas, os testes foram executados com os sistemas instalados em discos rígidos Seagate Barracuda, modelo ST1000DM003, com capacidade de armazenamento de 1 TB e 7200 RPM.

Na Subseção 3.3.4 os testes de disco foram executados no sistema convidado Windows 10 com o hospedeiro Alpine Linux instalado em unidades de estado sólido Crucial, modelo CT480BX500SSD1, com capacidade de armazenamento de 480 GB.

3.3.1 Testes gráficos

A Tabela 3 registra as pontuações obtidas no *benchmark* gráfico Cloud Gate, da suíte 3DMark[7]e a Figura 7 mostra o *benchmark* em execução em uma das máquinas. Ao analisar a Tabela 3 é possível observar que houve uma pequena perda de desempenho nos testes realizados nos ambientes virtualizados em relação aos ambientes sem virtualização. Também pode-se destacar o fato de ambas as propostas de virtualização obterem desempenhos semelhantes, exceto no modelo Dell Optiplex 9020, no qual a arquitetura com ZFS obteve pontuação bem mais alta que a arquitetura baseada em *qcow2*.

[7] <https://www.3dmark.com>

Tabela 3 – Resultados do *benchmark* Cloud Gate, da suíte 3DMark

Modelo	Sem Virtualização		Arquitetura com ZFS		Arquitetura com *qcow2*	
	Pont.	Percentil: 28/6/21	Pont.	Percentil: 4/7/19	Pont.	Percentil: 18/7/21
Dell 7010	5973	27%	5334	23%	5211	19%
Dell 9020	6163	27%	6767	23%	5420	19%
Dell 7040	7956	33%	7393	29%	7466	33%

Fonte: Produzido pelo autor

Figura 7 – *Benchmark* Cloud Gate, da suíte 3DMark, em execução

Fonte: Produzido pelo autor

3.3.2 *Benchmarks* de disco sem virtualização

Para efeito de comparação, o sistema operacional Windows 10 foi instalado nos três modelos de computador diretamente no disco físico sem utilizar virtualização. Foram realizados testes de desempenho de escrita e leitura de disco com a ferramenta NetworkDLS DiskMark[8], versão 1.0.0.7. Cronometrou-se, também, o tempo de inicialização do sistema.

Escrita e leitura variando tamanho de bloco e distribuição da RAM

Na ferramenta DiskMark, é possível variar os parâmetros de tamanho de bloco em bytes (*Set Size*), número de blocos que será lido ou escrito (*Rounds*) e a quantidade de vezes que o teste será executado (*Runs*). Nos experimentos realizados, os tamanhos de bloco testados foram de 128 kB, 256 kB, 512 kB, 1 MB, 2 MB e 4 MB. O parâmetro *Rounds* foi ajustado para 128 e *Runs* para 320. Assim, o tamanho total para leitura e escrita (*Total Set Size*) obtido foi, respectivamente, de 5 GB, 10 GB, 20 GB, 40 GB, 80 GB e 160 GB. Cada experimento foi repetido 10 vezes para dar uma noção da variância estatística das medidas.

Além disso, os testes mencionados nesta seção foram executados nos três modelos de computador e variando a distribuição de memória RAM para igualarem-se aos cenários propostos para as máquinas virtuais. Por esse motivo, foram removidos da memória total 768 MB, 1024 MB, 2048 MB, 3072 MB e 4096 MB, respectivamente. Assim, cada teste mencionado foi executado com o sistema operacional configurado com:

[8] <http://www.networkdls.com/Software/View/DiskMark>

a) 6682 MB de RAM;

b) 6426 MB de RAM;

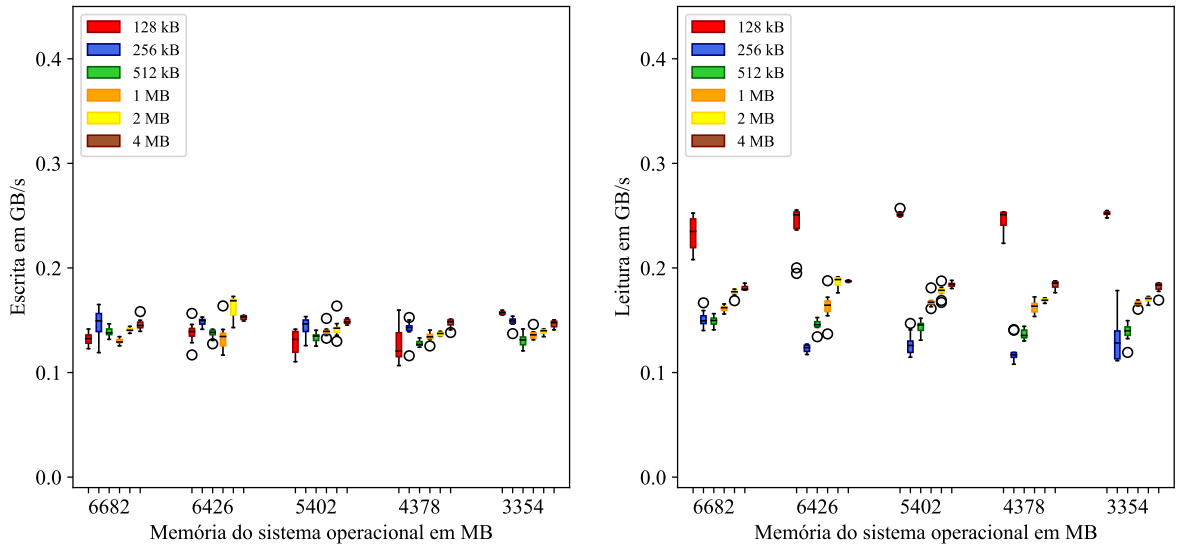
c) 5402 MB de RAM;

d) 4378 MB de RAM;

e) 3354 MB de RAM.

Na Figura 8, é exibido o *benchmark* de disco do modelo Optiplex 7010. É possível visualizar uma similaridade de desempenho em todos os casos de escrita no disco físico, tendo uma taxa média de escrita entre 0,11 GB/s e 0,17 GB/s. Já nos testes de leitura, os blocos de tamanho 128 kB obtiveram melhor desempenho em todos os casos com taxa média de escrita próxima de 0,25 GB/s.

Figura 8 – Taxa média de escrita e leitura por tamanho de bloco – Optiplex 7010

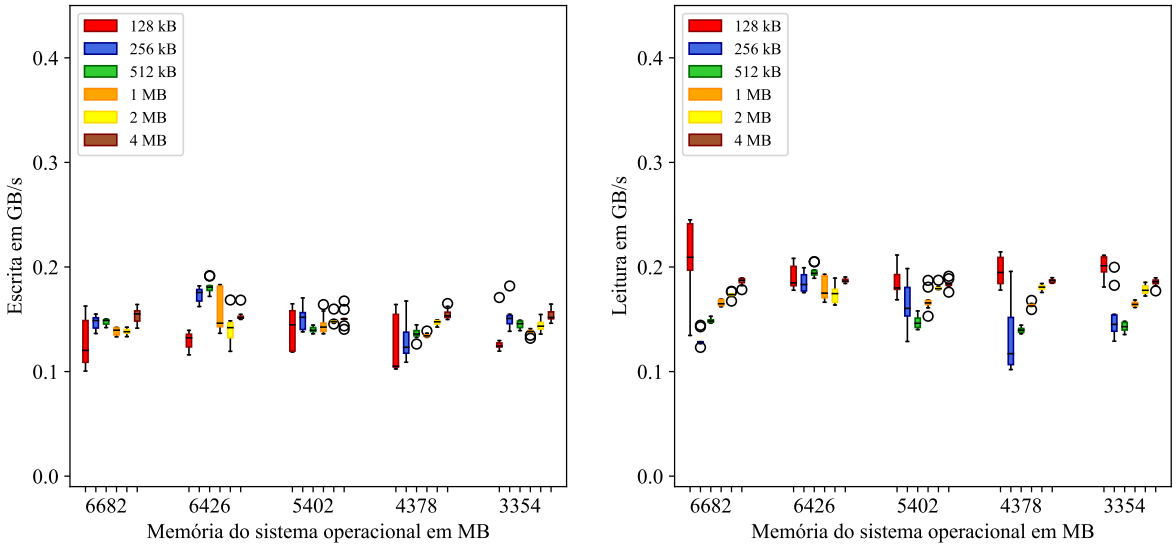


Fonte: Produzido pelo autor

Na Figura 9, tem-se os experimentos executados no modelo Optiplex 7040. Os testes de escrita tiveram um desempenho similar independentemente do tamanho de bloco escolhido, não ultrapassando também a taxa de 0,20 GB/s. Nos testes de leitura, todos os casos de teste apresentaram desempenho semelhantes, exceto no caso em que o sistema foi configurado com 6682 MB de RAM, no qual o bloco de 128 kB apresentou taxas de leitura acima de 0,20 GB/s.

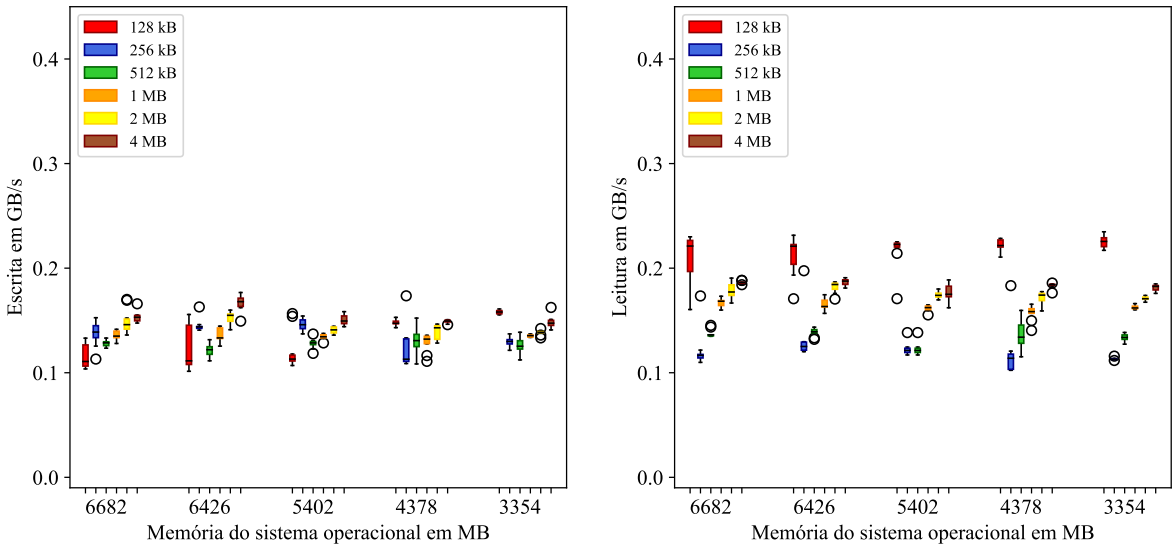
Na Figura 10, são exibidos os resultados dos experimentos executados no modelo de computador Optiplex 9020. Nos experimentos de escrita, as taxas ficaram concentradas entre 0,10 GB/s e 0,18 GB/s em todos os casos de teste. No testes de leitura, é possível observar um desempenho melhor para os blocos de tamanho de 128 kB, com taxas acima 0,20 GB/s na maior parte dos testes.

Figura 9 – Taxa média de escrita e leitura por tamanho de bloco – Optiplex 7040



Fonte: Produzido pelo autor

Figura 10 – Taxa média de escrita e leitura por tamanho de bloco – Optiplex 9020

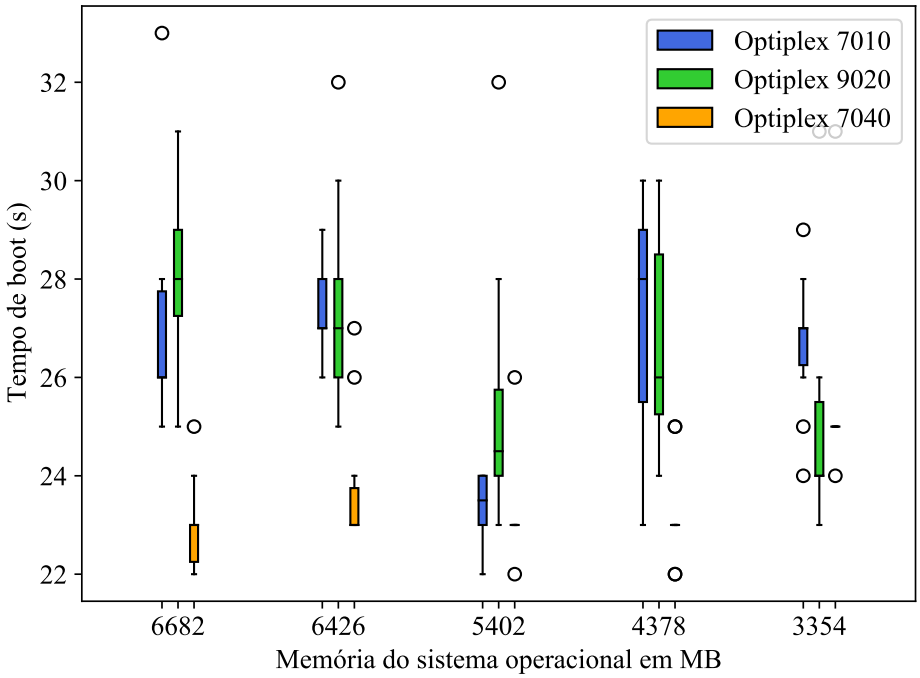


Fonte: Produzido pelo autor

Tempo de inicialização do sistema

O tempo de inicialização do sistema também foi medido. O experimento foi realizado para cada modelo de computador variando a quantidade de memória. Cada cenário foi executado 10 vezes. O experimento foi realizado apenas com o sistema operacional Windows 10. Os tempos cronometrados são apresentados na Figura 11.

Figura 11 – Tempo de inicialização do sistema operacional Windows 10



Fonte: Produzido pelo autor

Pode-se observar na Figura 11 que o modelo Optiplex 7040 apresentou os menores tempos de inicialização, com a maioria das medidas iguais ou próximas de 23 segundos em todos os casos. É interessante destacar que os modelos Optiplex 7010 e Optiplex 9020 apresentaram melhores resultados quando o sistema foi configurado com 5402 MB e 3354 MB em comparação aos outros casos de teste, nos quais tinham mais memória disponível, como 6682 MB e 6426 MB por exemplo. Apesar disso, todos os modelos apresentaram bons tempos de inicialização, com tempos iguais ou inferiores a 33 segundos.

3.3.3 Arquitetura baseada em ZFS

Para validar a solução baseada em ZFS, também foram realizados testes de desempenho de escrita e leitura de disco com a ferramenta NetworkDLS DiskMark, versão 1.0.0.7. Cronometrou-se, também, o tempo de inicialização do sistema. Por fim, foram acompanhadas as atualizações do Windows para verificar o quão eficiente era a geração de imagens completas e de imagens incrementais. Todos os experimentos desta seção foram realizados com a arquitetura

baseada em ZFS. A implementação de ZFS utilizada foi o ZFS on Linux versão 0.8.1. A configuração adotada para *cache* de disco foi a padrão do QEMU (*writeback*). Os mesmos parâmetros e o mesmo roteiro dos experimentos descritos na Seção 3.3.2 foram seguidos removendo-se apenas o caso de teste:

a) 768 MB de RAM para o hospedeiro e 6682 MB de RAM para o convidado.

Escrita e leitura variando tamanho de bloco e distribuição da RAM

Os testes mencionados foram executados nos três modelos de computador e variando a distribuição de memória RAM entre o hospedeiro Alpine e o convidado Windows. Cada teste mencionado foi executado com:

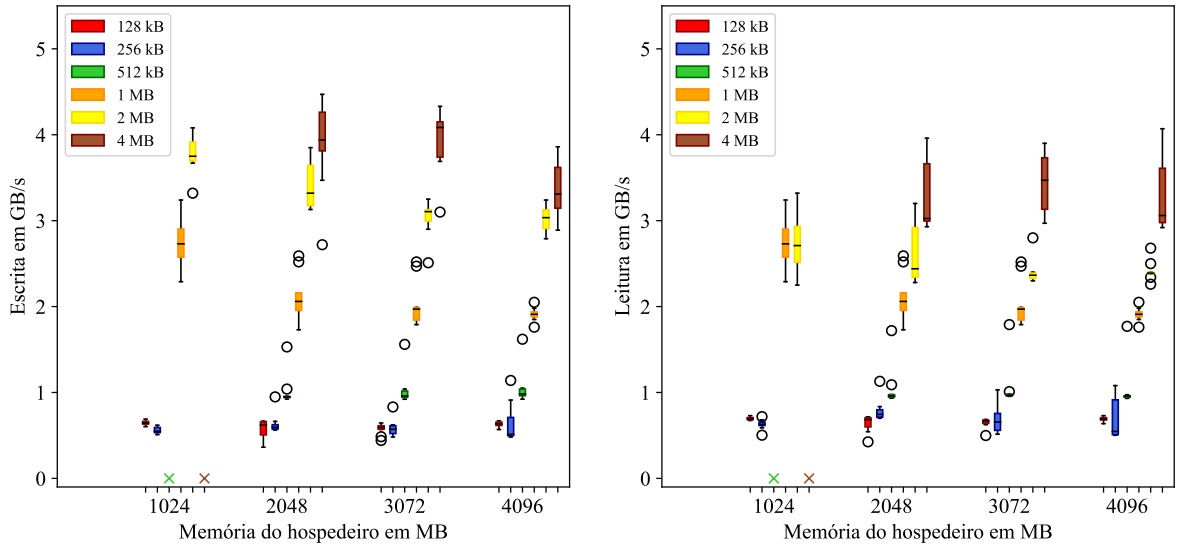
a) 1024 MB de RAM para o hospedeiro e 6426 MB de RAM para o convidado;

b) 2048 MB de RAM para o hospedeiro e 5402 MB de RAM para o convidado;

c) 3072 MB de RAM para o hospedeiro e 4378 MB de RAM para o convidado;

d) 4096 MB de RAM para o hospedeiro e 3354 MB de RAM para o convidado.

