Nuno Ricardo Cordeiro Alves

nº 42224

Linked clones baseados em funcionalidades de *snapshot* do sistema de ficheiros

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Paulo Lopes, Professor Auxiliar,
NOVA University of Lisbon

Júri

Presidente:	João Alexandre Carvalho Pinheiro Leite, Prof. Associado, Faculdade Ciências e Tecnologias da UNL - Departamento Informática
Arguentes:	Manuel Martins Barata, Prof. Coordenador, ISEL, Dep. Engenharia Electrónica e Telecomunicações e de Computadores
Vogais:	Paulo Orlando Reis Afonso Lopes, Prof. Auxiliar, Faculdade Ciências e Tecnologias da UNL - Departamento Informática

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2016

Linked clones baseados em funcionalidades de *snapshot* do sistema de ficheiros

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2]. [1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

Esta dissertação resultou de um trajeto longo e com as suas dificuldades, no qual fui recebendo um excelente apoio de várias pessoas. Neste sentido, os méritos que ela possa ter, devem-se aos contributos dessas mesmas pessoas que, durante a sua elaboração, me proporcionaram testemunhos, experiência e apoio de várias formas. Foram elas que a tornaram possível, expressando por isso e a todos, a minha mais profunda gratidão.

Quero antes de mais, destacar o papel fundamental do meu orientador, o Sr. Professor Paulo Lopes, que não só pelas suas contribuições, correções, apoio e acima de tudo persistência e determinação, mas também pela excelente capacidade que tem como professor para ensinar e passar conhecimento que foram fulcrais para que este trabalho pudesse ser realizado.

Quero também agradecer ao Sr. Engenheiro Miguel Martins, colaborador da Reditus e quem trabalhou arduamente para trazer à realidade o protótipo sobre o qual foi desenvolvida esta dissertação, não só pela sua paciência e grande disponibilidade em explicar os mecanismos internos do protótipo, mas também pelo apoio e motivação que ofereceu, sempre com grande profissionalismo. Agradeço também ao Sr. Engenheiro Henrique Mamede e ao Sr. Engenheiro Sérgio Rita, ambos colaboradores da Reditus, pela oportunidade oferecida.

Quero também deixar um enorme agradecimento e um forte abraço ao meu colega, (e agora Engenheiro) Eduardo Martins, que escreveu a sua dissertação em paralelo com a minha e cujo o apoio, amizade e debate sobre o tema foram valiosíssimas para que a minha dissertação visse a luz do dia.

Por fim, quero expressar a mais sentida gratidão ao meu Pai, à minha Mãe e ao meu Irmão mais novo, que são para mim o principal pilar na minha vida, e que mesmo nunca contribuindo de forma intelectual para a minha dissertação, a sua presença constante, amor e apoio incondicional, foram fundamentais e insubstituíveis não só para a elaboração da mesma, mas também para a minha formação e afirmação como pessoa ao longo de toda a minha vida, e que sem dúvida, nunca teria chegado onde cheguei sem eles.

De mais, agradeço às restantes pessoas presentes na minha vida, familiares e amigos que não mencionei, que de uma forma ou de outra contribuíram para que eu concluísse este objetivo que não se resume à elaboração de uma tese, mas sim ao percurso académico como um todo.

O meu muito obrigado a todos.

Resumo

Hoje em dia o uso de hipervisores está largamente disseminado, sendo que é grande a sua utilização nos Centros de Dados: na consolidação de servidores (poupança de energia, espaço, redução no fardo de administração), na rápida instanciação (*deployment*) e remoção (*retirement*) de máquinas virtuais (VMs), na facilidade com que se retoma sua execução em caso de faltas (*crash*/avaria de um servidor físico), etc..

Esta dissertação foca-se em VDIs, ***Virtual Desktop Infrastructures*** de um tipo em particular: *Client-based VDIs*, nas quais a execução dos *desktops* (virtuais) é efetuada nos *desktops* físicos, e não em grandes servidores. Em ambientes VDI, os *desktops* (virtuais) são instâncias de *templates* de VMs; tais instâncias são, por diversas razões, criadas como *thin clones* baseados em *snapshots*. A proposta que fazemos é a de que nestes ambientes com elevadas taxas de criação e destruição de VMs, o recurso a técnicas nativas de *snapshoting* do próprio sistema de ficheiros é uma solução superior à utilização das funcionalidades de *snapshoting* do hipervisor.

Palavras-chave: VDI, Btrfs, snapshot, thin clone

Abstract

Today usage of hipervisors is largely disseminated, particularly in Data Centers: for server consolidation (energy and space savings, lighter administrative work), fast deployment and retirement of (virtual) servers, and fault tolerance (one may resume a failed VM very quickly if the physical server or the VM itself crashes, etc.).

This dissertation is about Virtual Desktop Infrastructures (VDI) of a particular type: the Client-based VDI, one where virtualised desktops are run on physical ones (i.e., on the PCs themselves) as opposed to being executed in large servers. In VDI environments, user (virtualised) desktops are instances of VM templates; those instances are, for various reasons, created as snapshot-based thin clones. We propose that in these environments with high VM creation and retirement rates, file system-based snapshots (using the native capabilities of file systems) are a better solution than hipervisor-based snapshots.

Keywords: VDI, Btrfs, snapshot, thin-clone

Índice

Lista de Figuras

Capítulo 1

Introdução

1.1 Introdução

1.1.1 História

A história da interação pessoa-máquina (*HCI - Human Computer Interaction*) é rica e multivariada [18]. Centrando-a no tema do nosso trabalho, os aspetos mais marcantes são o aparecimento a) das estações gráficas interligadas a "grandes" sistemas, e b) dos postos de trabalho (também conhecidos como *desktop computers*) pessoais.

Estes desenvolvimentos nas interfaces coexistiram/resultaram das mudanças tecnológicas ao nível do *hardware* (integração numa escala cada vez maior dos componentes) e do *software* (nos sistemas de operação, dos *batch* para os *timesharing*/interativos, dos mono para os multi-utilizador e nas aplicações centradas no utilizador tendo como alvo o aumento da produtividade).

A figura 1.1 exibe uma arquitetura de um ambiente baseado em X-Windows (circa 1984 [18]): as estações gráficas baseadas em Unix, interligadas entre si e/ou a sistemas de médio/grande porte, usadas em ambientes tão diversos como os centros de investigação/universidades, empresas de projetos de engenharia (civil, mecânica, aeronáutica), gabinetes de arquitetura, etc. A figura 1.2 representa o paradigmático e ubíquo computador pessoal (PC) com o ambiente *Microsoft Windows*.

As arquiteturas apresentadas ilustram também um aspeto que é central no nosso trabalho: no ambiente *X-Windows* a execução do "núcleo computacional" das aplicações desenrola-se no servidor, que tem de ter recursos suficientes para suportar múltiplos utilizadores em simultâneo, enquanto que a execução das primitivas gráficas se desenrola na estação gráfica; já no ambiente PC, tudo é executado no posto de trabalho.

Como é evidente, a história não se fica por aqui, e se o ambiente X favorece e incentiva a partilha de aplicações e dados, já no ambiente PC apenas se atinge esse objetivo se

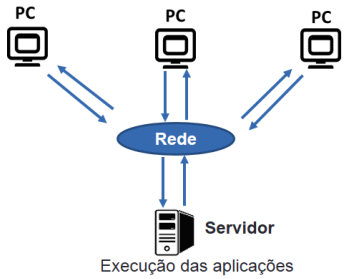


Figura 1.1: Sistema X-Windows

Figura 1.2: Sistema *Microsoft Windows*

os computadores estiverem interligados numa rede onde há repositórios/servidores de partilha de dados (uma vez que as aplicações se encontram "naturalmente" instaladas nos próprios PCs), tal como mostra a figura 1.3.