Figura 12 – Taxa média de escrita e leitura por tamanho de bloco – Optiplex 7010



Fonte: Produzido pelo autor

Na Figura 12, é exibido o *benchmark* de disco do modelo Optiplex 7010. É possível visualizar que o melhor desempenho de escrita no disco virtual ocorreu com blocos de tamanho 4 MB e deixando entre 2048 MB e 3072 MB de memória RAM para o hospedeiro.

Ocorreram problemas de estabilidade ao executar o *benchmark* com blocos de tamanho 512 kB e 4 MB na configuração com 1024 MB de RAM para o hospedeiro, por isso foi atribuído o valor zero para essas execuções no gráfico. O sistema convidado desligava repentinamente e, ao

acessar o sistema hospedeiro via SSH, verificava-se que o processo do QEMU havia sido morto pelo *OOM killer* do *kernel*, ou que havia encerrado com a mensagem "*qemu: qemu_thread_create: Resource temporarily unavailable*". Portanto, esse problema deve estar relacionado com um consumo demasiado de memória pela implementação ZFS on Linux em condições de elevado acesso ao disco. O problema persistiu mesmo tentando configurar o ZFS para alocar a menor quantidade possível de memória, por meio das opções de *boot* `zfs.zfs_arc_max=67108864 zfs.zfs_arc_meta_limit=33554432 zfs.zfs_scan_mem_lim_fact=128`.

Na Figura 13, tem-se os experimentos executados no modelo Optiplex 7040. Os testes de escrita com blocos de 1 MB, 2 MB e 4 MB tiveram um desempenho similar nos casos em que o hospedeiro ficou com 2048 MB e 3072 MB de RAM. Já nos testes de leitura, os blocos com tamanho de 2 MB obtiveram o melhor desempenho. Nesse modelo de computador, os testes com blocos de 512 kB e 4 MB também causaram problemas de estabilidade quando foram deixados 1024 MB de RAM para o hospedeiro.

Na Figura 14, são exibidos os resultados dos experimentos executados no modelo de computador Optiplex 9020. No casos de leitura, é possível observar um desempenho bastante similar para todos os tamanhos de bloco quando são deixados pelo menos 2048 MB de RAM para o hospedeiro. Nos experimentos de escrita, ocorreu um comportamento parecido, exceto por uma queda brusca de desempenho para blocos de 4 MB quando são deixados 3072 MB ou mais de RAM para o hospedeiro.

Possível fenômeno de fragmentação da memória

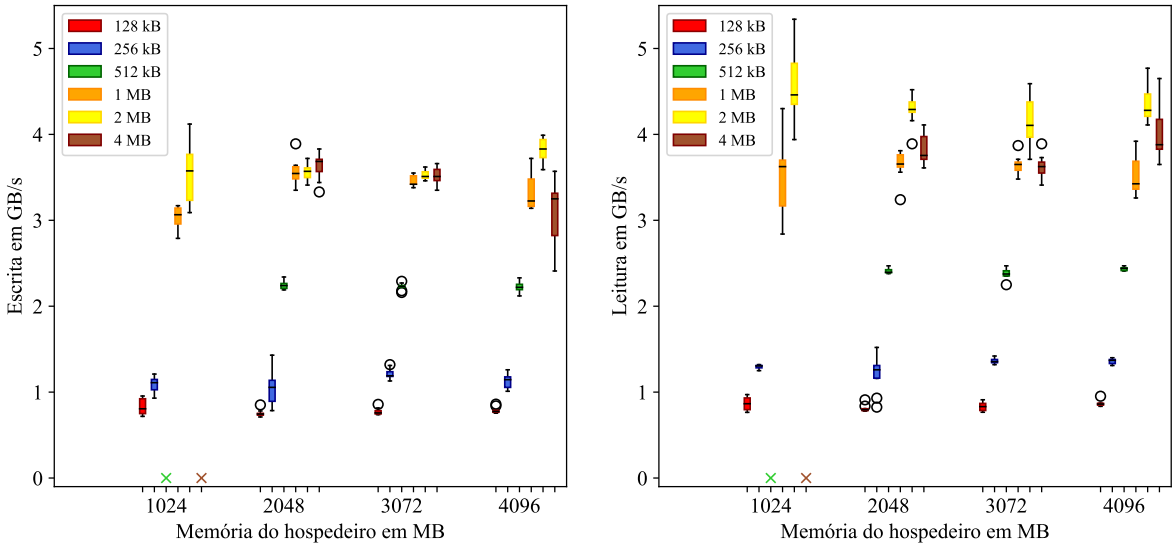
Notou-se que, quando se utilizava blocos de 4 MB (e também com blocos de 2 MB, embora menos frequentemente), não era possível reproduzir as mesmas taxas de escrita e leitura ao repetir o experimento. Mesmo reiniciando o sistema convidado (Windows), não era possível restaurar o mesmo desempenho de antes. Era necessário reiniciar a máquina por completo (incluindo sistema hospedeiro) para repetir o experimento e obter as mesmas taxas.

Uma possível hipótese é que isso tenha acontecido devido à fragmentação de memória, embora essa hipótese não tenha sido validada pelo presente trabalho.

Para observar melhor esse fenômeno, foram realizadas 5 medidas reiniciando a máquina após cada teste (sem fragmentação) e outras 5 medidas sem reiniciar a máquina entre testes (supostamente, com fragmentação). Todos esses experimentos foram realizados deixando 2048 MB de memória RAM para o hospedeiro. As Figuras 15 e 16 apresentam essas medidas para os modelos Dell Optiplex 7010 e 7040, respectivamente. No cenário com fragmentação, a taxa de escrita é até duas ordens de grandeza mais baixa, o que leva o tempo total de execução do teste a ser bem mais elevado que no cenário sem fragmentação.

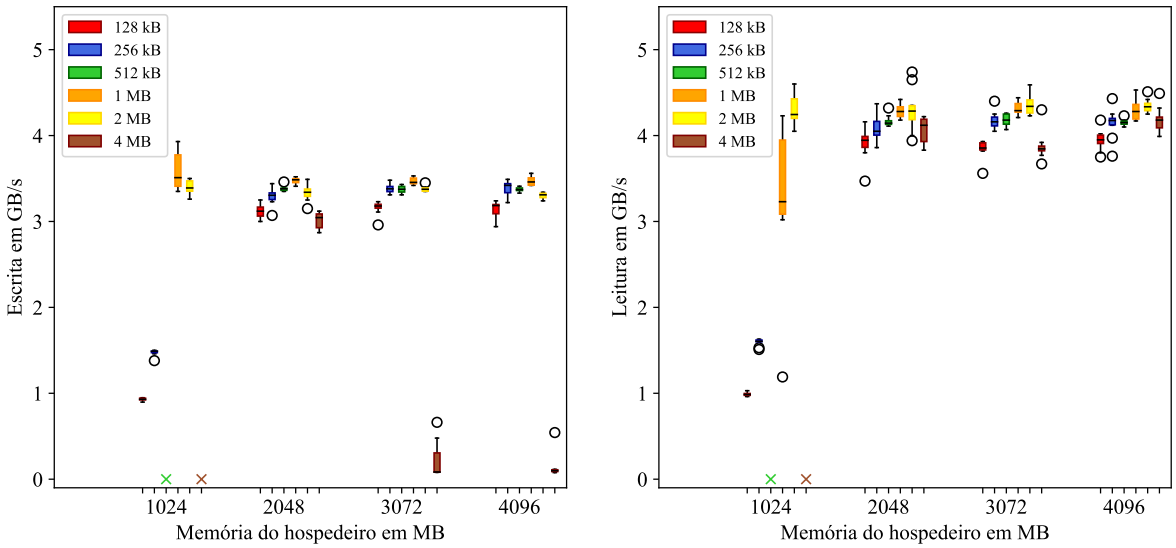
Por algum motivo, esse fenômeno ocorreu com pouca frequência no modelo Dell Optiplex 9020. Por isso, optou-se por não repetir essas medidas para esse modelo.

Figura 13 – Taxa média de escrita e leitura por tamanho de bloco – Optiplex 7040



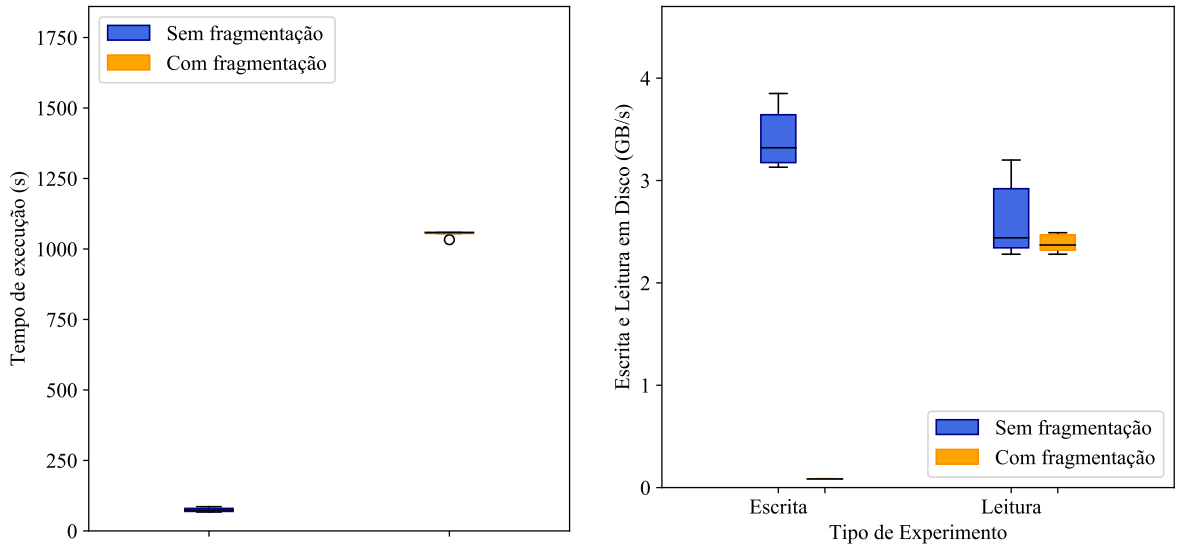
Fonte: Produzido pelo autor

Figura 14 – Taxa média de escrita e leitura por tamanho de bloco – Optiplex 9020



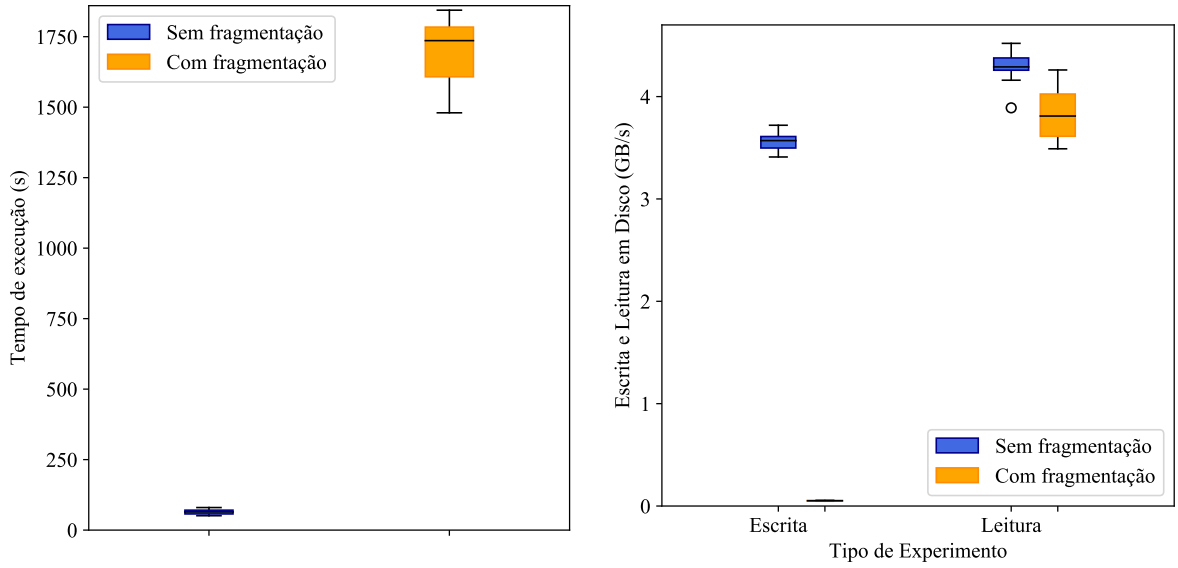
Fonte: Produzido pelo autor

Figura 15 – Cenários com fragmentação de memória – Dell Optiplex 7010



Fonte: Produzido pelo autor

Figura 16 – Cenários com fragmentação de memória – Dell Optiplex 7040

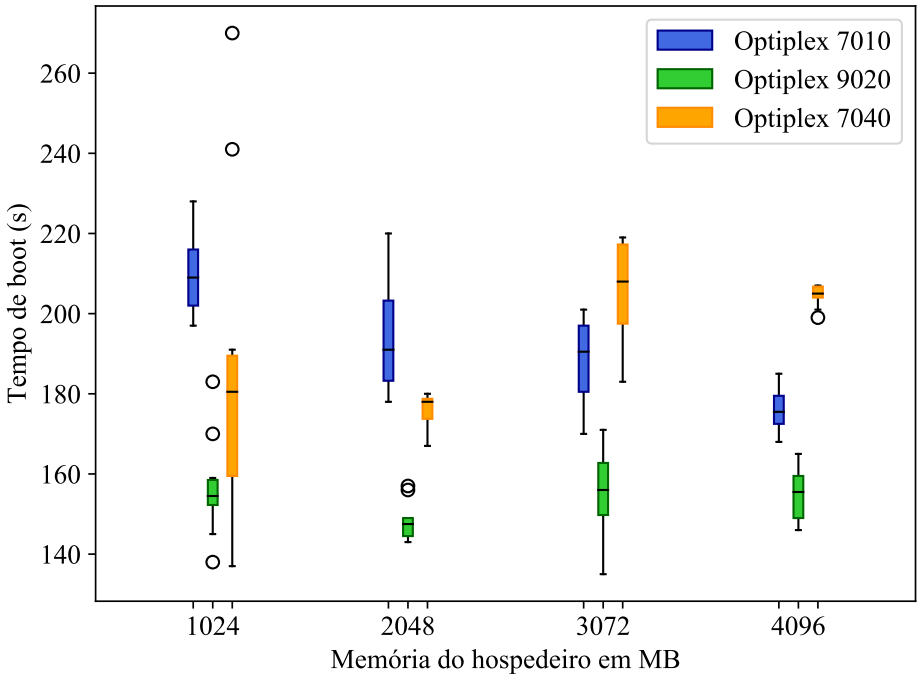


Fonte: Produzido pelo autor

Tempo de inicialização do sistema

O tempo de inicialização do sistema como um todo também foi medido. O experimento foi realizado para cada modelo de computador variando a quantidade de memória disponibilizada para o hospedeiro e para o convidado. Cada cenário foi executado 10 vezes. O experimento foi realizado apenas com o sistema convidado Windows 10. O tempo de *boot* retratado na Figura 17 é o tempo de inicialização do convidado somado ao tempo de inicialização do hospedeiro.

Figura 17 – Tempo de inicialização do sistema hospedeiro mais inicialização de um convidado Windows 10



Fonte: Produzido pelo autor

Pode-se observar que o modelo Optiplex 9020 apresentou os menores tempos de inicialização, obtendo o melhor desempenho quando 2048 MB de memória RAM são deixados para o hospedeiro. Observa-se também que aumentar mais que isso a quantidade de memória do hospedeiro prejudica o desempenho da inicialização do convidado. Apesar de possuir o mesmo número de núcleos que os outros, possuir memória do tipo DDR3 e de ter o segundo processador mais recente dentre os três modelos, o Optiplex 9020 pode ter apresentado o melhor desempenho por ter um processador Core i7 e 8 *threads*, ao passo que o Optiplex 7040 tem um processador Core i5 e apenas 4 *threads*.

Acompanhamento de atualizações do Windows

Atualizações de sistema são importantes para receber as últimas correções de estabilidade e de segurança. No entanto, se o sistema convidado de cada disco de usuário for atualizado por conta própria, haverá um grande desperdício de espaço em disco e de tráfego na rede. Portanto,

espera-se que as atualizações sejam gerenciadas a partir de um ponto central e distribuídas na forma de novas imagens base. Para estudar esse processo, foi realizado um acompanhamento das atualizações lançadas para o Windows 10.

O objetivo principal desse acompanhamento foi verificar o tempo para geração e compactação de *datasets* nos quais o sistema Windows está inserido. É possível gerar um *dataset* contendo todo o sistema Windows ou extrair as diferenças com relação ao *snapshot* de uma versão anterior. Para cada versão do Windows foi tirado um *snapshot* do sistema e verificado se o processo de geração da diferença entre *snapshots* era mais rápido do que a geração do *dataset* integral do sistema. Essa informação contribui para decidir o método mais vantajoso a ser executado antes de distribuir a imagem por meio da rede.

Para realizar os testes, foi utilizado um computador do modelo Optiplex 9020 que possui 1 TB de disco rígido. O tempo de geração e de compactação de cada *dataset* foi medido 3 vezes para se obter uma boa estimativa. Na Figura 18, é exibido o comportamento do tamanho do *dataset* no qual foi colocada a imagem de disco do Windows. O *dataset* cresce de 11 GB para 16 GB da versão 17.03 para a 18.03, e depois se mantém estável em aproximadamente 13 GB.