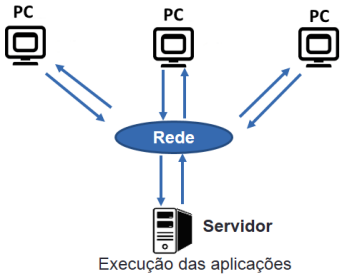


Figura 1.3: Computadores em rede para partilha de dados.

1.1.2 Aspetos funcionais e operacionais dos *Desktops Windows*

Funcionalmente os dois ambientes anteriormente descritos são semelhantes, oferecendo ao utilizador as já mencionadas ferramentas de aumento de produtividade: processador de texto, folhas de cálculo, aplicações de desenho, etc. No entanto, enquanto que o trajeto dos ambientes X evolui no sentido da simplificação das tarefas de administração (por exemplo, gestão de utilizadores, permissões, etc) e operação (instalação e manutenção de *software* de sistema e de aplicação) culminando com a introdução de estações gráficas *diskless* que carregam o *software* da rede, o ambiente MS-Windows torna-se cada vez mais complexo e difícil de gerir não só devido à diversidade de *hardware* (diferentes fabricantes e versões assim como gerações distintas de PCs) mas também à dificuldade de manter o posto de trabalho operacional face a "ameaças" externas e internas como vírus e todas as suas variantes, instalação descontrolada de *software* por parte dos próprios utilizadores ou as constantes atualizações (*patches*) dos sistemas de operação e aplicações pelos fabricantes - algumas das quais falham no processo de atualização, etc..

1.1.3 Redução da complexidade e custos de operação: as abordagens iniciais

Umas das apostas para resolver os problemas causadas pela proliferação de PCs nas organizações passa pela utilização de ferramentas e instalação (*deploy*) automatizada de imagens. Neste contexto, uma imagem é um pacote (*bundle*) de *software* que inclui o sistema de operação e o conjunto de aplicações a instalar num grupo de PCs. Naturalmente não há uma única imagem que contenha todas as aplicações e configuração necessárias, mas sim um conjunto de imagens que abarcam a diversidade de *hardware* existente e eventualmente, *software* de aplicação a instalar para diferentes "perfis" de utilizadores. Depois de construída a imagem, a ferramenta faz o *deploy* nos equipamentos selecionados, a que se segue um processo de configuração que introduz as características únicas de cada posto de trabalho em instalação - por exemplo, no caso do *Windows* é necessário o nome do computador, o SID (*Security IDentifier*), o nome do domínio a que pertence, as configurações de rede, etc..

Numa outra abordagem, em meados dos anos 90, a Microsoft incorpora no *Windows NT 4* um produto originalmente desenvolvido pela Citrix, e por esta designado *WinFrame*, que propõe um ambiente conceptualmente similar ao do *X-Windows*, no qual um servidor executa as aplicações e realiza todo o trabalho gráfico (aqui diferindo claramente do X), utilizando os PCs apenas como estações de "apresentação" das imagens geradas no servidor, sendo que a interação estação/servidor é mediada por um protocolo designado RDP (*Remote Desktop Protocol*). Esta solução, designada pela *Microsoft* como *Windows Terminal Server*, resolve naturalmente os problemas operacionais anteriormente referidos, mas requer servidores de grande capacidade (ao nível dos seus recursos como CPU, memória, discos, NICs) aumentando apreciavelmente o seu custo de aquisição (especialmente quando visto na perspetiva do custo por posto de trabalho).

1.1.4 Introdução da Virtualização no mundo dos *Desktops*

As tecnologias de virtualização de computadores (adiante mais aprofundadamente estudadas) baseados em arquiteturas x86 (Intel/AMD) têm enorme desenvolvimento na década de 90 e, embora o seu impacto inicial seja fundamentalmente naquilo que se convencionou designar por virtualização de servidores, acabaram por "invadir" outras áreas de TI e em particular serem utilizadas naquilo que se designa por infraestruturas de *Desktops* virtuais (VDI - *Virtual Desktop Infrastructure*). As VDI são, tal como o nome indica, infraestruturas de *desktops* (por exemplo, PCs) que têm uma característica fundamental: são máquinas virtuais (VMs - *Virtual Machines*).

No momento em que se decide que o suporte para um posto de trabalho (o *desktop*) não é mais uma máquina física mas sim uma máquina virtual, adquirem-se imediatamente vários "graus de liberdade" que de outra forma dificilmente seriam possíveis de atingir e as diferentes realizações disponíveis no mercado e/ou investigadas com maior ou menor sucesso correspondem, portanto, a escolhas nessa palete de possibilidades.

É assim possível: **a)** executar uma VM, "imagem" virtualizada de um posto de trabalho, num servidor e aceder ao *desktop* (virtualizado nessa imagem) através de um protocolo como o RDP ou outro similar, ou optar por **b)** executar essa mesma VM num PC, *laptop* ou *tablet* de utilização indiferenciada da organização (i.e., que não seja especificamente pertencente a um utilizador, mas de uso genérico para qualquer elemento da organização).

1.1.5 Motivação e Objetivos

As empresas atualmente debatem-se com vários problemas, tanto ao nível do número de postos de trabalho (*desktops* físicos) aos quais têm de dar manutenção (variando entre as dezenas até às centenas ou milhares) como à capacidade de conseguir alargar o número de postos rapidamente, mantendo ao mínimo o custo envolvido tanto na sua aquisição, como na administração da infraestrutura.

O método tradicional passa por instalar o *software* manualmente, indo a um posto de trabalho de cada vez. Este método, além de ser lento, sobrecarrega a equipa de manutenção dos postos de trabalho, satura a rede quando existem atualizações (que são desencadeadas no primeiro *power-up* do dia), e é ineficiente e dispendioso quando se descobre que uma atualização teve uma consequência inesperada (por exemplo, um novo bug) e é preciso repor a versão anterior.

As soluções para estes problemas passam por minimizar a quantidade de *software* instalada nos postos de trabalhos (que pode ir desde postos que não possuem disco rígido, *thin clients*, até postos que têm uma menor variedade de *software* instalado).

Em alternativa à tradicional instalação de todo o *software* no posto de trabalho, este pode ser constituído como uma máquina virtual que é executada diretamente no próprio posto ou, em alternativa, num servidor que depois "fornece" ao posto de trabalho o ecrã a exibir ao utilizador. Mesmo existindo diferentes implementações do paradigma de "virtualização do posto de trabalho", neste trabalho apresenta-se uma das mais relevantes, conhecida como *virtual desktop infrastructure*, na qual a computação é executada em máquinas virtuais (que correspondem ao postos de trabalho), armazenadas num centro de dados, e que podem ser executadas do lado do servidor (*server-based*) ou do lado do cliente (*client-based*) [15].

Assim, propomo-nos a estudar formas eficientes de armazenar as referidas VMs, um dos problemas da virtualização de *desktops*, propor uma solução, e um conjunto de testes que será necessário executar para avaliar a eficiência da mesma.

Este trabalho de tese insere-se num projeto designado por iCBD, *Infrastructure for Client-Based (virtual) Desktop (Computing)*, ou seja, uma infraestrutura para execução no posto de trabalho (neste caso, o cliente) de *desktops* virtualizados. O projeto foi agraciado com um financiamento no programa P2020, tendo sido submetido pela Reditus S.A., assessorada por uma equipa do DI-FCT/NOVA composta pelos Professores Doutores Nuno Preguiça, Paulo Afonso Lopes e Pedro Duarte Medeiros [15].