Como mostra a Figura 19, o tempo de criação e compactação do *dataset* praticamente acompanha o crescimento do *dataset*, exceto no caso da geração do *dataset* da versão 19.03. Por mais que o tamanho da imagem não tenha crescido da versão 18.09 para a 19.03, o tempo de geração e compactação do *dataset* aumentou significativamente. Uma hipótese é que a grande quantidade de operações de gravação e exclusão tenha gerado fragmentação do disco. Por fim, verifica-se que, em todos os testes, extrair e compactar diferenças entre *snaphosts* de um *dataset* foi vantajoso com relação a extrair e compactar o *dataset* completo.

3.3.4 Arquitetura baseada em *qcow2*

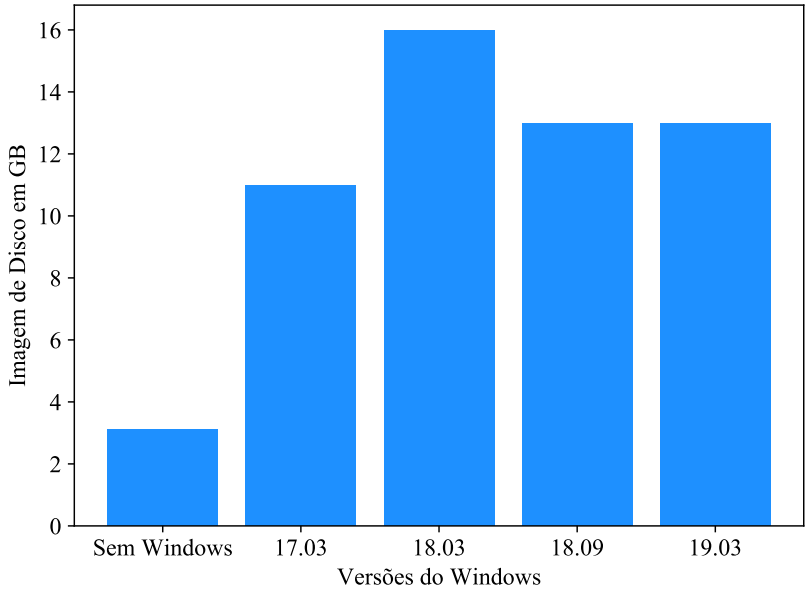
Para validar a solução baseada em *qcow2*, também foram realizados testes de desempenho de escrita e leitura de disco com a ferramenta NetworkDLS DiskMark, versão 1.0.0.7. Cronometrou-se, também, o tempo de inicialização do sistema. A configuração adotada para *cache* de disco foi a padrão do QEMU (*writeback*). Os mesmos parâmetros e o mesmo roteiro dos experimentos descritos na Seção 3.3.3 foram seguidos acrescentando-se o caso de teste:

a) 768 MB de RAM para o hospedeiro e 6682 MB de RAM para o convidado.

Escrita e leitura variando tamanho de bloco e distribuição da RAM

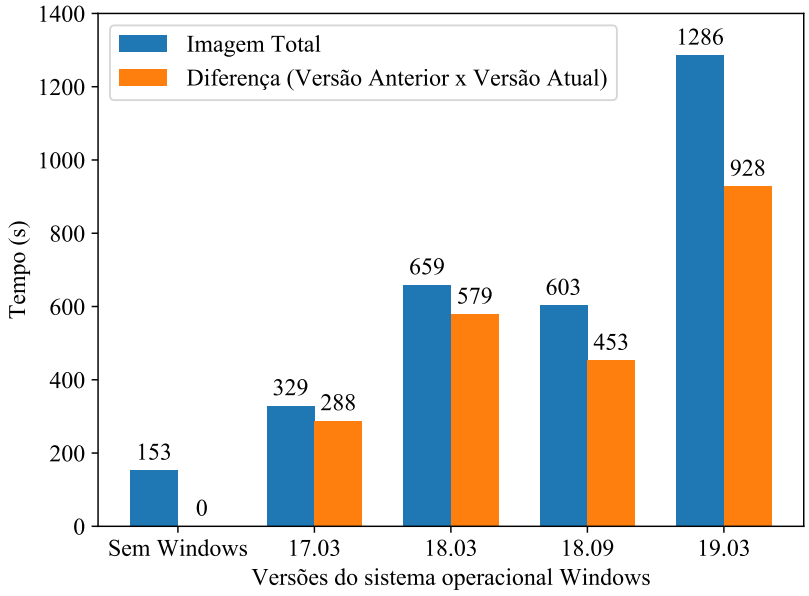
Na Figura 20, é exibido o *benchmark* de disco do modelo Optiplex 7010. É possível observar que o melhor desempenho de escrita no disco virtual ocorreu com blocos de tamanho 2 MB e deixando 2048 MB ou mais de RAM para o hospedeiro. Em relação ao *benchmark* de leitura, os blocos de 128 kB, 256 kB, 512 kB, 1 MB e 2 MB apresentaram desempenho praticamente constante em todos os casos propostos de reserva de RAM para o hospedeiro.

Figura 18 – Crescimento do *dataset* com a imagem de disco do Windows 10



Fonte: Produzido pelo autor

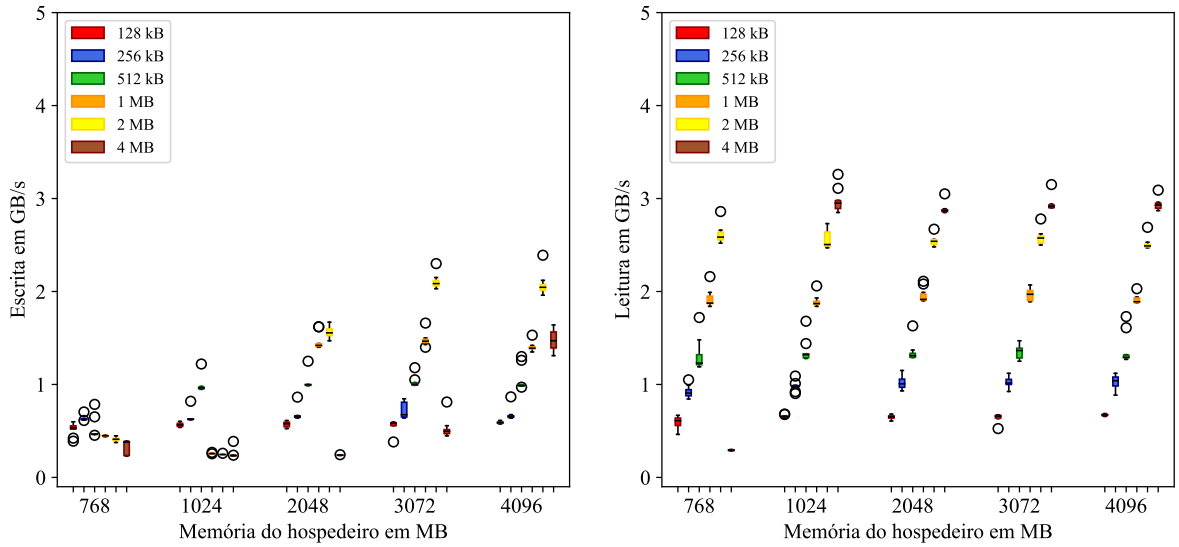
Figura 19 – Tempo médio para geração e compressão de imagens



Fonte: Produzido pelo autor

Os testes de leitura com blocos de 4 MB na Figura 20, por sua vez, apresentaram os melhores desempenhos deixando 1024 MB ou mais de RAM para o hospedeiro. Entretanto, no caso de teste do hospedeiro com apenas 768 MB há uma queda significativa no desempenho, provavelmente devido à redução da quantidade de memória disponível para o sistema hospedeiro realizar *cache* do disco virtual.

Figura 20 – Taxa média de escrita e leitura por tamanho de bloco – Optiplex 7010



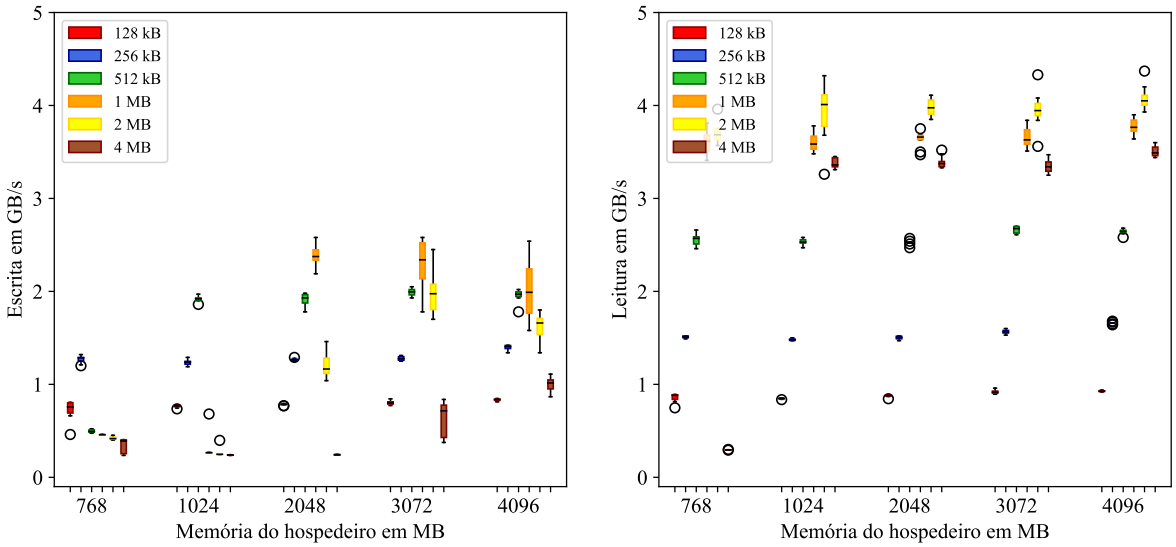
Fonte: Produzido pelo autor

Na Figura 21, tem-se os experimentos executados no modelo Optiplex 7040. Os testes de escrita com blocos de 1 MB apresentaram o melhor desempenho nos casos em que o hospedeiro ficou com 2048 MB, 3072 MB e 4096 MB. Nos outros casos, os blocos de tamanho 256 kB e 512 kB obtiveram o melhor desempenho nos casos em que o hospedeiro ficou com 768 MB e 1024 MB de RAM respectivamente. Já nos testes de leitura, os blocos com tamanho de 1 MB e 2 MB obtiveram os melhores desempenhos em todos os casos propostos de reserva de RAM para o hospedeiro.

Na Figura 22, são exibidos os resultados dos experimentos executados no modelo de computador Optiplex 9020. Nos testes de escrita, os blocos com tamanho de 2 MB apresentaram o melhor desempenho nos casos em que o hospedeiro ficou com 3072 MB e 4096 MB de RAM. Os blocos com tamanho de 256 kB, 512 kB e 1 MB obtiveram melhor desempenho nos casos em que o hospedeiro ficou com 768 MB, 1024 MB e 2048 MB de RAM respectivamente. Nos experimentos de leitura, é possível observar que os blocos com tamanho de 4 MB obtiveram o melhor desempenho nos casos em que o hospedeiro ficou com 1024 MB ou mais de RAM.

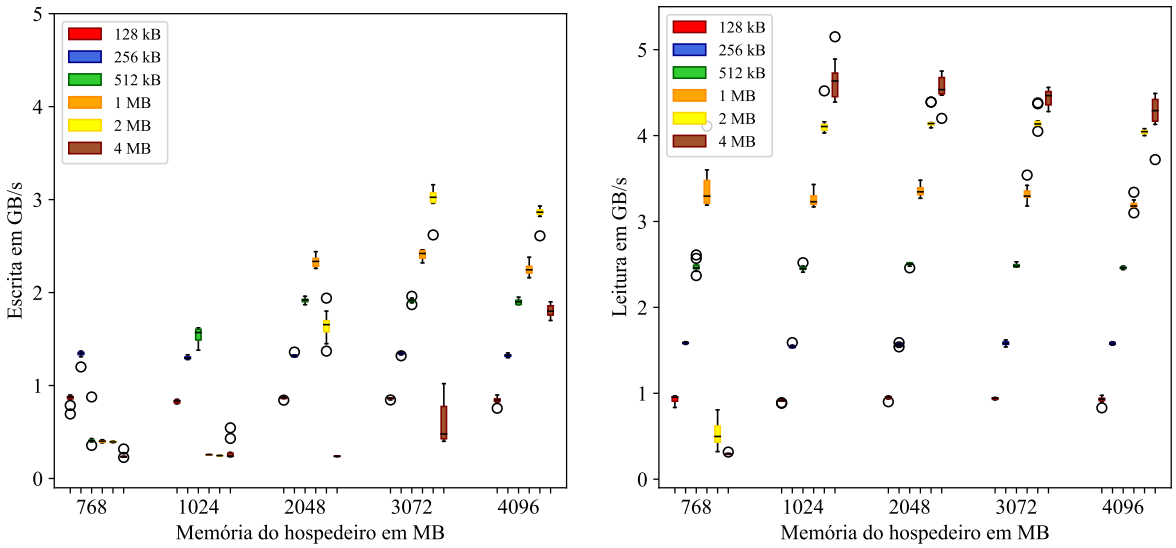
De maneira geral, o desempenho da arquitetura baseada em *qcow2* apresentou resultados relativamente inferiores à arquitetura baseada em ZFS quanto à taxa média de escritura e leitura em disco virtual, porém apresentou estabilidade em todos os experimentos propostos. Apesar

Figura 21 – Taxa média de escrita e leitura por tamanho de bloco – Optiplex 7040



Fonte: Produzido pelo autor

Figura 22 – Taxa média de escrita e leitura por tamanho de bloco – Optiplex 9020



Fonte: Produzido pelo autor

da arquitetura baseada em ZFS obter melhores resultados no quesito de desempenho, ela não forneceu garantia de funcionamento em alguns experimentos.

Em comparação aos testes de disco executados nos três modelos de computador sem efetuar a virtualização do Windows, ambas as arquiteturas de virtualização propostas apresentaram taxas de escrita e leitura muito mais elevadas, provavelmente porque o sistema hospedeiro Alpine Linux e o virtualizador QEMU fazem uso de camadas de *cache* adicionais.

Foi possível confirmar essa hipótese ao monitorar o uso de disco do hospedeiro Alpine Linux enquanto os testes de gravação eram executados no convidado Windows. Por exemplo, durante a execução de um dos testes, enquanto no sistema Alpine as taxas de escrita no disco não passavam de 250 MB/s, no Windows elas apresentavam uma média de 2 GB/s. Além disso, as gravações no sistema Alpine eram feitas a cada certo intervalo de tempo, diferentemente do que era mostrado no *benchmark* no Windows. Ou seja, provavelmente as gravações estavam sendo feitas no disco considerando o intervalo de tempo decorrido e a quantidade de dados em *cache*.

Tempo de inicialização do sistema

O tempo de inicialização do sistema como um todo também foi medido seguindo o mesmo roteiro descrito na Seção 3.3.3. O experimento foi realizado apenas com o sistema convidado Windows 10. O tempo de *boot* retratado na Figura 23 é o tempo de inicialização do convidado somado ao tempo de inicialização do hospedeiro.

Pode-se observar que o modelo de computador Optiplex 9020 também apresentou os melhores tempos de inicialização, superando até o Optiplex 7040, que é o modelo mais recente dos três. Em relação à arquitetura baseada em ZFS, pode-se observar que nesta arquitetura baseada em *qcow2* todos os modelos apresentaram tempos de inicialização mais curtos, tendo a maioria do tempos de inicialização medidos entre 60 e 95 segundos.

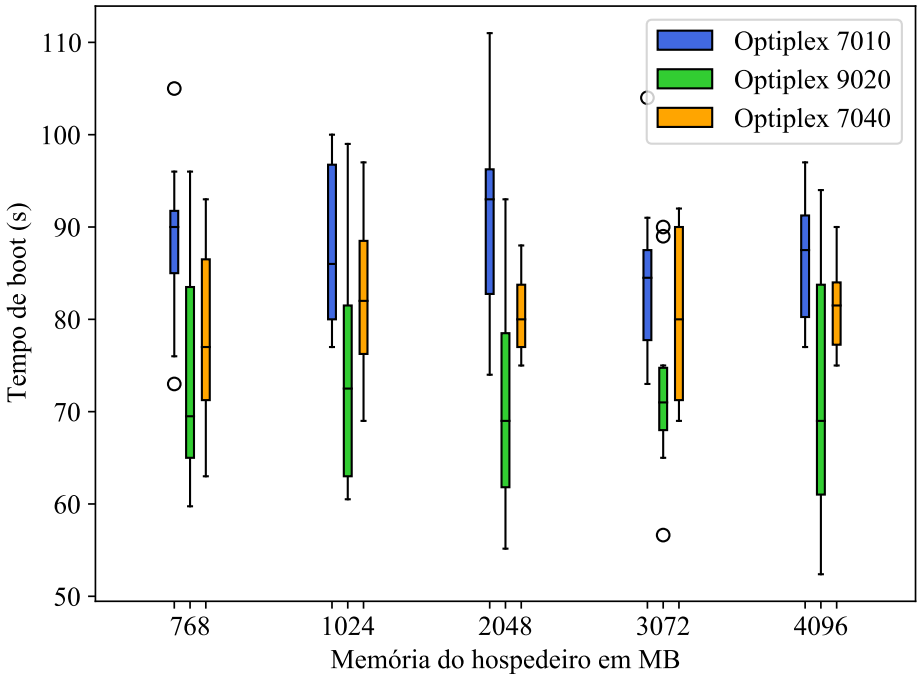
No pior caso, o tempo de inicialização ficou em torno de 110 segundos no modelo mais lento Optiplex 7010, enquanto que na arquitetura baseada em ZFS, no melhor caso, o tempo de inicialização ficou em torno 140 segundos com o modelo mais rápido Optiplex 9020. Esse fator também foi considerado para a escolha da arquitetura baseada em *qcow2* ser escolhida para ser implantada no sistema hospedeiro.