O objetivo do projeto é o desenvolvimento de uma infraestrutura computacional que

suporta a execução, de uma forma não intrusiva, de *desktops* virtuais nos PCs que geralmente se podem encontrar numa organização. A característica fundamental deste projeto reside precisamente no aspeto não intrusivo que o distingue de outras soluções anteriormente tentadas (por exemplo, pela Citrix). Aqui o disco é totalmente preservado não havendo lugar à instalação de qualquer *software* adicional, sendo o processo de execução desencadeado por arranque e/ou acesso remoto (*remote boot* de todo o *software* e demais dados necessários).

O foco deste trabalho, no panorama geral do projeto, está no armazenamento (*storage*) das VMs: das suas imagens, dos seus *snapshots* e dos seus ficheiros e estruturas de dados que suportam as suas instâncias quando em execução. Em particular, esta tese debruça-se sobre a utilização de sistemas de ficheiros com suporte nativo para *snapshots* (de ficheiros) que alberguem, de forma eficiente, persistente ou temporária, os itens anteriormente referidos: imagens, *snapshots* (de VMs), ficheiros de suporte à execução da VM, etc..

Antes de iniciarmos a apresentação mais detalhada do problema iremos, no capitulo seguinte, clarificar alguns conceitos e definições de forma a compreendermos melhor porque razão é, em VDI, o armazenamento tão importante.

Capítulo 2

Virtualização

2.1 Virtualização

2.1.1 Máquina Virtual

Uma máquina virtual (VM) pode ser definida como uma duplicata, isolada e eficiente, de uma máquina real, ou física [21].

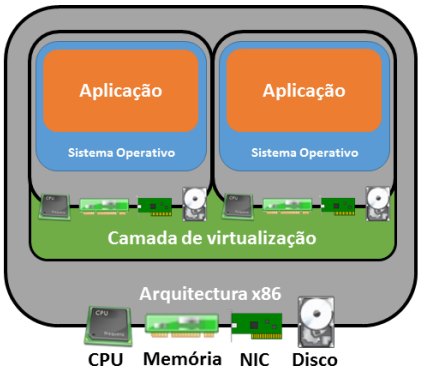


Figura 2.1: Máquina virtual.

Assim, numa VM pode ser instalado e executado todo e qualquer produto *software* que corre numa máquina física, incluindo um sistema de operação (SO). Para marcar a diferença entre um SO que corre no contexto de uma VM e um outro que corre diretamente sobre uma máquina física é costume designar o primeiro por SO *hospedado* e o segundo, quando necessário, por SO *nativo*.

2.1.2 Hipervisor

Para permitir a execução de uma máquina virtual e a sua interação com o sistema envolvente, é necessário uma camada de virtualização como a exibida na figura 2.1.

Esta camada de *software* tem o nome de *virtual machine monitor* (ou **hipervisor**) e tem o objetivo de atuar como controlador e sistema de tradução de *input/output* entre a máquina virtual e o *hardware*.

O grande desafio do hipervisor é controlar de forma eficiente os recursos físicos da plataforma "hospedeira", como por exemplo, o mapeamento dos endereços de memória e operações de I/O. Todas as VMs são controladas e monitorizadas pelo hipervisor, que também oferece ferramentas para as gerir.

Os recursos próprios, virtuais, da VM são geralmente descritos numa linguagem formal (e.g., XML) e, naturalmente, guardados num ficheiro. Encontram-se aí as especificações do número de CPUs virtuais (vCPUs) que a VM tem, a quantidade de RAM (vRAM), de disco (vDisk) e de placas de rede (vNICs) disponíveis, etc., bem como informações que caracterizam o recurso em questão, como a arquitetura do vCPU, o modelo de vNIC, entre outras.

2.1.3 Características únicas dos ambientes virtuais

Numa infraestrutura de virtualização é possível clonar máquinas virtuais, copiando toda a informação da máquina virtual "origem", incluindo a sua configuração *hardware*, aplicações, sistema de operação e dados do utilizador, para uma nova máquina virtual.

Clonar uma VM pode poupar imenso tempo quando se deseja criar uma nova VM muito semelhante a uma outra já existente, utilizando-se depois ferramentas especificas para mudar algumas configurações base, em vez de se criar, instalar e configurar uma VM totalmente nova. No entanto, quando é preciso criar vários clones da mesma VM (ou de forma frequente), existe uma outra opção em virtualização, que é partir de uma **VM *template***.

Templates são como que "cópias-mestre" de máquinas virtuais que podem ser clonadas mas não podem ser executadas e são muito mais difíceis de alterar do que máquinas virtuais; assim, é possível criar uma "biblioteca" de *templates* de VMs com diferentes ambientes (SOs) e aplicações instaladas (que sejam frequentemente usadas na organização), e rapidamente criar novas VMs prontas a serem utilizadas, e fáceis de gerir.

Outra vantagem de utilizar *templates* verifica-se na clonagem de discos. Um *linked clone*, técnica que descreveremos em maior detalhe no próximo capítulo, apenas guarda os dados alterados no clone, em modo diferencial (um "delta" dos dados), enquanto que os dados não alterados - aplicações, sistema de operação, etc., existem apenas no *template*, ao qual o *linked-clone* acede quando necessita; as alterações à VM clonada não são refletidas no *template* utilizado [32].

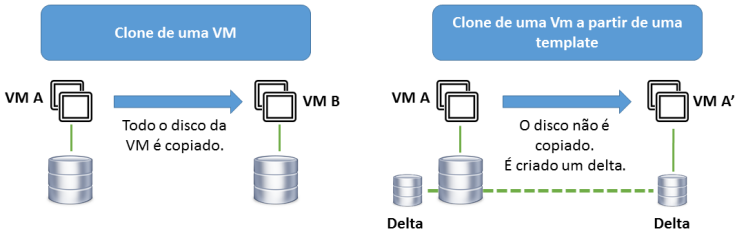


Figura 2.2: *Full* e *linked-clone*

2.1.4 O papel da virtualização: perspetiva geral

As empresas de TI precisam de tecnologias que lhes permitam responder ás necessidades atuais e futuras dos seus clientes, com o menor custo possível. Isso significa tanto o custo de aquisição como os custos de operação (incluindo energia, manutenção e gestão das infraestruturas), e uma forma elementar de reduzir estes custos é utilizar o menor número de recursos possível.

Hoje em dia, um computador tem uma performance tal que permite que várias aplicações e utilizadores partilhem os seus recursos, sob controle de um SO multiutilizador e multitarefa. Assim, surge naturalmente a ideia de, sob controle de um hipervisor, executar vários SOs em simultâneo, cada um com as suas aplicações e usos específicos, mantendo níveis de desempenho aceitáveis. Tal é possível pois alguns recursos (por exemplo, o CPU) são usualmente subaproveitados e, dotando os sistemas de outros recursos (e.g., memória, NICs, discos) em "quantidade suficiente" a partilha por múltiplas VMs pode fazer-se de forma muito eficiente.

Ao abordar este assunto aqui, optamos por discutir o papel da virtualização de forma abrangente, e referimo-nos especificamente à virtualização de servidores, e à de *desktops* - sendo que esta será mais detalhadamente discutida nas próximas secções - apenas quando necessário.

Em concreto, a virtualização, apresenta as seguintes vantagens:

- **Gestão da infraestrutura**

  Numa infraestrutura corporativa não virtualizada típica podem existir dezenas ou centenas de servidores, e milhares de postos de trabalho; em ambos os casos podem existir muitas configurações, SOs, ambientes, e aplicações distintas - e, em muitos casos, em diferentes versões. Administrar e manter toda esta infraestrutura requer muito investimento, em recursos materiais e humanos, e bastante esforço por parte das equipas de gestão da infraestrutura.

  Em contrapartida, numa infraestrutura de servidores virtualizada: **i)** há uma redução substancial no número de servidores físicos; **ii)** há uma redução substancial no tempo de *setup* de um servidor virtual, pois este é geralmente criado por clonagem, e não há quase nenhum esforço na configuração da rede e do armazenamento; **iii)**

as tarefas de atualização de *software* podem ser muito simplificadas (recorrendo a mecanismos como *snapshots*) e, se convenientemente preparadas, o tempo de indisponibilidade de um servidor pode ser muito reduzido (pode ser criado um novo servidor virtual por clonagem a partir de um *template* atualizado, e passar os serviços para o novo servidor).

Também uma VDI, na qual as VMs estão armazenadas centralizadamente, se torna muito mais fácil administrar, uma vez que se podem ter já prontos vários *templates* com os requisitos necessários a partir dos quais são dinamicamente instanciados *desktops* (VMs) prontos a serem utilizados. No caso de um dos postos de trabalho ter uma avaria, basta mudar de posto pois o *desktop* continua a estar disponível no servidor. Note-se que, sendo os *desktops* dinamicamente instanciados, as novas instâncias já "nascem" com as últimas atualizações de *software* aplicadas.

- **Segurança e Disponibilidade**

  A segurança e a disponibilidade são fatores chave em qualquer organização. Utilizando VDI é possível obter um maior controlo sobre a informação, pois os dados sensíveis são guardados no servidor e não no dispositivo utilizado para lhe aceder; e é ainda possível controlar o dispositivo que acede ao *desktop* e impedir que determinados dados sejam copiados para o dispositivo local. Ao nível da disponibilidade, na VDI ela é automaticamente garantida pelo processo de instanciação *on-the-fly* de *desktops*, enquanto que no caso de servidores virtuais a sua disponibilidade pode ser garantida por *clusters* de hipervisores - se um *host* físico que suporta um conjunto de VMs falha, estas podem ser lançadas imediatamente noutro *host*.

- **Independência face ao *hardware***

  Numa VDI qualquer dispositivo pode ser utilizado para aceder aos *desktops* virtuais, desde que: no caso das VDI *server-based*, o dispositivo seja capaz de executar a aplicação para aceder ao *desktop*; e, no caso das VDI *client-based*, o dispositivo seja capaz de executar o hipervisor que por sua vez irá executar o *desktop*. Nas infraestruturas virtualizadas, o hipervisor garante a independência relativa ao *hardware*: uma VM pode estar a ser executada num determinado *host* e ser mais tarde executada noutro com *hardware* diferente, desde que compatível.

2.2 *Virtual Desktop Infrastructure*

2.2.1 O conceito

Virtual Desktop Infrastructure [VDI] é mais um conceito que propriamente uma tecnologia. Chama-se VDI ao conjunto de tecnologias e processos (figura 2.3) que permitem armazenar um conjunto de *desktops* virtuais numa infraestrutura centralizada (tipicamente num centro de dados), e aceder-lhes através de um dispositivo físico (representado na figura pelos "Postos de trabalho" utilizando um protocolo para ligação remota [30].

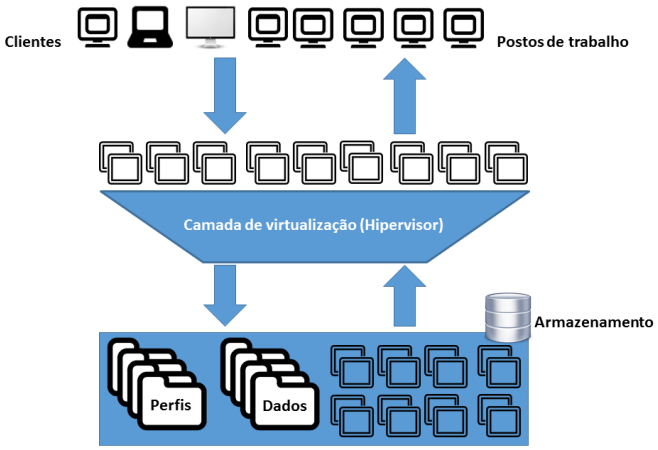


Figura 2.3: VDI - Arquitetura Lógica

O *desktop* virtual é (parte de) uma *virtual machine* e inclui a especificação da VM e do seu hardware virtualizado, sistema de operação, aplicações, dados diversos, etc., e está normalmente (mas não obrigatoriamente) armazenado num servidor; como anteriormente referimos, a execução da VM pode ocorrer a) num servidor, caso em que designamos a VDI como *server-based*, e o acesso a essa VM pode depois ser efetuado através de um *thin-client*, um computador pessoal (normal ou de baixa performance), ou outro tipo de dispositivo que seja capaz de executar a aplicação cliente necessária para se conectar ao servidor, ou b) no próprio posto de trabalho, caso em que designamos a VDI como *client-based* sendo, nesse caso, o acesso ao *desktop* garantido pelo *hardware* do dispositivo onde ele é executado.

A experiência de utilização de um *desktop* diretamente suportado por uma máquina física (portanto, o método tradicional), é de todo idêntica à de utilização de um *desktop* virtualizado, podendo-se até dizer que é muito difícil distinguir entre ambas. A virtualização de *desktops* afasta a interação, tradicionalmente muito próxima, do utilizador com o *hardware*, colocando a rede de permeio. Assim, o utilizador não tem a perceção nem a preocupação da forma como o seu *desktop* está a ser enviado, ou de onde [23].

Em VDI, e ao contrário do que acontece num sistema multiutilizador tradicional em que os utilizadores estão conectados a uma única instância do SO (partilhando-o, portanto), cada VM é isolada das restantes, protegendo a sua sessão de potenciais problemas causados pelas atividades de outros utilizadores noutras VMs, assim como do *hardware* ou *software* que a sustentam, dependendo apenas das camadas de controlo de virtualização para interagir com o resto do sistema [23].

Para explicar de forma visual as infraestruturas VDI mais comuns (mais adiante se

verá que a proposta iCBD difere em vários pontos desta forma de VDI) e como se desenrola o processo de conexão de um cliente ao seu *desktop*, apresenta-se o seguinte diagrama (figura 2.4):

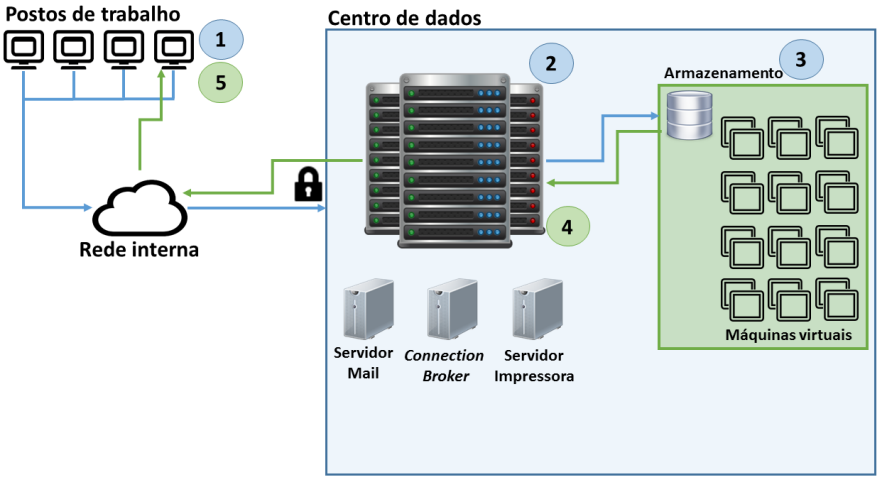


Figura 2.4: *Exemplo de Virtual Desktop Infras)tructure*

- **1:** O utilizador efetua login no seu posto de trabalho (que é o cliente VDI). Essa ligação é então encaminhada pela rede interna para a infraestrutura VDI;
- **2:** Na infraestrutura, um *connection broker* aceita a ligação feita pelo cliente para carregar um *desktop*, autentica as credenciais do utilizador, identifica qual ou quais as máquinas virtuais que este pode utilizar e inicia a execução da VM escolhida;
- **3:** É efetuado o *boot* da VM;
- **4:** Com a VM em execução (num servidor, utilizando os recursos computacionais deste), utiliza-se no cliente um protocolo de acesso remoto (como o Remote Desktop Protocol ou o PC-over-IP) para aceder ao *display* da VM;
- **5:** O utilizador, no cliente (terminal físico), tem agora acesso ao seu *desktop* tal como se este nele estivesse instalado.