3.4 Conclusões

Este capítulo propôs dois modelos de virtualização: utilizando ZFS e arquivos *qcow2*. Também expôs os desafios encontrados para validar a abordagem de *passthrough* de dispositivos, objetivando alcançar desempenho na virtualização próximo ao do sistema quando está sendo executado diretamente no *hardware* sem sistema intermediário.

Os resultados dos *benchmarks* gráficos nos ambientes virtualizados demonstraram que

Figura 23 – Tempo de inicialização do sistema hospedeiro mais inicialização de um convidado Windows 10



Fonte: Produzido pelo autor

houve uma pequena perda de desempenho em relação aos ambientes não virtualizados, mas que não prejudicaram a usabilidade do sistema. As duas propostas de virtualização também demonstraram ser equivalentes para os três modelos de computador, exceto para o modelo Dell Optiplex 9020 que apresentou um resultado melhor com a solução baseada em ZFS.

Os resultados dos *benchmarks* de escrita e leitura mostraram que a arquitetura baseada em *qcow2* foi relativamente inferior em relação à arquitetura baseada em ZFS, apesar de instalada sobre unidades de estado sólido com maior velocidade para realizar gravações e leituras; porém apresentou estabilidade em todos os casos de teste, o que não ocorreu na arquitetura baseada em ZFS. A arquitetura baseada em *qcow2* também apresentou menores tempos de inicialização em todos os cenários propostos como já se previa.

Por esses motivos, foi dada preferência à arquitetura baseada em *qcow2* com o sistema de arquivos Ext4 para ser implantada no sistema hospedeiro dos computadores dos laboratórios. Como trabalhos futuros seria interessante analisar e comparar os resultados com outros virtualizadores, como XEN e Intel HAXM (*Hardware Accelerated Execution Manager*) (KRAJCI; CUMMINGS, 2013) e, além disso, aprimorar o modelo de disponibilidade dos discos virtuais dos usuários que, nas soluções apresentadas, são armazenados apenas no computador local.

Capítulo 4
GERÊNCIA

Este capítulo apresenta funcionalidades da solução proposta que contribuem para a gerência do sistema hospedeiro e do *hardware*.

4.1 Instalação automatizada

Instalar a solução em um novo computador envolve uma série de tarefas que seriam entediantes e sujeitas a erro caso sejam feitas manualmente. Portanto, foi construído um instalador que funciona totalmente sem intervenção humana após ser inicializado por meio de um *pendrive* conectado a uma porta USB.

O instalador executa as seguintes tarefas:

a) Assegura-se que foi inicializado em um modelo de máquina suportado.

b) Configura todas as opções do SETUP a partir de um arquivo.

c) Particiona e/ou formata o disco.

d) Copia uma imagem da raiz do sistema hospedeiro sobre uma partição ou *dataset*.

e) Instala um *bootloader*.

f) Gera pares de chave SSH persistentes para o sistema hospedeiro.

g) Gera um UUID aleatório persistente para o sistema hospedeiro.

h) Reinicia o computador.

Configurar as opções do SETUP é possível graças ao utilitário Dell CCTK, que é capaz de ler e alterar essas configurações de dentro de um sistema Linux. Padronizar essas opções é importante pois, para o funcionamento adequado da solução, a tecnologia Intel VT-d precisa estar habilitada, deve ser possível ligar remotamente a máquina (*Wake on LAN*), deve estar definida uma senha para acessar o SETUP, e um usuário só deve ser capaz de inicializar o sistema a partir de *pendrive* ou CD-ROM se souber essa senha.

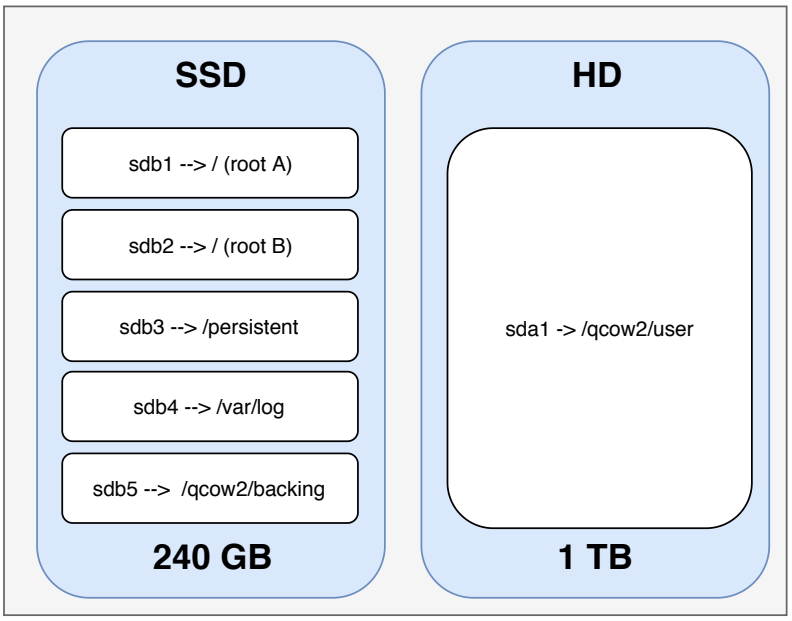
Para versões da solução baseadas no ZFS, o instalador formata o disco como um todo, inserindo-o em uma *pool*, e cria um *dataset* para a raiz, um para os arquivos persistentes (`/persistent`) e outro para *logs* (`/var/log`). O instalador usa o GRUB como *bootloader*, por este ser o único que suporta ser carregado a partir de um sistema de arquivos ZFS.

Para versões da solução baseadas em Ext4, o instalador cria duas partições para a raiz, mas utiliza somente a primeira delas. Além disso, cria uma partição para os arquivos persistentes (`/persistent`), uma para os *logs* (`/var/log`), uma para arquivos *qcow2* de imagens base (`/qcow2/backing`) e uma para arquivos *qcow2* de volumes de usuários (`/qcow2/user`). O instalador usa o Extlinux como *bootloader*, por ser uma opção mais simples que o GRUB e por facilitar o processo de atualização remota que será descrito na Seção 4.2.

Tanto `/persistent` como `/var/log` são colocados em *datasets* ou partições diferentes da raiz por se tratarem de diretórios contendo informações que devem persistir após uma atualização. A decisão de separar `/persistent` é devido ao fato de as chaves SSH e o UUID da máquina só precisarem ser gravados quando o instalador for executado, e depois pode ser sempre montado como somente leitura. Já `/var/log` precisa ser montado para escrita.

Uma característica interessante da solução baseada em Ext4 é que as partições podem ser alocadas em discos distintos. Por exemplo, a Figura 24 apresenta o esquema de particionamento adotado em um dos laboratórios de ensino da UFSCar. Nesse esquema, utiliza-se um disco SSD para armazenar a raiz e as imagens base, esperando-se promover inicialização rápida tanto do sistema hospedeiro como do sistema convidado. Por ser um meio de armazenamento mais barato e mais lento, o HD é utilizado somente para armazenar as alterações que os usuários tenham realizado sobre seus sistemas.

Figura 24 – Estrutura de partições utilizando SSD e HD



Fonte: Produzido pelo autor

Para permitir que uma mesma imagem da raiz do sistema hospedeiro funcione independente de como as partições estão distribuídas em um ou mais discos, o instalador formata as partições com um UUID padronizado, permitindo que elas possam ser referenciadas por meio de UUID no `/etc/fstab` do sistema hospedeiro.

4.2 Atualização remota do sistema hospedeiro

Desde o processo de instalação inicial, a solução proposta neste trabalho trata a raiz do sistema hospedeiro como uma imagem. Pode-se definir "imagem" como um sistema autocontido, imutável e facilmente replicável. Essas propriedades ajudam a controlar a complexidade do sistema e a manter sua consistência em um grande número de instalações.

Os benefícios de desenvolver e provisionar sistemas utilizando o paradigma de imagens são bastante conhecidos na comunidade de contêineres (MERKEL, 2014). No entanto, atualizar um sistema baseado em imagens que executa diretamente sobre o *hardware* físico (*bare metal*) requer cuidado pois, se ocorrer qualquer falha no processo de atualização, o sistema hospedeiro pode simplesmente parar de inicializar. Isso exigiria uma intervenção física para que o sistema fosse reinstalado, procedimento que este trabalho propõe-se a evitar ao máximo.

Portanto, atualizações da imagem do sistema hospedeiro precisam ser realizadas de forma atômica. Em outras palavras, é necessário garantir que a atualização ocorra de maneira completa ou que falhe por completo (mantendo a versão anterior), mas que nunca ocorra pela metade. As Subseções 4.2.1 e 4.2.2 discutem como esse objetivo pode ser alcançado, respectivamente, em um sistema baseado em ZFS e em um baseado em Ext4.

4.2.1 Atualização atômica de uma imagem ZFS

Alterar o *dataset* raiz enquanto o sistema encontra-se em execução pode ser problemático pois, por mais que a raiz seja montada como somente leitura, os programas passam a enxergar as novas versões dos arquivos logo após a atualização, o que pode causar inconsistências. Existem basicamente duas formas de tentar lidar com esse problema: (1) ter duas raízes diferentes e alternar entre elas a cada atualização; ou (2) aplicar a atualização durante a próxima inicialização.

Apesar da primeira abordagem ser bastante comum em celulares[1] e dispositivos embarcados em geral[2], não é trivial implementá-la com o ZFS devido à necessidade de usar o GRUB como *bootloader*. O componente do GRUB que decide a partir de qual sistema de arquivos as configurações e os módulos serão carregados é a *core image*, que é instalada em uma região do disco denominada *BIOS boot partition*. Dentro da *core image*, há uma *string* parecida com "`(,gpt1)/ROOT/alpineA@/boot/grub`", na qual `alpineA` poderia ser trocado por `alpineB` (e vice-versa) para alternar entre dois *datasets*. No entanto, não existe

[1] <https://source.android.com/devices/tech/ota/ab>
[2] <https://jumpnowtek.com/linux/An-upgrade-strategy-for-embedded-Linux-systems.html>

uma ferramenta pronta para realizar essa alteração de forma atômica e, para agravar o problema, a *core image* é armazenada em disco em formato compactado.

A segunda abordagem foi, portanto, considerada mais simples. A Figura 25 apresenta o trecho de código inserido no *script* `initramfs-init`, que executa antes do sistema de arquivos raiz ser montado.

Figura 25 – Trecho inserido no `initramfs-init` para atualizar a raiz do ZFS

```
if zfs list "${KOPT_zfsupgrade}" &>/dev/null; then
    echo "Please wait: system upgrade in process..."
    for snap in $(zfs list -Ht snap -o name "${KOPT_root#ZFS=}"); do
        zfs destroy "$snap"
    done
    if zfs send -ecw "${KOPT_zfsupgrade}" | \
       zfs recv -F "${KOPT_root#ZFS=}"; then
        zfs destroy -r "${KOPT_zfsupgrade}"
        reboot -f
    else
        echo "System upgrade failed."
    fi
fi
```

Fonte: Produzido pelo autor

A base para o funcionamento do *script* é a operação de cópia atômica, implementada pelo *pipe* da saída de `zfs send` para a entrada de `zfs recv -F`. Essa operação substitui completamente o *dataset* de destino ou, caso falhe ou seja interrompida, o mantém intacto.

Para obter confiança de que essa operação era realmente atômica, os desenvolvedores do ZFS on Linux foram consultados por meio do canal oficial de IRC do projeto. Além disso, foram realizados testes injetando falhas (que causavam *kernel panic*) em alguns pontos críticos do código do ZFS. Nenhum dos testes foi capaz de colocar o sistema em um estado no qual ele não pudesse mais inicializar.

É interessante notar que atualmente as ferramentas de espaço de usuário do ZFS não dão opção de sobrescrever o *dataset* de destino nas operações de `clone` ou de `rename`, caso contrário seria possível implementar a atualização de forma mais eficiente.

4.2.2 Atualização atômica de uma imagem Ext4

Como o Ext4 não possui o conceito de *datasets*, é difícil pensar em algum processo análogo ao que foi implementado para o ZFS. Por outro lado, o Ext4 é suportado pelo Extlinux (do projeto Syslinux[3]), viabilizando a implementação da abordagem inspirada em sistemas embarcados, que alterna entre duas partições raiz (A e B) a cada atualização.

[3] <https://syslinux.org>

O primeiro estágio do Extlinux reside na MBR, que fica no primeiro setor do disco. Esse estágio lê a tabela de partições do próprio disco e procura qual partição possui a *flag* de inicializável ativa. O primeiro estágio lê, então, o código do segundo estágio, que fica em um setor da partição denominado VBR, e salta para ele. Do segundo estágio em diante, a posição de todos os setores a serem lidos é calculada com relação ao início da partição. Isso significa que a partição pode ser colocada em qualquer parte do disco, que continuará funcionando normalmente.

O procedimento de atualização pode, portanto, ser implementado da seguinte forma:

a) Verificar qual das duas partições está atualmente montada como raiz.

b) Escrever a imagem da nova versão sobre a partição que não está montada.

c) Carregar a tabela de partições em memória.

d) Ligar a *flag* de inicializável da partição que não está montada, e desligar da que está.

e) Gravar a tabela de partições de volta no disco.

Supondo que as entradas da tabela correspondentes a ambas as partições raiz estejam no mesmo setor do disco, é possível gravar simultaneamente ambas as entradas de forma atômica. Como cada entrada de uma tabela no formato GPT tem 128 bytes, posicionar ambas as partições entre as quatro primeiras garante que suas entradas estejam no mesmo setor em praticamente qualquer disco atualmente disponível no mercado (setores têm pelo menos 512 bytes).

Antes de implementar uma ferramenta própria para manipular a tabela de partições, optou-se por estudar o código das ferramentas existentes, avaliando se conseguiriam operar de forma atômica. O sgdisk[4] foi estudado e pareceu funcionar de forma similar à proposta.

Por fim, o sgdisk foi avaliado experimentalmente. Instalou-se o Alpine em uma máquina virtual do VirtualBox. Para que as iterações do teste fossem mais rápidas, separou-se o `/boot` em sua própria partição em vez de incluí-lo na partição raiz. Portanto, em vez de criar duas partições raiz, foram criadas duas partições de *boot*. O *script* da Figura 26 foi colocado no diretório `/etc/local.d`, para executar ao final do processo de inicialização do Alpine. Em paralelo, o *script* da Figura 27 foi disparado fora da máquina virtual, acrescentando uma segunda fonte de reinicializações em momentos aleatórios. Após mais de 24 horas de teste contínuo, o sistema Alpine continuou inicializando normalmente.

[4] <https://github.com/samangh/gptfdisk>

Figura 26 – *Script* que alterna a partição inicializável e, em paralelo, causa reinicialização forçada em instantes aleatórios

```
#!/bin/sh
if [ `dd if=/dev/sda1 bs=512 count=1 2>/dev/null | \
sha256sum | awk '{print $1}'` = \
'076a27c79e5ace2a3d47f9dd2e83e4ff6ea8872b3c2218f66c92b89b55f36560' ];
then
    dd if=/root/bootpart.bin of=/dev/sda1 bs=100M count=1
    sh -c 'sleep $(( $RANDOM % 3)).$(($RANDOM % 10)); echo b >/proc/
       sysrq-trigger' &
    sync
    sgdisk /dev/sda -A "1:set:2" -A "2:clear:2"
    dd if=/dev/zero of=/dev/sda2 bs=100M count=1
else
    dd if=/root/bootpart.bin of=/dev/sda2 bs=100M count=1
    sh -c 'sleep $(( $RANDOM % 3)).$(($RANDOM % 10)); echo b >/proc/
       sysrq-trigger' &
    sync
    sgdisk /dev/sda -A "2:set:2" -A "1:clear:2"
    dd if=/dev/zero of=/dev/sda1 bs=100M count=1
fi
```

Fonte: Produzido pelo autor

Figura 27 – *Script* que reinicia uma máquina virtual em instantes aleatórios

```
while :; do
    echo reset;
    VBoxManage controlvm Alpine reset;
    sleep $((10 + $RANDOM % 60));
done
```

Fonte: Produzido pelo autor

4.3 Inventário e monitoramento

A tarefa de inventariar e monitorar laboratórios de computadores torna-se cada vez mais difícil conforme o número de computadores cresce. Por isso, ter alguma forma de automatizar essa tarefa é o que técnicos em geral buscam. Para isso, foi desenvolvido um *script* simples em *shell*, que é executado periodicamente pelo Cron e envia a saída de uma série de comandos para um servidor HTTPS, como na Figura 28. Procurou-se, assim, concentrar a maior parte da complexidade do lado do servidor, permitindo que o uso das informações seja refinado com mais facilidade ao longo do tempo.

Figura 28 – *Script* para coleta de *status* das máquinas

```
#!/bin/sh
function run() {
    echo "---- $@ ----"
    $@ 2>&1
}
{
    run cctk -o /dev/stdout
    run cat /proc/meminfo
    run df
    run cat /proc/net/dev
    run cat /proc/net/wireless
    run ip addr
    run ip route
    run ethtool eth0
} | curl -m 10 --data-binary @- \
         -H "Content-Type: text/plain" \
         -H "Auth: <segredo>" \
         -H "Uuid: $(cat /persistent/inventory/uuid)" \
         https://collector.example.com
```

Fonte: Produzido pelo autor

Algumas das informações coletadas pelo *script* são: espaço total e ocupado em disco, quantidade de discos, quantidade e uso de memória RAM, endereços IP, rotas, quantidade de dados trafegados na rede, *service tag* da Dell e uma *flag* que identifica se o gabinete da máquina foi aberto. Todas essas informações podem ser úteis para tomadas de decisão. O *script* de coleta é executado a cada 15 minutos ou quando houver alguma alteração nas conexões de rede.