Esta é a forma usual de implementar uma infraestrutura VDI. Existem outras variações, com mais ou menos recursos, e mais ou menos fatores de segurança; a secção seguinte apresenta com mais detalhe os modelos de VDI.

2.2.2 Modelos de serviço de *Remote Desktop*

Existem dois modelos muito distintos para o fornecimento de serviços de *desktop* remoto: um, mais antigo, resulta de um enriquecimento funcional de um sistema de operação *time-sharing* típico, com uma camada gráfica (virtual) a correr no servidor; o outro usa

máquinas virtuais que são executadas num hipervisor. É exemplo do primeiro o *WinFrame*, desenvolvido pela Citrix na década de 90 (e renomeado mais tarde pela Microsoft como *Windows Terminal Server*), e neste toda a execução, aplicacional e gráfica, se desenrola no servidor, sob a forma de múltiplos processos independentes, sob controle de um único SO. O segundo, recorre a máquinas virtuais, é genericamente referido como VDI, e tem duas variantes possíveis: num caso, as VMs são executadas num servidor, sendo os postos dos utilizadores (PCs, *laptops*, etc.) responsáveis por executar apenas a aplicação de acesso ao *desktop* remoto (protocolos RDP ou PCoIP); no outro, os postos são, de facto, os motores de execução das VMs, correndo eles próprios o hipervisor e acedendo às imagemns das VMs. Nestes casos designamos a primeira forma como "*Server-based* VDI" e a segunda como "*Client-based* VDI" [15].

- ***Remote Desktop* baseado em SO multi-utilizador**

  Esta arquitetura tem por objetivo disponibilizar os *desktops* (com as aplicações) aos utilizadores finais; estes usam os seus dispositivos para se conectarem a um servidor no qual partilham uma única instância de um SO (multiprogramação/*time-sharing*) e as aplicações nele instaladas, numa relação de muitos-para-um [3]. Ao efetuar todo o processamento, memória e armazenamento no servidor, os terminais (clientes) dos utilizadores não necessitam de grandes recursos, podendo assim ser do tipo *thin-client* (terminais com baixa performance, geralmente sem noção de estado e sem disco rígido, cuja única função é servir de meio de interação entre o utilizador e o servidor).

  O facto de se utilizarem *thin-clients* permite poupar no investimento em postos de trabalho assim como na sua gestão, que passa a ser centralizada no lado do servidor. Por outro lado, o servidor tem de ter recursos suficientes, o que o torna dispendioso, e é também um ponto único de falha, pelo que geralmente são utilizados vários servidores, numa arquitetura designada por *load balancing farm*.

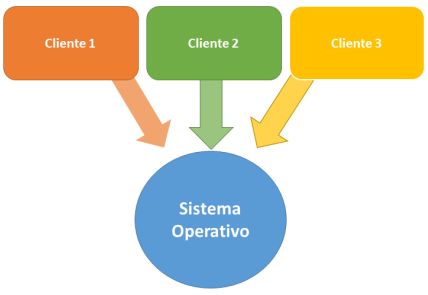


Figura 2.5: Sistema de *Remote Desktops* baseados em SO multi-utilizador

- ***Remote Desktop* baseado em máquinas virtuais**

  Como alternativa à computação baseada em servidor com um SO multi-utilizador anteriormente descrita, aparecem sistemas baseados na execução de máquinas virtuais que replicam os ambientes existentes nos PCs dos utilizadores, e que se designam por *Virtual Desktop Infrastructures* (VDI).

  Neste caso, ao permitir que os utilizadores tenham ao seu dispor um SO dedicado hospedado numa máquina virtual única, estabelece-se uma relação de um-para-um entre o dispositivo cliente e o sistema em execução, oferecendo algumas vantagens sobre o modelo anterior como, por exemplo, a possibilidade de personalização do *desktop* assim como das aplicações (o que é muito mais limitado no anterior), ou o poder oferecer múltiplos sistemas de operação num mesmo ambiente [25].

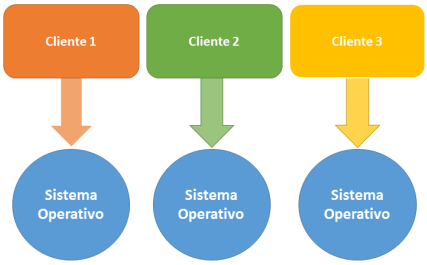


Figura 2.6: Sistema muitos-para-muitos de *Virtual Desktop Virtualization*

Existem três implementações distintas do paradigma VDI:

  - **VDI baseado em servidores**

    Neste modelo as imagens dos postos de trabalho virtuais (i.e., as VMs) estão alojadas em dispositivos de armazenamento, os quais "fornecerão", na altura da execução, a imagem ao servidor que a vai executar; este terá, naturalmente, um hipervisor instalado. O posto de trabalho do utilizador (i.e., o terminal físico) apenas precisa de conseguir executar uma aplicação de acesso a *desktops* remotos para se conectar à VM.

    Este modelo necessita, contudo, de um investimento enorme ao nível dos servidores (em CPU e memória) e no armazenamento (que necessita de ter alto débito e baixa latência). Existe ainda um problema relacionado com tarefas de visualização de videos em *streaming* ou execução de aplicações de manipulação de som e imagem que necessitam de enormes recursos ao nível dos CPUs do servidor, embora possam ser ajudadas se este for dotado de placas gráficas de gama alta - mas estas podem ser tão dispendiosas como o próprio servidor [15].

Para além das exibir as várias vantagens associadas às VDIs, este modelo permite ainda que o utilizador tenha o seu próprio dispositivo (*laptop*, *tablet*) para aceder à VM, desde que esse dispositivo possa executar a aplicação de acesso a *desktops* remotos.

– **VDI baseado em cliente**

Neste modelo, o posto de trabalho físico (i.e., o PC ou *laptop* do utilizador) dispõe de um hipervisor que fica encarregue de executar o *desktop* (SO e aplicações) sendo que este está alojado numa VM. Tanto a imagem da VM como o hipervisor podem estar alojados no disco local do posto de trabalho, armazenados num qualquer sistema que seja capaz de os fornecer ao dispositivo na altura da sua execução, por exemplo via rede, ou numa outra combinação destas duas formas de armazenamento [15].

Este modelo tem todas as vantagens das VDIs e ainda uma adicional relativamente à VDI baseada em servidores: precisa de muito menos investimento em servidores, pois ao utilizar os recursos dos proprios postos de trabalho físicos, acaba por necessitar de menos recursos nos servidores. Contudo, os maiores fornecedores de VDI, Citrix e VMware, não disponibilizam este tipo de solução no seu *portfolio* de produtos. Uma das razões prende-se, na nossa opinião, com a forma como a implementação era efetuada do lado do cliente (PC), requerendo a completa formatação do disco local, o que afastava os utilizadores (Citrix XenClient).

– ***Virtual Desktop as a Service***

Recentemente apareceu um novo modelo de VDI análogo, ao baseado em servidores, mas utiliza os recursos da *cloud* para fornecer a infraestrutura: o *Virtual Desktop-as-a-Service* (DaaS). Este modelo tem um bom potencial de minimização de custos no que se refere à aquisição e manutenção das infraestrutura, já que a organização apenas tem de disponibilizar os postos de trabalho e uma forma de estabelecer a ligação (que pode ir da ligação dedicada ao ponto-de-acesso à Internet) mas a experiência de utilização fica dependente das latências e *jitters* das ligações.

É também hipoteticamente possível combinar o modelo DaaS em conjunto com a VDI baseada no cliente, e assim a *cloud* seria utilizada apenas para armazenar as VMs - quando os utilizadores as quisessem executar, teriam de as descarregar para o posto de trabalho [15].

2.3 Arquitetura ICBD

A figura 2.7 apresenta uma visão das 3 camadas principais da ICBD, uma arquitetura VDI baseada nos clientes: **a)** uma camada de execução, que inclui os dispositivos (PCs, *Laptops*) que vão executar os *desktops* virtuais; **b)** uma camada de serviços que suporta

o arranque de um "cliente ICBD"; e **c)** uma camada de armazenamento, cujo objetivo principal é armazenar os diferentes *templates* (também conhecidos como *golden images*) que serão instanciados sob a forma de VMs [1] a serem executadas nos dispositivos (a).

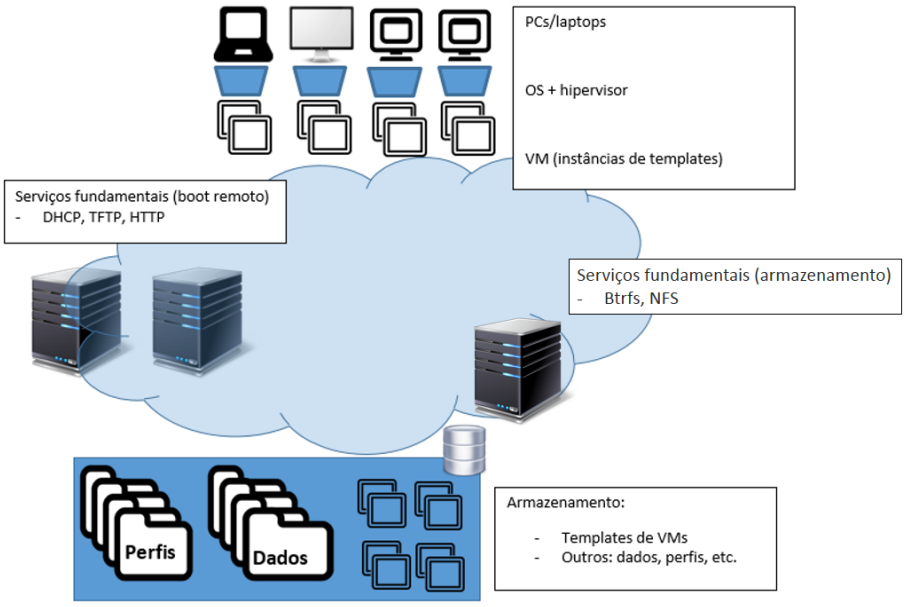


Figura 2.7: Arquitectura iCBD

2.3.1 Visão operacional: boot de um cliente iCBD

O processo de *boot* de um cliente iCBD desenrola-se em várias fases (figuras 2.8 a 2.12):

- Arranque PXE tradicional no qual também se contactam um servidor DHCP para obter um endereço IP, e um TFTP para obter um menu de *boot* (figura 2.8);
- Inicia-se uma sessão TFTP de transferência de um *kernel* Linux contendo um hipervisor (figura 2.9);
- O hipervisor acede a um clone (gerado a partir do *template* de uma VM) contendo o *desktop* e executa-o, começando pelo arranque do SO *guest* e dos seus serviços (figura 2.10);
- *Login* do utilizador no *desktop* virtual (figura 2.11);
- Conclusão do processo executando o *software* necessário para criar o "ambiente de trabalho" (figura 2.12).

Os "*screenshots*" exibidos foram obtidos numa instalação integralmente constituída por VMs, devido à falta de recursos hardware adequados, realizada para esta dissertação.

[1] Note-se que o Linux, por exemplo, pode ser geralmente executado nativamente sobre o dispositivo pelo que, se o utilizador assim o desejar e o seu dispositivo suportar, pode dispensar o uso do hipervisor.

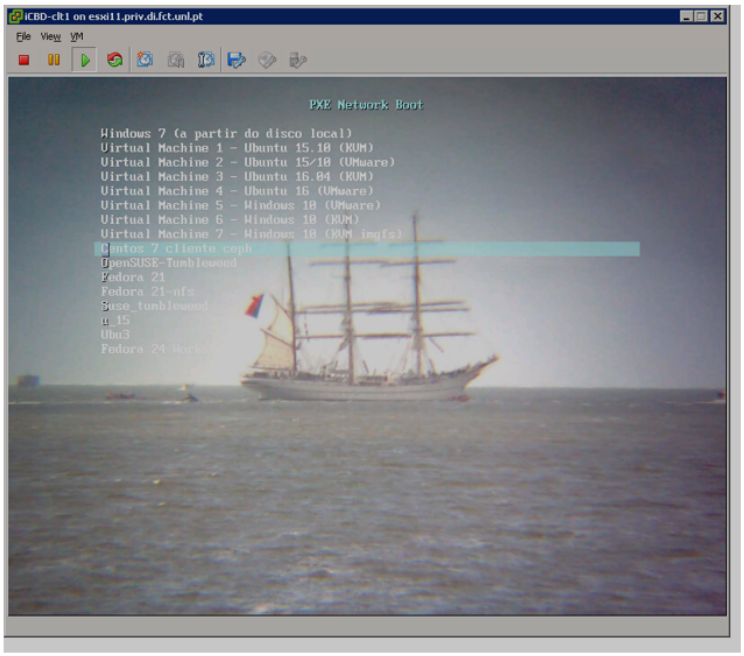


Figura 2.8: *PXE boot*: menu de opções de arranque do *kernel*.

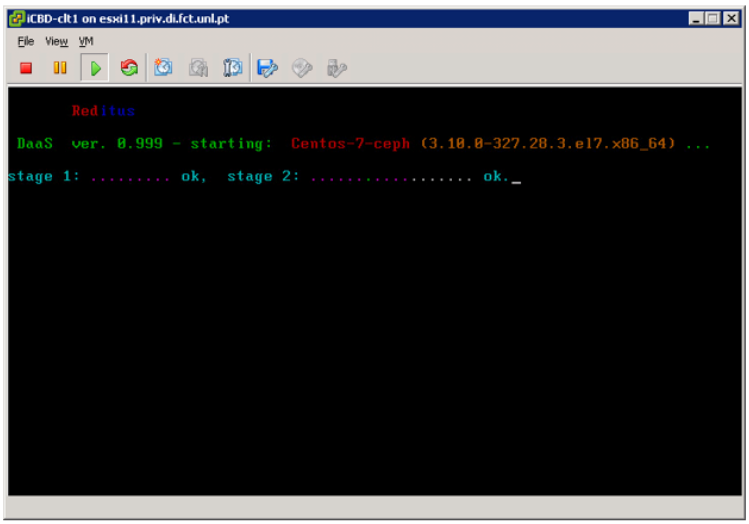


Figura 2.9: TFTP a carregar um *kernel* Linux contendo um hipervisor.