Além das informações de inventário, é interessante ter acesso aos *logs* das máquinas de maneira centralizada, possibilitando a aplicação de filtros e a realização de buscas. Por esse motivo, foi escolhida a ferramenta Fluent Bit, que já vem sendo adotada por grandes empresas como AWS, Microsoft, Google Cloud entre outras (SILVA, 2021a). O Fluent Bit é uma ferramenta de código aberto e multiplataforma desenvolvida em linguagem C capaz de processar *logs* de variadas fontes, unificá-los e encaminhá-los para um ou mais servidores. O Fluent Bit foi criado com o objetivo de ser leve, com baixo uso de processador e memória, possibilitando sua instalação em sistemas embarcados (SILVA, 2021b).

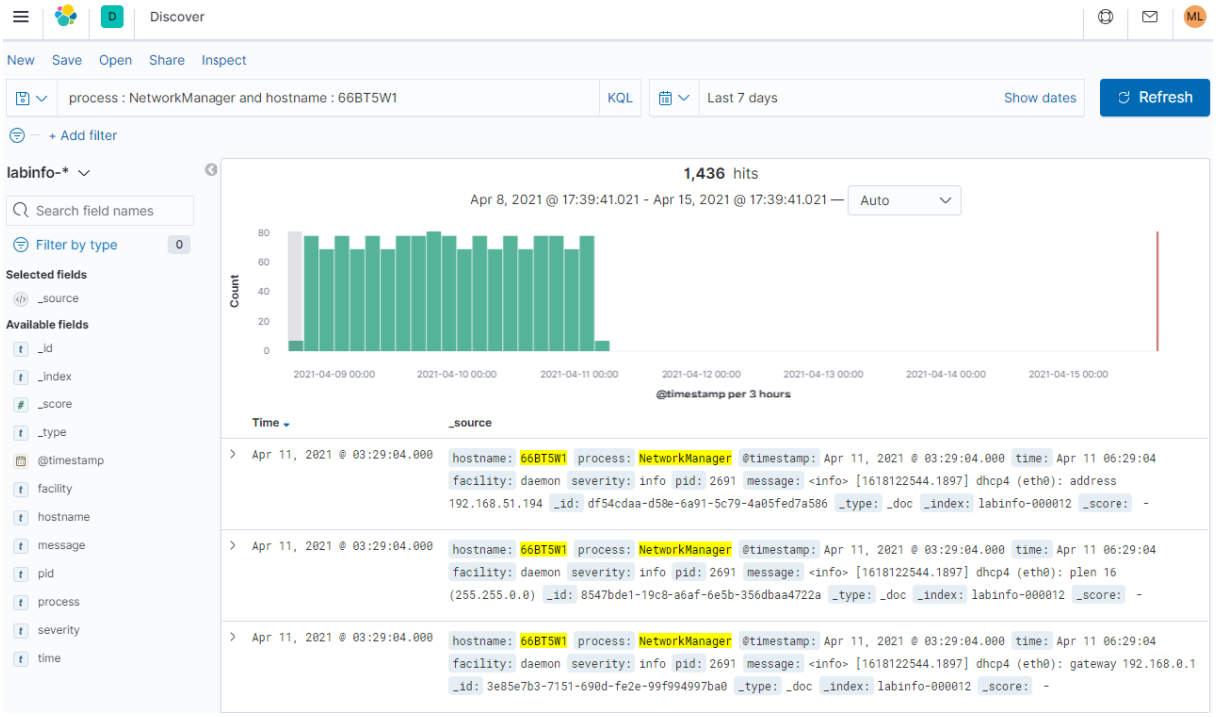
Apesar do sistema hospedeiro descrito neste trabalho não ser classificado como um sistema embarcado, ele possui limitações similares às de sistemas embarcados, pois repassa o máximo possível de recurso de memória, rede e tempo de processador para o sistema convidado. Sendo assim, executar o Fluent Bit no Alpine simultaneamente com o QEMU tende a interferir minimamente no desempenho do sistema em geral.

O Fluent Bit é configurado para encaminhar os *logs* para um servidor Elasticsearch.

Elasticsearch é um mecanismo de busca e análise de dados distribuído, gratuito e de código aberto, capaz de coletar dados de texto, numéricos, geoespaciais, estruturados ou não estruturados e exibi-los por meio do Kibana (GORMLEY; TONG, 2015). Kibana é uma ferramenta de visualização e gerenciamento de dados integrada ao Elasticsearch. Ela fornece opções para visualização de dados em histogramas, mapas, tabelas e gráficos em tempo real.

Na Figura 29 é exibido um exemplo de busca no Elasticsearch a partir do Kibana. Nesse exemplo, são fornecidos os parâmetros de processo, *hostname* e o período (últimos 7 dias). É possível visualizar os horários de envio desses dados e o padrão chave-valor das informações. Com isso, fica muito mais fácil diagnosticar problemas nos computadores dos laboratórios sem a necessidade de um técnico se deslocar até o local.

Figura 29 – Exemplo de pesquisa no Kibana de *logs* enviados ao Elasticsearch



Fonte: Produzido pelo autor

Um caso observado através de *logs* enviados para o Elasticsearch foi de um ataque de força bruta na tentativa de acesso a um dos computadores via SSH. Várias credenciais aleatórias estavam sendo inseridas por um IP externo desconhecido e em pouquíssimo tempo. Esse tipo de informação pode ser importante para aumentar a segurança da rede. O IP deste adversário poderia ser inserido em uma lista de bloqueio no *firewall* da instituição, por exemplo, e evitar novos ataques.

Existe um segundo servidor que faz a coleta de informações por meio de um *software* chamado Prometheus. Ele é um sistema de monitoramento de código aberto capaz de coletar dados de vários ambientes e auxiliar na identificação de eventos e visualização de informações. Além disso, ele é capaz de emitir alertas de acordo com a variação de parâmetros predefinidos

(TURNBULL, 2018). Por meio dele é possível obter métricas e dados como velocidade de interface de rede, dados de memória RAM, estatísticas de disco, quantidade de processadores, temperatura do processador, quantidade de *bytes* trafegados via interface de rede, entre outras.

Também é possível criar gráficos e observar o comportamento de um ou mais computadores em um determinado período. Na Figura 30, é exibido um gráfico de observação da transferência de uma imagem de disco para um conjunto de computadores. Como o Prometheus coleta em tempo real a quantidade de *bytes* trafegando pelas interfaces de rede dos computadores, utilizando esse parâmetro como medida é possível observar o volume de *bytes* trafegados e o período em isso ocorreu.

No gráfico criado, cada linha representa o volume de *bytes* recebidos pela interface de rede de um computador. O período especificado foi de uma hora e a verificação da média de *bytes* recebidos foi calculada de cinco em cinco minutos. Nesse caso, é exibido um exemplo de vários computadores com *download* com média abaixo de 2 MB/s. Esse é um exemplo no qual computadores apresentaram defeitos em seus cabos ou estavam conectados à rede sem fio com sinal de rádio fraco.

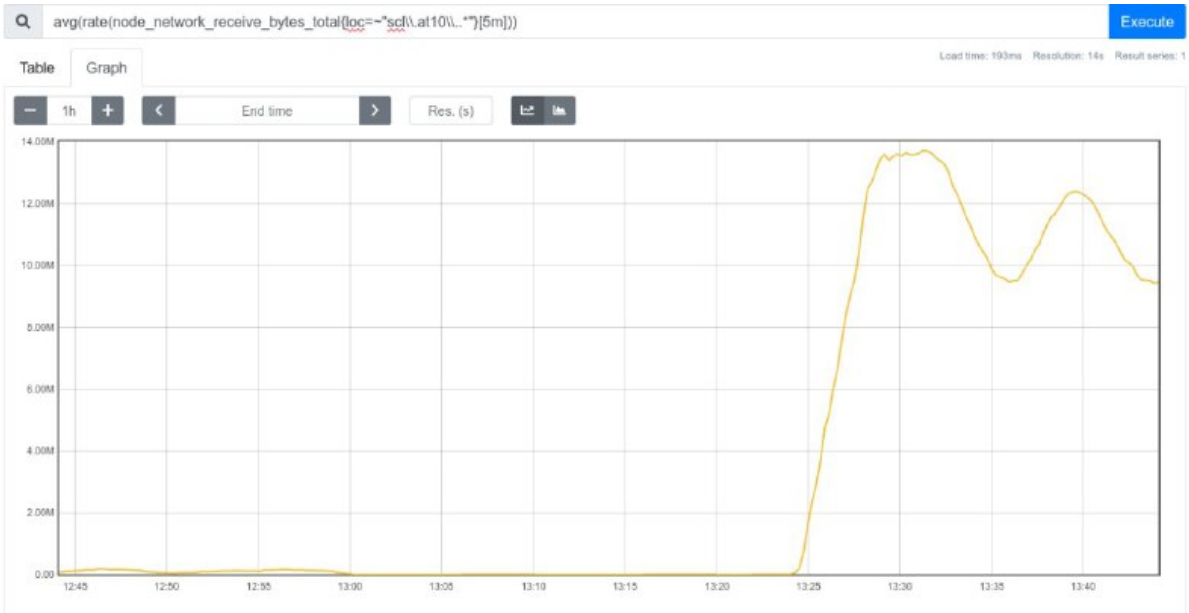
Figura 30 – Taxa de *bytes* recebidos por vários computadores no Prometheus

Fonte: Produzido pelo autor

Também é possível fazer filtros e criar gráficos com agrupamento de computadores por

meio dos parâmetros cadastrados, por exemplo, com o parâmetro de localização dos computadores. Na Figura 31 é exibido um gráfico com a soma de *bytes* recebidos pelos computadores do prédio chamado AT10. Mas poderia se fazer um filtro por campus, sala, computador e até por uma interface de rede específica somente alterando a *string* de busca. Essa funcionalidade é essencial para análises mais detalhadas como será visto na Seção 5.2.

Figura 31 – Filtro de exibição dos computadores do prédio AT10



Fonte: Produzido pelo autor

Esse tipo de monitoramento pode auxiliar os administradores dos laboratórios informatizados a visualizar essas informações de uma maneira mais simples e amigável. Além disso, os dados coletados podem ser usados para disparar alertas. Por exemplo, quando o gabinete de um computador for aberto; quando o consumo de dados estiver ocorrendo somente pela rede sem fio e não pela rede cabeada, mostrando a necessidade de manutenção; quando houver remoção de memória RAM ou remoção de disco; quando um processador apresentar altos níveis de temperatura por um longo período de tempo.

As informações também podem alimentar estatísticas. Por exemplo, pode-se verificar o consumo de disco ou quais máquinas ficam ligadas por mais tempo, para saber quais laboratórios estão sendo mais utilizados.

4.4 Criação de Imagens de Disco

A criação de imagens de disco personalizadas não é uma necessidade apenas dos laboratórios informatizados educacionais, mas também de outras instituições públicas e empresas, principalmente aquelas ligadas à computação em nuvem. Para os laboratórios, além da automatização do instalador do sistema hospedeiro, foi criado um *script* em *shell* para geração de imagem

do sistema convidado Arch Linux, sendo instalados os programas de uso comum da universidade atualmente. No *script* também são realizadas as configurações de ambiente e são habilitados os módulos Virtio para o disco e para a placa de rede, com intuito de melhorar a performance.

Já a imagem do convidado Windows 10 foi gerada manualmente, porém a instalação dos *softwares* foi automatizada. Os programas mais básicos foram instalados via gerenciador de pacotes Chocolatey, versão 0.10.15. Os programas de uso comum solicitados pelos professores para as suas aulas foram instalados via WPKG versão 1.3.1, um programa automatizado capaz de instalar, atualizar e remover programas do Windows via rede sem a necessidade de interferência de um técnico. Ele faz uso de instaladores .MSI, que permitem instalar programas com configurações previamente estabelecidas e em segundo plano. Um *script* de customização do Windows 10 divulgado em um repositório famoso do GitHub[5] foi utilizado para amenizar a utilização de memória RAM do Windows. Essa customização permitiu o consumo de memória do sistema em repouso de 2,7 GB para aproximadamente 1,6 GB.

Criou-se também um *script* para ocultar o adaptador de rede e o dispositivo de disco, pois como foram para-virtualizados, o sistema estava considerando como dispositivos ejetáveis. Apesar do dispositivo de disco não poder ser ejetado, o adaptador de rede poderia. Para evitar que um usuário ejetasse acidentalmente, o *script* foi configurado para ocultá-los já na inicialização.

4.5 Conclusões

Neste capítulo foram abordados os temas de instalação automatizada do sistema hospedeiro de ambas as soluções, tanto da arquitetura baseada em ZFS quanto da arquitetura baseada em Ext4 com *qcow2*. Também foram propostos modelos para permitir atualizações atômicas em ambas as arquiteturas. Testes foram feitos e demonstraram que, apesar de inserir erros e simular quedas de energia durante o processo de atualização, o sistema hospedeiro continuou funcionando.

Na seção de inventário e monitoramento, foi possível observar alguns exemplos de situações cotidianas com a ferramenta coletora de *logs* Fluent Bit e com as ferramentas de monitoramento Elasticsearch, Kibana e Prometheus; evidenciou-se o grau de importância do monitoramento constante para prevenção de incidentes e detecção rápida de problemas.

Pretende-se como trabalho futuro adaptar a solução baseada em Ext4 com *qcow2* para computadores que suportam apenas UEFI, pois a solução proposta foi construída para modelos que suportam BIOS. Também seria interessante automatizar o processo completo de criação de imagem e instalação de programas no Windows 10.

[5] https://github.com/Disassembler0/Win10-Initial-Setup-Script

Capítulo 5

DISTRIBUIÇÃO DE IMAGENS DE DISCO

Existem vários modelos de comunicação que podem ser usados para realizar transferência de dados. Geralmente aponta-se como mais tradicional o modelo de cliente/servidor, no qual o processo cliente solicita um serviço ou recurso do processo servidor que, por sua vez, fornece o serviço ou recurso (BIDGOLI, 2008). Seguindo apenas esse modelo, todas as máquinas buscariam a última versão de uma imagem de disco em um servidor, porém isso poderia gerar um gargalo dependendo do número de máquinas e do tamanho da atualização.

O modelo de distribuição com *IP Multicast* contribui para evitar esse gargalo, pois permite criar um grupo de computadores ao qual se atribui um mesmo endereço IP, distribuindo pacotes ou datagramas destinados a esse IP para todos os computadores do grupo (DEERING, 2008). Um fator limitante desse modelo é estabelecer a taxa de transmissão de pacotes em uma rede heterogênea pois, caso seja definida uma taxa muito alta, computadores com enlaces mais lentos não conseguem receber os pacotes, gerando a necessidade de retransmissão por parte do servidor e sobrecarga na rede. Transmitir em uma velocidade muito baixa acarreta em um maior tempo de envio do conteúdo e ociosidade de banda em computadores com enlaces mais rápidos.

Outra opção é o *BitTorrent*, que é um protocolo *peer-to-peer* (P2P) para transferência de arquivos grandes na Internet. Um usuário que utiliza o *BitTorrent* pode receber partes de um mesmo arquivo de vários usuários simultaneamente. Ao se comunicar dessa forma, diminui-se significativamente as requisições que seriam feitas a um único servidor, dado que nesse modelo todos os usuários assumem o papel de clientes e servidores ao mesmo tempo (COHEN, 2003).

No presente projeto, pretende-se enviar imagens de disco para vários laboratórios de modo eficiente a partir de um servidor. Para se alcançar este objetivo, alguns modelos de transmissão de arquivos e distribuição de conteúdo foram analisados e são discutidos na Seção 5.1.

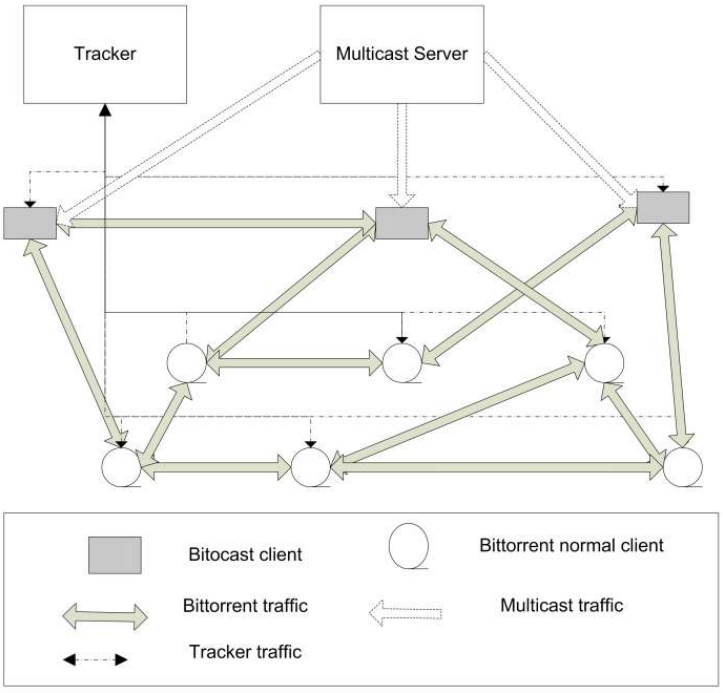
5.1 Revisão da Literatura

5.1.1 Bitocast

Silva (2009) apresenta uma solução híbrida que utiliza simultaneamente BitTorrent e IP Multicast para distribuir conteúdo. O principal objetivo é reduzir o uso de rede de provedores de Internet (ISPs), reduzindo o tempo de *download* e o consumo de banda de *upload*.

O autor propõe o uso de um servidor Multicast que transmite dados diretamente para os grupos de IP, um *tracker* responsável por transmitir aos *peers* que o contactam uma lista aleatória de nós da rede que possuem o conteúdo buscado, e clientes Bitocast que realizam o *download* do conteúdo. No cliente Bitocast, são inseridas as informações do servidor Multicast e o cliente pode escolher a proporção de banda que cada protocolo pode usar. Para efeito de simulação, a banda de todos os clientes foi dividida em 50% para cada protocolo. A Figura 32 apresenta a estrutura da solução.

Figura 32 – Bitocast



Fonte: Silva (2009)

No servidor Multicast, o autor disponibilizou grupos de IP Multicast distintos com variadas taxas de transmissão. Assim, os clientes Bitocast podem fazer *download* no grupo mais próximo da sua capacidade de recepção. Esse modelo acelera o processo de *download* dos *peers* e reduz o descarte de pacotes na rede.

As simulações foram baseadas no cenário de *flash crowd*, muito comum em atualizações de sistemas distribuídos. O autor fez simulações utilizando BTSim[1] e modificou os clientes

[1] <https://research.microsoft.com/projects/btsim>

para receberem pacotes do servidor Multicast enquanto simultaneamente recebem por meio do BitTorrent. Nas simulações, distribuiu-se um arquivo de 100 MB por meio de 5 servidores Multicast transmitindo blocos em *Round Robin*. As taxas de transmissão foram de 128, 512, 1024, 2048 e 4096 kbps. A probabilidade dos nós que se tornam *seeders* durante a simulação saírem da rede após terminar o *download* foi definida como 50%. A largura de banda de cada nó foi distribuída de acordo com a Tabela 4.

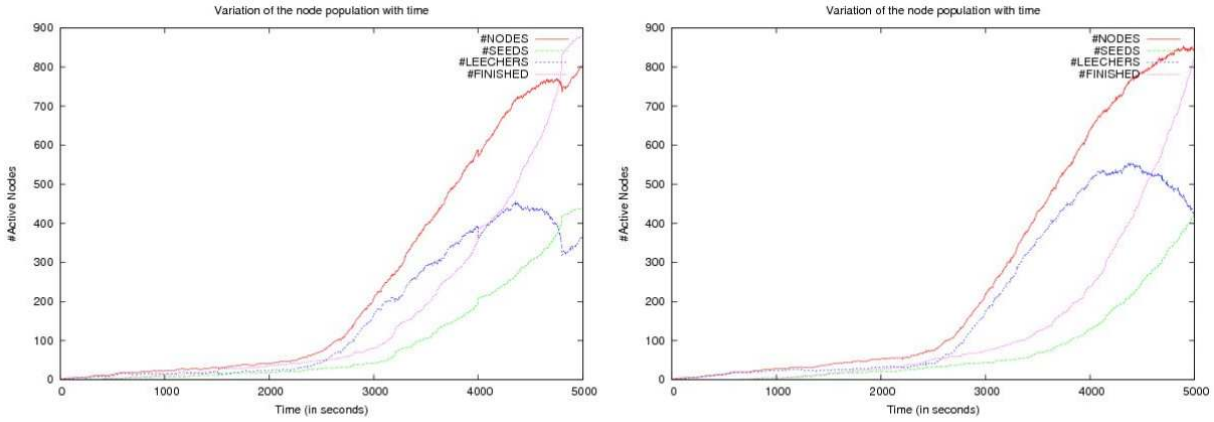
Tabela 4 – Largura de banda dos nós

Download (kbps)	Upload (kbps)	Percentual do total de nós
512	256	5%
1024	254	25%
2048	512	25%
4096	1024	40%
8192	2048	5%

Fonte: Silva (2009)

Na Figura 33, o gráfico do lado esquerdo representa os resultados obtidos com o Bitocast e o do lado direito representa os resultados obtidos com o BitTorrent. É possível visualizar que o Bitocast apresentou resultados melhores que o BitTorrent. Com o crescimento da rede, os nós Bitocast tornam-se *seeders* com mais rapidez do que na rede BitTorrent pura. Além disso, o número de *leechers* na rede com Bitocast cresce mais vagarosamente do que na rede BitTorrent. Isso deve-se ao fato que os nós do Bitocast recebem blocos diretamente do servidor Multicast.

Figura 33 – *Nodes across time (Bitocast vs BitTorrent)*



Fonte: Silva (2009)

Para reduzir ainda mais o tráfego de pacotes na rede, o autor modificou o modelo para um modo cooperativo, no qual os clientes Bitocast tinham noção de quais blocos o servidor Multicast iria transmitir nos próximos ciclos. Assim, evitou-se que os clientes Bitocast solicitassem aos seus vizinhos blocos que estavam para chegar via Multicast.

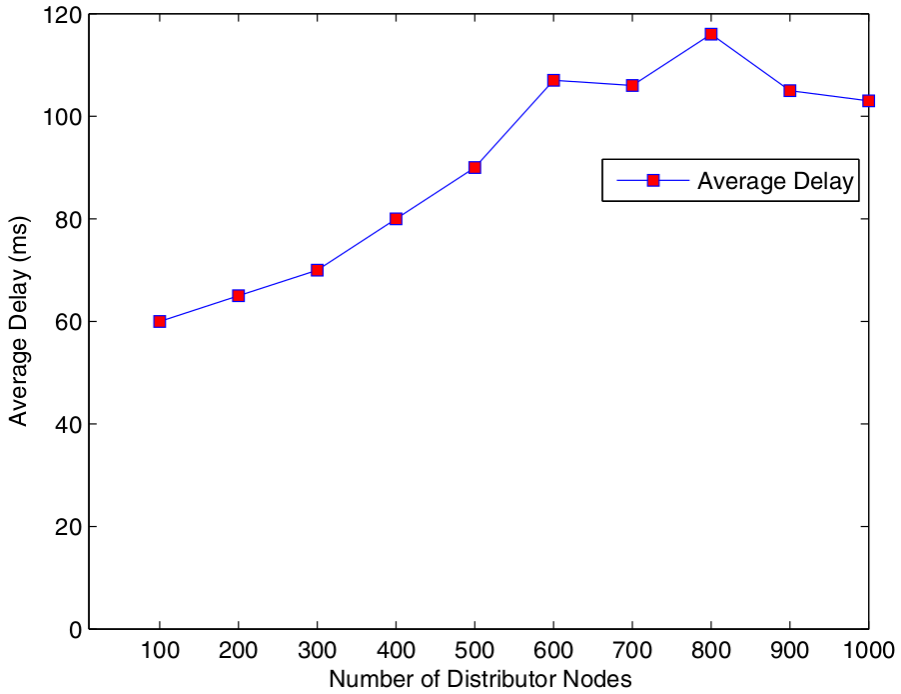
Existem algumas limitações na abordagem da proposta, como o fato do servidor IP Multicast precisar cadastrar previamente os nós em cada um dos grupos de IP Multicast. Assim, é necessário ter uma noção da banda disponível no enlace de cada um dos clientes Bitocast para realizar essa associação. O autor também menciona que como projeto futuro seria importante implantar o Bitocast em um cenário real, já que o Bitocast foi apenas simulado.

5.1.2 *A self-adaptive ALM architecture for P2P media streaming*

Amad et al. (2015) propõem uma arquitetura de *Application Layer Multicast* (ALM) para ser utilizada sobre uma rede P2P para transmissão de mídia. Essa arquitetura seleciona automaticamente a topologia da rede com base nos parâmetros de avaliação da rede como: número de nós, densidade da rede, dinamicidade dos nós e existência de potenciais nós distribuidores. A topologia pode ser centralizada ou descentralizada, ou seja, com um nó distribuidor ou com distribuidores intermediários.

Dado um número de nós maior que 20, o nó central seleciona os nós com menor *delay* para também serem distribuidores, gerando assim a árvore de distribuição. Nas simulações que os autores fizeram, a arquitetura proposta apresentou resultados positivos em relação à diminuição de sobrecarga nos servidores e baixo *delay* na transmissão de mídia, como pode-se observar na Figura 34.

Figura 34 – *Average Delay Vs Network Size*



Fonte: Amad et al. (2015)

Os autores utilizaram a ferramenta PeerSim[2] para simular a rede. O atraso entre nós

[2] <https://pdos.csail.mit.edu/archive/p2psim>

vizinhos foi gerado aleatoriamente entre 0 e 150 milissegundos, foram criados entre 10 e 50 nós distribuidores aleatoriamente e cada nó distribuidor poderia ter entre 0 e 50 nós filhos conectados diretamente a ele. Na Figura 34 é possível observar que a média de *delay* entre o nó servidor e os nós filhos ficou abaixo dos 120 milissegundos, mesmo com 800 a 1000 nós existentes na rede. Um *delay* de 150 milissegundos já seria um *delay* aceitável para aplicações sensíveis a atrasos em redes P2P (MARFIA et al., 2007). Entretanto, os autores não mencionam uma explicação para a queda na média de *delay* para uma rede acima de 800 nós.

Apesar dos bons resultados, um dos problemas mencionados pelos próprios autores é o fato de que a cada entrada ou saída de um nó o algoritmo analisa se deve centralizar ou descentralizar a estrutura. Então, caso a rede seja muito dinâmica, isso pode causar um grande impacto no desempenho da transmissão. Outro fator importante é a questão da densidade da rede. No caso de haver poucos nós candidatos a nó distribuidor, a rede tenderá a sobrecarregar os servidores.

5.1.3 *One-to-many file transfers using Multipath-Multicast with coding at source*

Ogawa et al. (2016) propõem um algoritmo para transferência de grandes arquivos a partir de um único transmissor para múltiplos receptores sobre uma rede *OpenFlow* utilizando *Multipath-Multicast*. O número de fases do algoritmo é igual ao número de nós que receberão o arquivo.

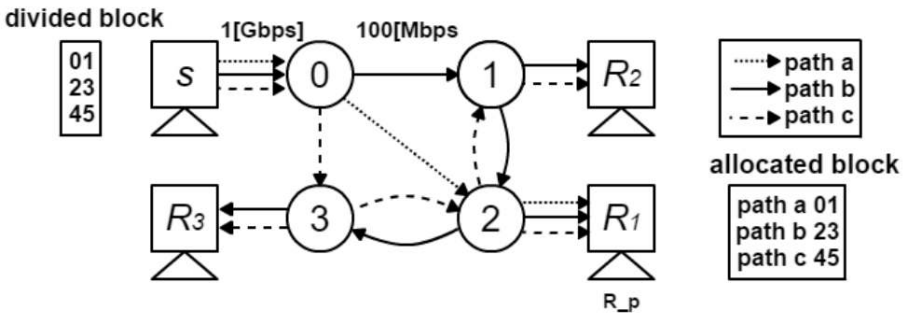
No início, o algoritmo seleciona o nó mais distante do servidor. O arquivo a ser transferido é dividido em vários blocos de mesmo tamanho que são enviados sequencialmente por todas rotas possíveis ao nó receptor escolhido. Os blocos transmitidos pelo servidor também são entregues aos outros receptores existentes no caminho. A segunda fase inicia quando o primeiro receptor houver recebido todos os blocos. Então, um segundo receptor é eleito para receber os outros blocos que faltam. As Figuras 35, 36 e 37 exemplificam esse caso.

O artigo propõe também um modelo *coded-MPMC* que aprimora o modelo anterior fazendo a combinação de dois blocos para enviar para mais de um receptor. Por exemplo, considere que um receptor possui o bloco A e não possui o bloco B e, simultaneamente, outro receptor possui o bloco B e não possui o bloco A. Assim, faz-se XOR entre o bloco A e o B e envia-se o bloco codificado para os dois receptores. Quando o bloco é entregue, ambos os nós receptores aplicam XOR com o bloco que possuem e obtêm o bloco que lhes falta.

Os autores realizaram 1000 execuções para cada algoritmo e extraíram a média do tempo para concluir a transferência de um arquivo de 100 MB entre 42 receptores e um transmissor em uma rede com topologia tipo RENATER[3]. Todos os enlaces entre os nós eram de 155 Mbps e sem perda de conexão entre eles. Na Figura 38, K representa a lista dos receptores primários (o

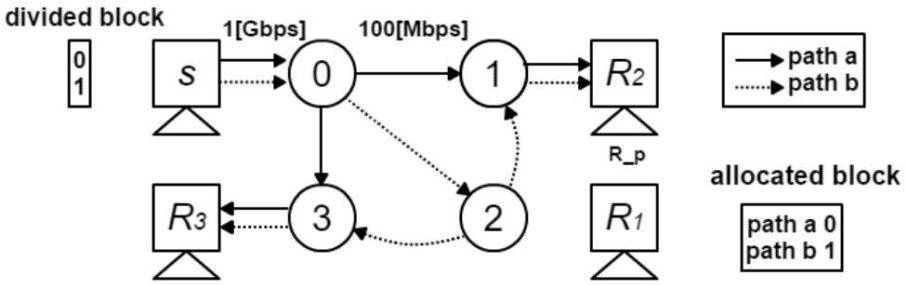
[3] <http://www.topology-zoo.org/dataset.html>

Figura 35 – ***Phase 1 in the MPMC scheme***



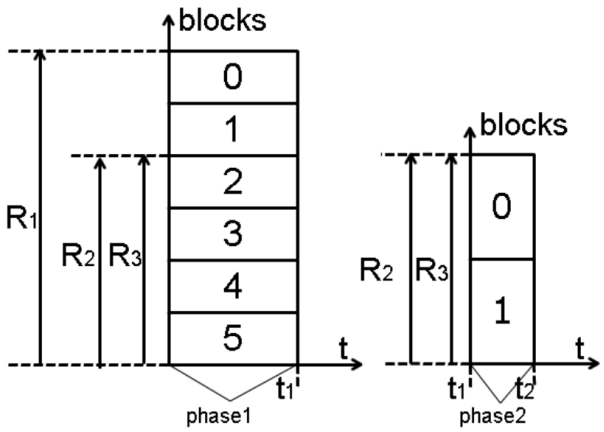
Fonte: Ogawa et al. (2016)

Figura 36 – ***Phase 2 in the MPMC scheme***



Fonte: Ogawa et al. (2016)

Figura 37 – ***Schedule of the MPMC scheme***

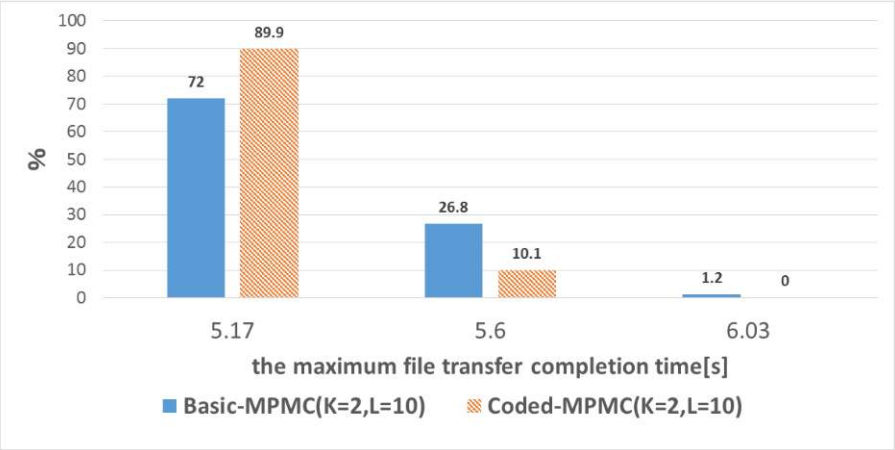


Fonte: Ogawa et al. (2016)

atual nó receptor dos blocos ou os nós que já receberam todos os blocos) e L representa a lista de receptores secundários selecionados (os L primeiros nós que estão na fila para receber os blocos).

É possível perceber que o modelo *coded-MPMC* obteve melhores resultados que o *based-MPMC*, pois reduziu o número de blocos enviados e o tempo de conclusão de transferência do arquivo para toda a rede. Nessas condições, 5,17 segundos seriam o suficiente para transmitir o arquivo para um nó, mas com o *coded-MPMC* o algoritmo conseguiu transferir nesse tempo praticamente para toda a rede, com 89,9% dos receptores tendo completado o *download*.

Figura 38 – *Distribution of the maximum completion time in the limited search case (K=2, L=10)*



Fonte: Ogawa et al. (2016)

Para escolher os blocos que serão enviados, o algoritmo analisa na fila de receptores quais blocos podem ser combinados de acordo com os que faltam para cada um deles. Como essa análise pode ser muito demorada, os autores fixaram limites de verificação de combinações para não retardar a transferência dos blocos. No caso de não haver uma combinação possível na análise, o bloco é enviado sem codificação XOR.

Existem alguns fatores que não favorecem essa arquitetura: o fato do modelo se enquadrar apenas às redes *OpenFlow* e a falta de tratamento para perda de conexão não se encaixa com a realidade de muitas redes. Também é importante considerar se uma das rotas tiver um caminho muito lento, pois isso pode retardar o processo de distribuição em toda a rede. Seria mais adequado enviar mais blocos na rota que possui um caminho mais rápido, por exemplo, e não distribuir igualmente os blocos para cada rota.

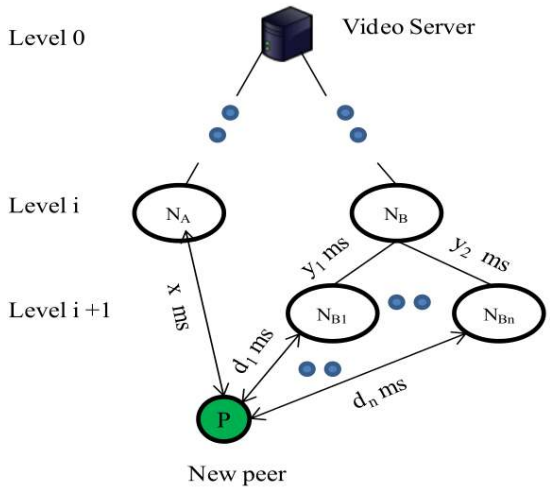
5.1.4 *An Improved Delay-Resistant and Reliable Hybrid Overlay for Peer-to-Peer Video Streaming in Wired and Wireless*

Maheswari e Ramesh (2018) propõem uma solução para distribuição de vídeos com alta resolução, tanto para dispositivos conectados à rede cabeada quanto à rede sem fio. O algoritmo proposto é composto de dois estágios. No primeiro estágio ocorre o processo de construção da árvore que se sobrepõe à rede P2P. No segundo estágio, a árvore muda para uma estrutura híbrida com uma árvore e *clusters* de nós. Ela se forma com base na ordem de chegada dos nós, da localização e da reputação de cada um deles. A reputação deles leva em consideração o número de filhos que o nó pode suportar, o tempo que o nó existe na rede e a distância do nó ao servidor.