2.3.2 Suporte à execução simultânea de múltiplos clientes

O processo anterior não parece muito complexo, e é muito idêntico a tudo o que já é usado para o *boot* remoto de sistemas Unix/Linux. Quais são, então, os aspetos inovadores da infraestrutura iCBD?

Antes de responder à questão acima colocada é importante salientar que a referida simplicidade decorre, em primeiro lugar, de termos apenas considerado a existência de um único cliente. Para suportar a execução simultânea de vários clientes, é necessária a existência de "áreas privadas" por cliente, para suportar o normal funcionamento do SO

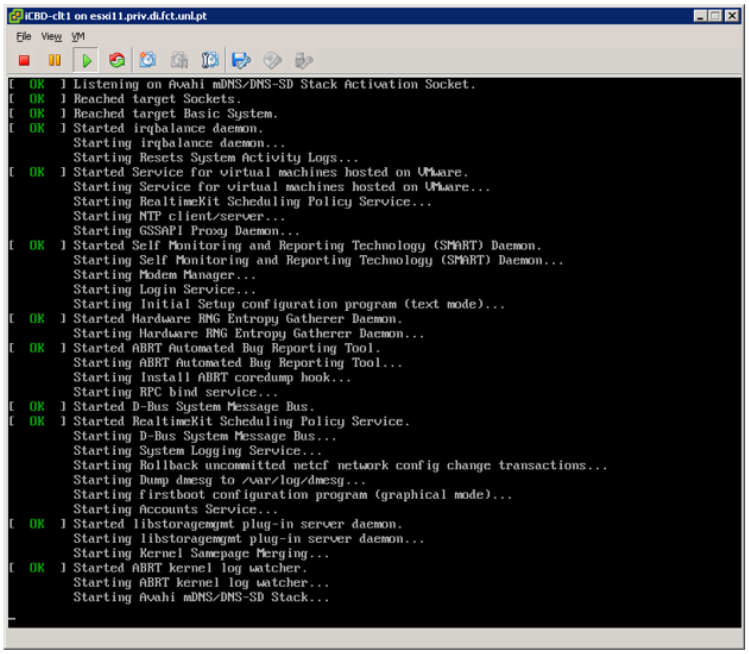


Figura 2.10: VM - *desktop* Linux neste exemplo - executada e início de arranque dos serviços (*init*).

guest, que necessita de registar muitas informações enquanto é executado.

Reconhecendo que a criação de réplicas, tantas quantos os clientes, de todo o "disco de sistema" é extremamente dispendiosa em espaço de armazenamento, a solução tradicional, baseada em NFS, recorre à criação de uma "árvore partilhada" no servidor, na qual existem zonas que são apenas de leitura e consequentemente podem ser partilhadas entre os clientes (e.g., /bin, /usr, etc.) que as montam em regime *read-only*. Para as zonas não-partilhadas, são criadas áreas privadas para cada cliente, que as monta em *read-write*.

Nestas soluções, utilizadas desde os anos 90, a gestão de "clientes *remote boot*" (criação de estruturas de dados para novos clientes, remoção, alteração) é/era suportada pelos utilitários dos SOs Unix, nessa altura mais comuns: Aix, HP/UX, SunOS/Solaris, etc.. Nas soluções muito posteriores (primeira década do novo século), baseadas em iSCSI ou FC, para cada "cliente" (*initiator*) é criado um volume correspondendo a um tradicional disco físico de *boot*, sendo este montado para leitura/escrita – perdendo-se aqui a possibilidade de partilhar as "áreas comuns".

A ausência neste texto de referências a outros SOs que não de tipo Unix não acontece por acaso: de facto a única forma sustentável (i.e., que não exija técnicas que podem funcionar numa versão e não numa outra) de fazer *boot* de, por exemplo, um sistema Windows a partir de "imagens externas" é utilizar iSCSI ou FC.

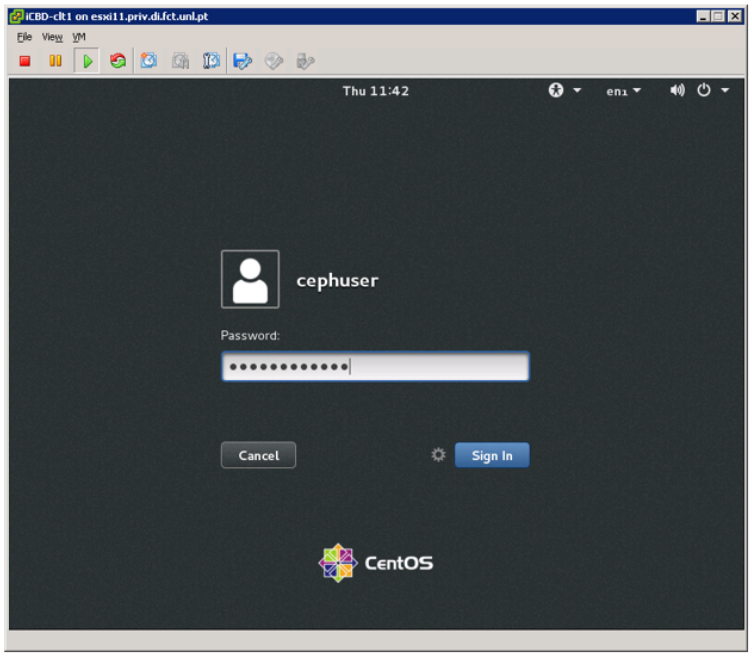


Figura 2.11: Desktop activo, fase de *login* do utilizador.

2.3.3 iCBD: *remote boot* de máquinas nativas ou virtuais a partir de *templates*

O iCBD, ao suportar não só a execução em modo nativo [2] mas também de VMs, ultrapassa os constrangimentos anteriormente referidos. Desta forma podemos revisitar toda a descrição anterior (e as opções nela descritas) considerando que o SO que fazemos arrancar inicialmente inclui um hipervisor instalado, como por exemplo o KVM ou *VMware* (*Workstation* e/ou *Player*), sendo por isso possível passar-lhe a indicação de qual a VM a executar. O utilizador nem se apercebe, se a VM for iniciada em modo *full screen*, que o sistema que está usar é uma máquina virtual, como se mostra nas figuras 2.11 e 2.12.

O passo seguinte da solução proposta pelo iCBD é na redução, considerável, da complexidade conceptual e do fardo de administração. Onde as soluções baseadas em NFS recorrem à criação (ainda que automatizada) de uma árvore de ficheiros por cliente e as baseadas em iSCSI recorrem a processos manuais [3] lentos de construção dos volumes para cada cliente, a iCBD propõe que a administração seja realizada sobre uma única entidade: a *template* de VM.

Assim, quando o utilizador escolhe executar nativamente uma versão Linux, esta é "extraída" em tempo de execução de um *template*. Já quando escolhe executar uma versão *Windows* (ou um Linux sob a forma de VM), o processo decorre em duas etapas: primeiramente é executada nativamente uma versão Linux ("extraída" de um *template*, naturalmente, e versão essa que inclui um hipervisor); seguidamente, é efetuado um clone

[2] Seja de Linux ou de um qualquer outro SO que seja passível de executar em ambiente *remote boot*.
[3] Não é do conhecimento do autor a existência de uma solução automatizada que realize estas tarefas.