No primeiro estágio, quando um nó deseja conectar-se à transmissão de vídeo, ele envia ao servidor a sua requisição junto com os detalhes da sua reputação. O servidor retorna o endereço com quem ele deve se conectar. Os nós com as melhores reputações e mais próximos do servidor de vídeo são escolhidos como nós de referência e formam o primeiro nível vinculado ao servidor, chamado de G1. Um nó com reputação inferior é ligado ao nó de referência mais próximo dele, desde que não exista outro caminho com menos latência até ele.

O algoritmo procura manter a profundidade da árvore nivelada em todos os ramos, porém há uma exceção: na Figura 39, um novo nó P chega e o nó referência mais próximo N_B não pode recebê-lo, porém há uma posição distante na árvore com N_A. Nesse caso, o algoritmo calcula o menor *delay* entre P e os filhos de N_B de acordo com a fórmula: $z = min(\ y_i + d_i\ )$. Sendo x o *delay* entre N_A e P, se $x < z$, então P é conectado ao N_A, senão P é conectado a um dos filhos de N_B.

Figura 39 – ***Arrival Time and Location-Based - (ATLB-A2) joining procedure***



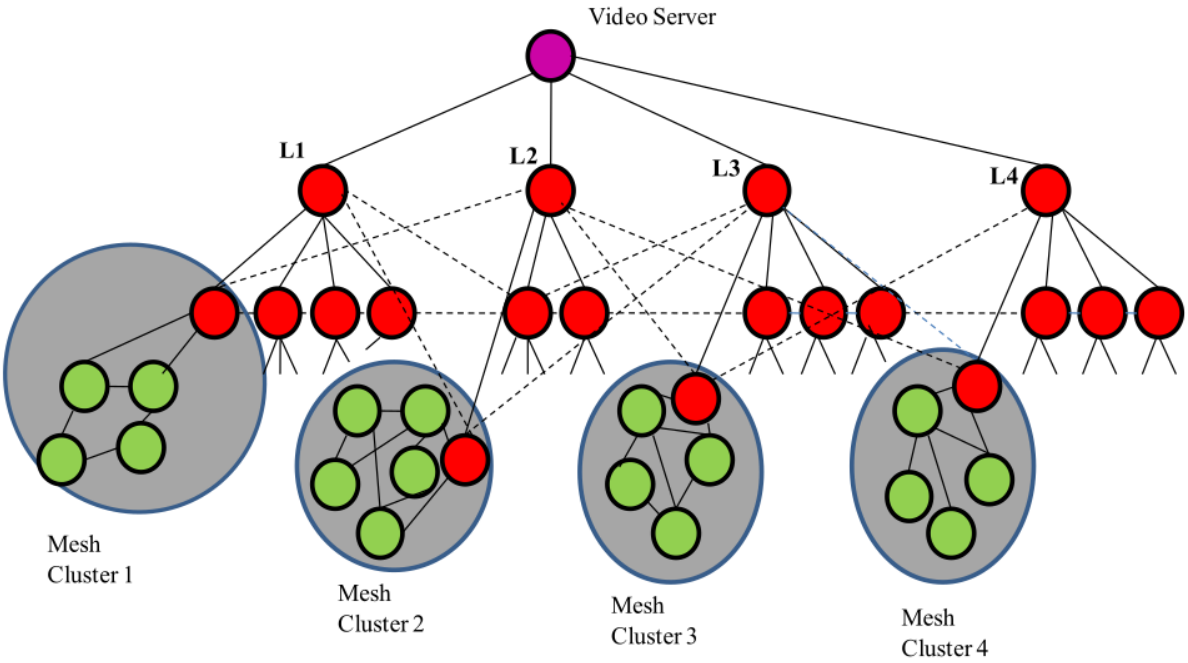
Fonte: Adaptado de Maheswari e Ramesh (2018)

No segundo estágio, quando todos os nós já foram anexados, a árvore é transformada para que os nós com maior capacidade de distribuição fiquem mais próximos do servidor: estes são os nós com melhor reputação. Os nós com baixa reputação tornam-se nós folhas e formam

os *clusters*. Cada nó tem conexões auxiliares com dois outros nós do mesmo nível e três de um nível superior; assim, caso a conexão principal falhe, ele terá quatro outras conexões alternativas. Na Figura 40 é exibida a estrutura final da proposta.

A proposta do artigo se encaixa bem em um contexto no qual é conhecida a quantidade de nós que irão conectar-se à rede no momento da transmissão. No caso da proposta deste trabalho, poderia ser uma abordagem a ser considerada, desde que em toda atualização do parque computacional todas as máquinas fossem ligadas simultaneamente. Caso contrário, será necessário estabelecer outro critério para definir o momento de transição do primeiro estágio para o segundo estágio.

Figura 40 – *ATLB-A2 tree and mesh cluster overlay*



Fonte: Adaptado de Maheswari e Ramesh (2018)

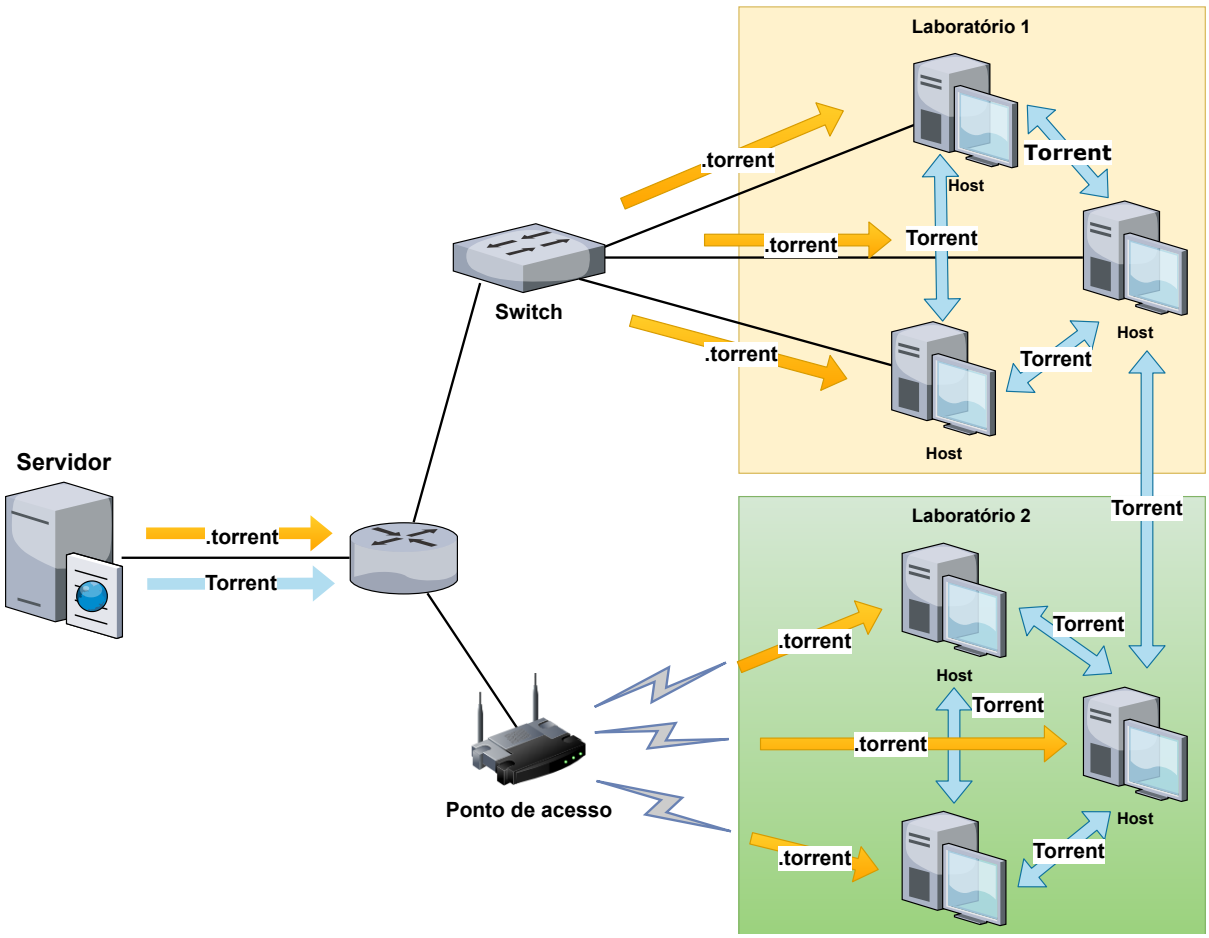
É possível observar que em todos os trabalhos o *BitTorrent* continua sendo referência para distribuição de arquivos. Por essa razão, optou-se por se fazer uso do protocolo visando descentralizar a distribuição. O *IP Multicast* não será abordado neste momento devido à não garantia de entrega de pacotes. Entretanto, futuramente pretende-se combinar os dois em um único protocolo, aproveitando o melhor de cada um deles para realizar a distribuição das imagens de disco.

5.2 Modelo de Distribuição

Após completar a instalação do sistema Alpine Linux com a solução baseada em *qcow2* em todos os computadores dos laboratórios e, além disso, concluir a preparação das imagens de disco, a distribuição pode ser executada. A Figura 41 ilustra de maneira geral o cenário

da distribuição conduzido nas salas de aula informatizadas mantidas pela Secretaria Geral de Informática da UFSCar. A distribuição foi executada utilizando duzentos e cinquenta e dois computadores distribuídos em nove laboratórios.

Figura 41 – Cenário de distribuição



Fonte: Produzido pelo autor

O processo de distribuição ocorre da seguinte forma:

1. Primeiramente um arquivo .torrent é gerado para a imagem de disco já pronta. Esse arquivo contém informações do arquivo, tamanho, *hashes* dos pedaços e a URL do *Tracker* (COHEN, 2003). O *Tracker* é responsável por informar aos *peers* quais nós da rede possuem a imagem de disco. Neste caso, o servidor será o único com a imagem no início da distribuição. Foi utilizada a ferramenta *mktorrent* para geração do arquivo .torrent.

2. Após a geração do arquivo, os computadores dos laboratórios configurados com a função *Wake on LAN* habilitada são ligados remotamente por meio do comando `etherwake -i eth0 -b`.

3. Em seguida, esse arquivo .torrent é enviado para todos os computadores.

4. Na sequência é iniciada a transferência da imagem em todos os computadores. Inicialmente, há um gargalo na distribuição por existir apenas um nó com a imagem, mas em pouco tempo, novos nós tornam-se *seeders* da imagem.

5. Após a finalização da transferência, os discos virtuais dos alunos são removidos dos computadores e a imagem base antiga também. Por isso, nenhum usuário pode estar com uma máquina virtual aberta, caso contrário, o seu disco virtual local ficará inconsistente com a imagem base e não funcionará mais.

6. Para finalizar, um arquivo no formato *INI* é criado em cada computador com as informações da nova imagem disponível.

5.3 Cenários Monitorados

No decorrer do processo foi possível monitorar a distribuição das imagens por meio das ferramentas de monitoramento incluídas neste projeto e alguns eventos foram destacados. Por exemplo, na Figura 42 é possível visualizar um conjunto de computadores conectados à rede, fazendo *download* da imagem com taxas em torno de 2 MB/s ou menos. Esse prédio, chamado AT9, estava com *uplink* de apenas 100 Mbps.

Analisando a distribuição, notou-se que os computadores estavam conectados a muitos *peers* externos ao prédio AT9, saturando o *uplink*, sendo que poderiam fazer *download* de pedaços do arquivo dos computadores do próprio laboratório, já que o *swtich* no qual eles estavam conectados possuía enlace de 1 Gbps. Para resolver esse problema, alguns *seeders* externos ao prédio foram desligados, subindo a taxa de *download* da maioria deles para 9 MB/s.

Esse fato foi importante para se analisar melhor um parâmetro do Aria2c chamado `seed-ratio`. Na distribuição, ele foi configurado com valor igual a *0.0*. Esse parâmetro significa que os *peers* que terminarem o *download* irão semear o arquivo para os outros até que o programa seja encerrado. Como os laboratórios de outros prédios terminaram o *download* antes dos computadores do AT9, eles tornaram-se *seeders* primeiro e os computadores do AT9 adicionaram *peers* externos para a sua lista de *peers* principais para solicitar novos pedaços do arquivo. Ao se remover alguns *peers seeders* externos ao AT9 a taxa de *bytes* recebidos aumentou.

Ainda analisando os gráficos do AT9, na Figura 43 é possível visualizar um conjunto de computadores conectados à rede pelo cabo fazendo *download* da imagem com taxas em torno de 8 MB/s. Após o término do *download*, um único computador que estava conectado pela rede sem fio começou a aumentar sua taxa de *download*, passando de aproximadamente 0,4 MB para um pouco mais de 8 MB. Como ela estava conectada pela rede sem fio, não conseguia se

Figura 42 – Taxa de *bytes* recebidos por vários computadores no Prometheus

Fonte: Produzido pelo autor

comunicar com as demais conectadas pela rede cabeada, por estar em outro segmento de rede naquele momento. Ou seja, ela só conseguia baixar do servidor.

Analisando a rede, notou-se que o gargalo estava ocorrendo porque ela compartilhava o mesmo *uplink* das demais. Quando os computadores da rede cabeada terminaram seus *downloads*, o *uplink* ficou com banda disponível.

Na Figura 44 é exibido um teste de distribuição da imagem do Windows 10 com todos os computadores. A linha apresentada no gráfico é a soma da quantidade de *bytes* baixados por todos. A taxa total entre onze horas e dez minutos e onze horas e quinze minutos chegou a alcançar 2,7 Gigabytes por segundo, ou 21,6 Gigabits por segundo.

Na Figura 45 é possível observar a quantidade de dados trafegados na rede em um *switch* específico durante uma das transferências realizadas. Fica claro no final do gráfico que houve um volume muito alto de tráfego nesse período, o que não parecia ser habitual neste segmento de rede. Isso é importante, pois com este método de distribuição é possível aproveitar toda a largura de banda disponível e, consequentemente, realizar testes de *stress* na rede.

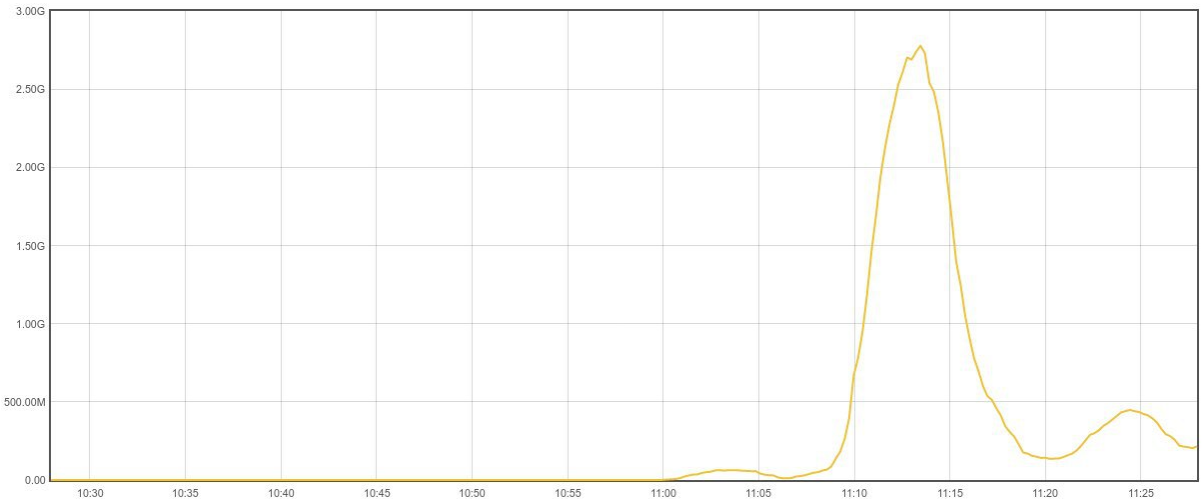
É importante ressaltar que esse modelo de transferência trouxe grandes benefícios para universidade como:

a) Automatização de grande parte do processo de instalação e clonagem de imagens, que antes era feito manualmente e em um laboratório por vez;

Figura 43 – Computador em segmento de rede diferente exibido no Prometheus

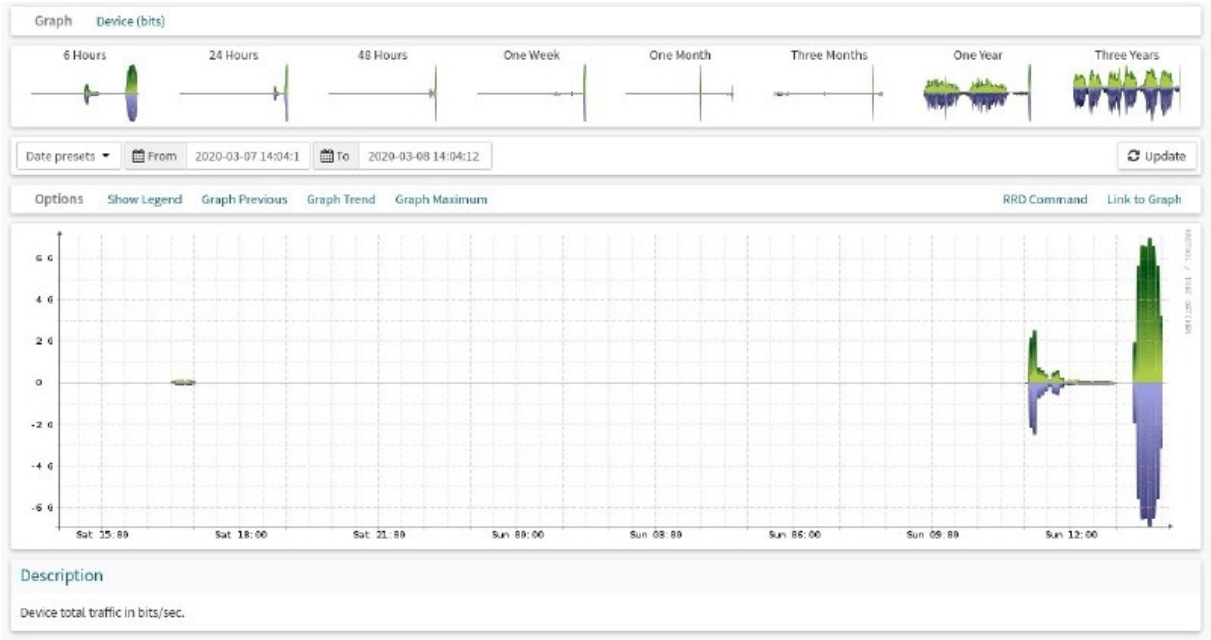
Fonte: Produzido pelo autor

Figura 44 – Taxa de *bytes* trafegados na rede dos laboratórios



Fonte: Produzido pelo autor

Figura 45 – Tráfego observado em um *switch* durante a distribuição de imagens



Fonte: Produzido pelo autor

b) Redução do tempo de atualização dos laboratórios de semanas para minutos;

c) Menor custo e economia de tempo com deslocamento de técnicos (já que o Campus da UFSCar é relativamente extenso e os laboratórios não estão localizados em um único prédio);

d) Garantia de integridade das imagens;

e) Descobrimento de gargalos na rede por configurações restritivas de banda, problemas em dispositivos ou por falta de sinal de rádio;

f) Visão geral do conteúdo de *softwares* que estão sendo disponibilizados aos usuários em todos os laboratórios.

5.4 Desafios Enfrentados

Em um ambiente simulado, muitas situações inesperadas não são levadas em consideração por não serem previstas durante o planejamento da ferramenta, ou devido às limitações do próprio simulador e, principalmente, por ser impossível representar todas as falhas que podem ocorrer no mundo real. Para efetuar a distribuição das imagens para todos os laboratórios pela primeira vez, alguns problemas foram identificados e que não estavam previstos. Outros porém já eram esperados e por meio do monitoramento foi possível identificá-los mais facilmente. Após a identificação destes, as manutenções foram feitas e garantiram que a distribuição ocorresse com sucesso.

No momento de ligar todos os computadores remotamente, por exemplo, alguns não ligaram. Ao se dirigir ao local, notou-se que alguns deles estavam com o cabo de energia ou de rede desconectado. Outros, tinham ligado, porém o processo de inicialização tinha sido interrompido. Analisando melhor o problema, foi identificado que a bateria de BIOS estava fraca e precisava ser trocada. Ao efetuar essa correção, o problema foi sanado.

Pela central de monitoramento verificou-se que alguns computadores possuíam 4 GB de RAM, sendo que todos os modelos deveriam ter 8 GB por padrão de fábrica. Isso não afetou a transferência das imagens, mas com certeza afetaria o desempenho da virtualização. Novos componentes tiveram que ser comprados para recompor esses modelos.

Em um dos computadores não foi possível efetuar a atualização de BIOS. Esse fator impossibilitaria realizar a virtualização dos convidados com passagem de periféricos, por isso, a máquina foi substituída. Outro modelo possuía uma placa de vídeo externa e não se adequava ao *script* de passagem de periféricos. Por isso, optou-se também por substituí-lo por outro com o *hardware* padrão.

Durante a distribuição de imagens, identificou-se que alguns computadores apresentaram lentidão demasiada para efetuar o *download*. Ao analisá-los mais cuidadosamente pelas ferramentas de monitoramento, notou-se que eles estavam recebendo os dados em baixa velocidade, bem menor do que as placas de rede desses modelos suportam. Ao checar no local, alguns estavam com o cabo de rede danificado ou o ponto de rede no qual eles estavam conectados estava com problema. Em um deles, a porta do *switch* estava com problema, o que foi solucionado trocando-se o cabo de porta. Após as correções, todas conseguiram baixar mais rapidamente.

Todos os problemas ressaltados nesta seção foram vivenciados devido aos testes serem executados em um ambiente real. Também vale destacar o fato de que o modelo proposto trouxe grandes vantagens para a universidade e aos administradores dos laboratórios, como maior facilidade de identificação problemas que poderiam prejudicar as aulas e a possibilidade de realizar correções pontuais com antecedência.

5.5 Conclusões

Neste capítulo foi apresentado um modelo de distribuição de imagens de disco com *BitTorrent*. Alguns testes de distribuição foram realizados com nove laboratórios localizados em diferentes prédios do Campus da UFSCar de São Carlos. Vale ressaltar que a distribuição foi realizada tanto pela rede cabeada quanto pela rede sem fio, apresentando alternativas de gerenciamento.

Apesar da heterogeneidade da rede, o modelo de distribuição mostrou-se eficiente. Os problemas encontrados não impediram que as transmissões das imagens fossem concluídas com sucesso, pois o protocolo *BitTorrent* é capaz de adaptar-se às redes desse tipo.

Foi possível identificar e sanar vários problemas pontuais que poderiam prejudicar a distribuição, graças às ferramentas de monitoramento adotadas. Depois de feitas as correções, o processo de distribuição ficou ainda mais eficaz. Entretanto, no início de cada distribuição ainda há um gargalo por haver apenas um servidor com as imagens. Como trabalho futuro, pretende-se implementar o protocolo *IP Multicast* juntamente com *BitTorrent*, como descrito em alguns dos trabalhos relacionados reportados no início do capítulo. Desta forma, acredita-se que é possível alcançar um maior número de *seeders* mais rapidamente, aumentando ainda mais a velocidade dos *downloads*.

Capítulo 6
CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma solução para laboratórios de ensino informatizados abrangendo quatro questões principais: autenticação de usuários na rede, virtualização de sistemas, gerência do sistema hospedeiro e distribuição de imagens de disco por meio da rede. A proposta de arquitetura demonstrou ser eficiente em muitos aspectos para gerência de laboratórios e distribuição de imagens.

A implementação do módulo de autenticação gerou um trabalho em *workshop* (LIMA et al., 2019). Essa implementação é capaz de funcionar de forma federada, além de isolar a área de cada usuário e verificar se a autenticação ocorreu com sucesso antes de liberar o acesso à máquina virtual.

Quanto à virtualização das imagens de disco, foi desenvolvida uma versão da solução que utiliza o sistema de arquivos ZFS e seus volumes ZVol para prover os discos virtuais, e outra que emprega arquivos no formato *qcow2*. Uma vez preparado o disco, o módulo de autenticação dispara um *script* que configura todos os pré-requisitos para virtualização com o QEMU. A abordagem de virtualização com ZFS é funcional, mas pode apresentar instabilidade em casos extremos de uso do disco. Esse problema foi solucionado com a introdução de uma arquitetura com Ext4 e *qcow2*.

Ambas as arquiteturas de virtualização propostas neste trabalho foram modeladas de forma a permitir que o sistema hospedeiro seja atualizado de maneira atômica. Testes de atualização foram realizados e, mesmo inserindo erros e simulando quedas de energia, o sistema hospedeiro continuou funcionando.

Ferramentas que facilitam a criação, instalação, atualização e gerência do sistema hospedeiro foram implementadas e outras já existentes foram adicionadas, melhorando a qualidade e o refinamento das informações. A todo momento, o sistema hospedeiro pode ser gerenciado via SSH. O sistema envia *logs* através do Fluent Bit para um servidor de coleta e de consulta dos dados com Elasticsearch e Kibana. Relatórios periódicos contendo métricas e informações de configuração são colhidos por meio do servidor com Prometheus. Essas ferramentas evidenciaram o grau de importância do monitoramento constante para prevenção de incidentes e detecção

rápida de problemas.

Scripts foram desenvolvidos para a geração de imagens dos sistemas convidados juntamente com a instalação de *softwares*. Para distribuir as imagens de disco, simultaneamente e aproveitando toda a largura de banda fornecida pela infraestrutura de rede local, foi utilizada a ferramenta Aria2c, a qual permitiu que as transferências das imagens de disco fossem realizadas por meio do *BitTorrent*. O modelo de distribuição proposto trouxe grandes vantagens para a universidade e aos administradores dos laboratórios, como por exemplo: redução do tempo de atualização dos laboratórios de semanas para minutos, menor custo e economia de tempo com deslocamento de técnicos e garantia de integridade das imagens.

A solução proposta foi implantada nas salas de aula informatizadas mantidas pela Secretaria Geral de Informática da UFSCar, e continuará a ser refinada com base na utilização observada quando ocorrer a volta às aulas presenciais, após o controle da situação atual de pandemia. Acredita-se que o modelo apresentado possa ser seguido e implantado por outras universidades, proporcionando melhorias nesse tipo de ambiente que tem sido cada vez mais importante no ensino, pesquisa e extensão. Além disso, espera-se que o conhecimento gerado por este trabalho possa ser aplicado ao gerenciamento de grandes parques de máquinas em outros contextos e outras áreas da indústria.

REFERÊNCIAS

AMAD, M.; MEDDAHI, A.; VANWORMHOUDT, G. A self-adaptive ALM architecture for P2P media streaming. In: *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*. Piscataway: IEEE, 2015. p. 1–6. Disponível em: <https://doi.org/10.1109/NOTERE.2015.7293516>. Citado na página 70.

AMIT, N.; BEN-YEHUDA, M.; YASSOUR, B.-A. IOMMU: Strategies for mitigating the IOTLB bottleneck. In: VARBANESCU, A. L.; MOLNOS, A.; NIEUWPOORT, R. van (Ed.). *Computer Architecture*. Berlin: Springer Berlin Heidelberg, 2012. p. 256–274. ISBN 978-3-642-24322-6. Citado na página 32.

ARORA, D.; KUMAR, V.; JAGDEEP, B.; VERMA, P. Proposed model for virtual labs interaction with openstack integration using kvm hypervisor. *International Journal of Scientific & Technology Research*, Citeseer, Volume 3, 2014. Citado na página 16.

BELLO, E. H. *Pluggable Authentication Module for 802.1x authentication protocol*. 2014. <https://github.com/ehbello/pam-8021x>. Citado na página 25.

BERTONI, G.; DAEMEN, J.; PEETERS, M.; ASSCHE, G. V. Keccak. In: JOHANSSON, T.; NGUYEN, P. Q. (Ed.). *Advances in Cryptology – EUROCRYPT 2013*. Berlin: Springer Berlin Heidelberg, 2013. p. 313–314. ISBN 978-3-642-38348-9. Citado na página 37.

BIDGOLI, H. *The Handbook of Computer Networks*. New Jersey: Wiley, 2008. ISBN 978-0-471-78461-6. Citado na página 67.

BONWICK, J.; AHRENS, M.; HENSON, V.; MAYBEE, M.; SHELLENBAUM, M. The zettabyte file system. In: *Proceedings of the 2nd Usenix Conference on File and Storage Technologies*. Berkeley: USENIX, 2003. v. 215. Citado na página 37.

BRASIL. *Lei do Marco Civil da Internet no Brasil*. 2014. [Online; acessado em Setembro de 2019]. Disponível em: <http://www.planalto.gov.br/CCIVIL_03/_Ato2011-2014/2014/Lei/L12965.htm>. Citado na página 25.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, v. 25, n. 6, p. 599 – 616, 2009. ISSN 0167-739X. Disponível em: <https://doi.org/10.1016/j.future.2008.12.001>. Citado na página 30.

COHEN, B. *Incentives build robustness in BitTorrent*. 2003. 5 p. Citado 2 vezes nas páginas 67 e 76.

CONGDON, P.; ABOBA, B.; SMITH, A.; ZORN, G.; ROESE, J. *IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines*. 2003. Disponível em: <https://tools.ietf.org/html/rfc3580>. Citado na página 28.

DEERING, S. *IP Multicast*. 2008. Disponível em: <https://tools.ietf.org/html/rfc1112>. Citado na página 67.

EDMUNDSON, D.; LECLANCHE, J.; MIKHAILOV, N.; FIORINI, P. L. *Simple Desktop Display Manager*. 2020. <https://github.com/sddm/sddm>. Citado na página 21.

FLETCHER, J. An arithmetic checksum for serial transmissions. *IEEE transactions on Communications*, IEEE, v. 30, n. 1, p. 247–252, 1982. Citado na página 37.

GASPAR, A.; LANGEVIN, S.; ARMITAGE, W.; SEKAR, R.; DANIELS, T. The role of virtualization in computing education. *SIGCSE Bull.*, ACM, New York, v. 40, n. 1, p. 131–132, mar. 2008. ISSN 0097-8418. Disponível em: <http://doi.acm.org/10.1145/1352322.1352181>. Citado na página 30.

GORMLEY, C.; TONG, Z. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. [S.l.]: "O'Reilly Media, Inc.", 2015. Citado na página 63.

HARDAWAY, D.; HOGAN, M. J.; MATHIEU, R. G. Outsourcing the university computer lab. *Computer*, v. 38, n. 9, p. 100–102, 2005. Citado na página 30.

INTEL. Graphics interface. In: *Intel Open Source HD Graphics Programmers' Reference Manual (PRM)*. Santa Clara: Intel Corporation, 2014. Disponível em: <https://01.org/sites/default/files/documentation/intel_os_gfx_prm_vol11_-_gfx_interface_0.pdf>. Citado na página 33.

JOSEFSSON, S. *The Base16, Base32, and Base64 Data Encodings*. 2006. Disponível em: <https://tools.ietf.org/html/rfc4648>. Citado na página 37.

KRAJCI, I.; CUMMINGS, D. Using Intel Hardware Accelerated Execution Manager on Windows, Mac OS, and Linux to speed up Android on x86 emulation. In: ____. *Android on x86: An Introduction to Optimizing for Intel Architecture*. Berkeley, CA: Apress, 2013. p. 285–302. ISBN 978-1-430-26131-5. Disponível em: <https://doi.org/10.1007/978-1-4302-6131-5_11>. Citado na página 55.

LIMA, M. L. G. de; CAMPOS, P. H. L.; MATIAS, P. Módulo de autenticação via eduroam para o pluggable authentication modules. In: *Anais do IX Workshop de Gestão de Identidades Digitais (WGID)*. Porto Alegre: SBC, 2019. Citado 2 vezes nas páginas 25 e 83.

LIU, B.; LISHEN, L.; QIN, X. Research on hardware I/O passthrough in computer virtualization. *Proceedings of the International Symposium on Computer Science and Computational Technology*, Henan Polytechnic University, Henan, p. 353–356, 2010. Citado na página 32.

MAHESWARI, B. U.; RAMESH, T. K. An improved delay-resistant and reliable hybrid overlay for peer-to-peer video streaming in wired and wireless networks. *IEEE Access*, v. 6, p. 56539–56550, 2018. Disponível em: <https://doi.org/10.1109/ACCESS.2018.2871932>. Citado 2 vezes nas páginas 74 e 75.

MARFIA, G.; PAU, G.; GERLA, M. P2P streaming systems: a survey and experiments. *ST Journal of Research*, p. 1–4, 2007. Citado na página 71.

MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, v. 2014, n. 239, p. 2, 2014. Citado na página 58.

MERKLE, R. C. A certified digital signature. In: BRASSARD, G. (Ed.). *Advances in Cryptology — CRYPTO' 89 Proceedings*. New York, NY: Springer New York, 1990. p. 218–238. ISBN 978-0-387-34805-6. Citado na página 37.

OGAWA, K.; IWAMOTO, T.; TSURU, M. One-to-many file transfers using multipath-multicast with coding at source. In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. Piscataway: IEEE, 2016. p. 687–694. Disponível em: <https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0101>. Citado 3 vezes nas páginas 71, 72 e 73.

PENNINGTON, H.; CARLSSON, A.; LARSSON, A.; HERZBERG, S.; MCVITTIE, S.; ZEUTHEN, D. *D-Bus Specification*. 2016. Disponível em: <https://dbus.freedesktop.org/doc/dbus-specification.html>. Citado na página 26.

RUSSELL, R. Virtio: Towards a de-facto standard for virtual I/O devices. *SIGOPS Oper. Syst. Rev.*, ACM, New York, v. 42, n. 5, p. 95–103, jul. 2008. ISSN 0163-5980. Disponível em: <https://doi.org/10.1145/1400097.1400108>. Citado na página 32.

SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. OpenStack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, v. 55, n. 3, p. 38–42, 2012. Citado 2 vezes nas páginas 16 e 31.

SILVA, E. *Fluentd & Fluent Bit - The Production Grade Ecosystem*. 2021. <https://docs.fluentbit.io/manual/about/fluentd-and-fluent-bit>. Citado na página 62.

SILVA, E. *Fluentd & Fluent Bit - The Production Grade Ecosystem*. 2021. <https://docs.fluentbit.io/manual/about/history>. Citado na página 62.

SILVA, T. N. de Matos de Andrade e. *Bitocast - A hybrid BitTorrent and IP Multicast content distribution solution*. 108 p. Dissertação (Mestre em Informática) — Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia, Lisboa, 2009. Citado 2 vezes nas páginas 68 e 69.

TURNBULL, J. *Monitoring with Prometheus*. [S.l.]: Turnbull Press, 2018. Citado na página 64.

WILLIAMSON, A. VFIO: A user's perspective. In: *KVM Forum*. San Francisco: The Linux Foundation, 2012. Citado na página 32.