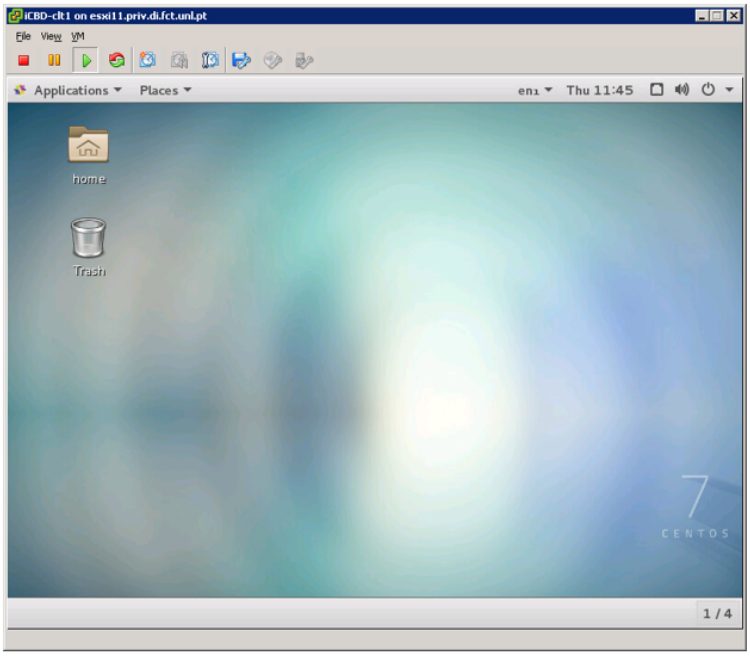


Figura 2.12: *Desktop* pronto-a-usar.

do *template* (Windows, Linux, ou outro) escolhido, sendo em seguida executado o referido clone.

Capítulo 3

Armazenamento - o problema

O problema que se pretende abordar neste trabalho prende-se com o armazenamento das máquinas virtuais em ambiente VDI. O armazenamento em VDI oferece desafios únicos, pois tanto os discos das máquinas virtuais como os dados dos utilizadores (dessas mesmas máquinas) podem estar alojados no mesmo local, o torna um dos pontos mais críticos na implementação de um projeto VDI.

Isto implica um especial cuidado quando se está a planear a arquitetura do sistema de armazenamento bem como a sua infraestrutura concreta: o *hardware*, a rede, os protocolos, e o *software* - que são os componentes de uma solução que no final há de apresentar um valor para a relação custo/desempenho.

O ideal, já o dissemos, é conseguir que a experiência de utilização utilizando ambiente VDI seja tão próxima quanto possível da tradicional, aquela em que sistemas de operação e aplicações estão diretamente instalados nos postos de trabalho. No entanto, para o conseguir, é necessário compreender os problemas e desafios da VDI de forma que estes possam ser mitigados ou mesmo resolvidos.

3.1 Armazenamento de Máquinas Virtuais

3.1.1 Máquinas virtuais: os dados e os metadados

A nível do sistema de armazenamento, uma máquina virtual é composta por um conjunto de metadados (itens que definem a sua especificação) e dados - SO e aplicações guardadas no(s) disco(s) da VM, e valores dos parametros da BIOS, etc.. Tais metadados e dados têm de ser guardados num sistema de armazenamento, seja este de que tipo for; contudo, por razões de simplicidade de exposição, admitiremos que são guardados sob a forma de ficheiros. Os principais ficheiros necessários para o funcionamento de uma VM são **a)** o ficheiro de configuração da VM, **b)** o ficheiro do disco virtual, **c)** o ficheiro da NVRAM

(valores de parâmetros da BIOS) e o ficheiro de *log*. Estes ficheiros são utilizados pelo hipervisor responsável por gerir e executar a VM, utilizando para isso os recursos físicos da máquina hospedeira tal como mostra a figura 3.1.1.

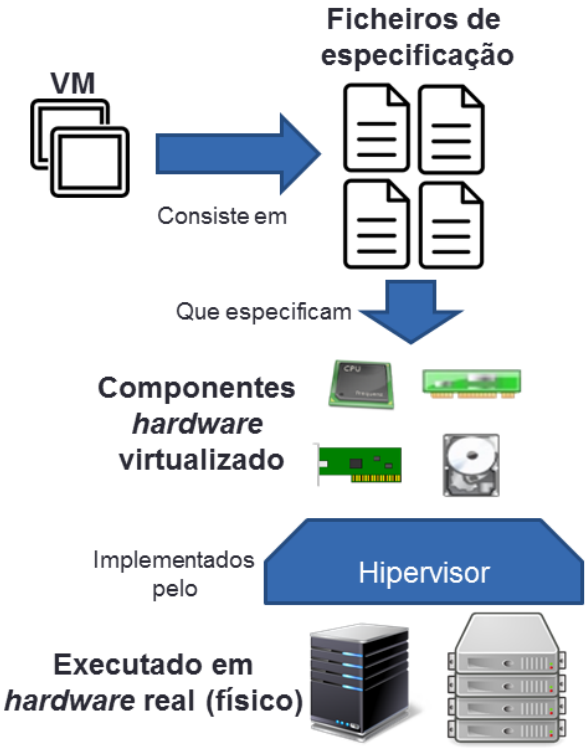


Figura 3.1: Composição de uma máquina virtual.

Para além destes, a VM pode ainda ter outros ficheiros caso existam *snapshots* da VM. Como o nome indica, um *snapshot* é uma imagem, num determinado instante, do estado de um recurso; aqui, estamos particularmente interessados em *snapshots* de volumes (discos lógicos), e de ficheiros (individualmente ou agrupados, por exemplo, numa diretoria).

Podemos imaginar a implementação de um *snapshot* da seguinte forma: **a)** o estado do recurso é armazenado sob a forma de um "objeto" persistente e imutável; e, **b)** as alterações ao estado do recurso passam a ser armazenadas num outro objeto. Assim, é possível regressar a um estado anterior qualquer, desde que o objeto correspondente a esse estado esteja disponível. Os *snapshots* são especialmente interessantes em ambientes virtualizados pois o hipervisor é capaz de realizar *snapshots* dos recursos mais importantes de uma VM: CPU, memória e disco(s).

Para que os múltiplos objetos, correspondentes aos múltiplos *snapshots*, não ocupem espaço desnecessariamente, recorre-se a técnicas de compactação de dados na implementação dos *snapshots*; assim, o novo objeto criado para registar a sequência de (novos) estados de um recurso que foi alvo de um *snapshot* apenas regista as modificações efetuadas, mantendo-se os estados não alterados no objeto inicial (que, recordemos, está em modo *read-only*). Diz-se também que o *snapshot* corrente é um "delta" do anterior.

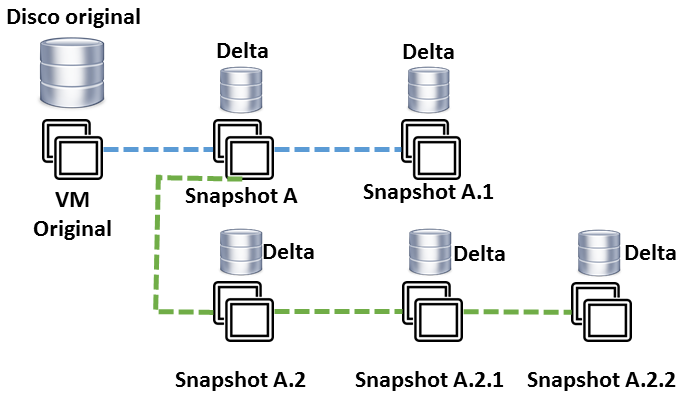


Figura 3.2: Conjunto de *snapshots*

Quando um *snapshot* é apagado, os seus dados são fundidos (*merged*) com os do *snapshot* a montante.

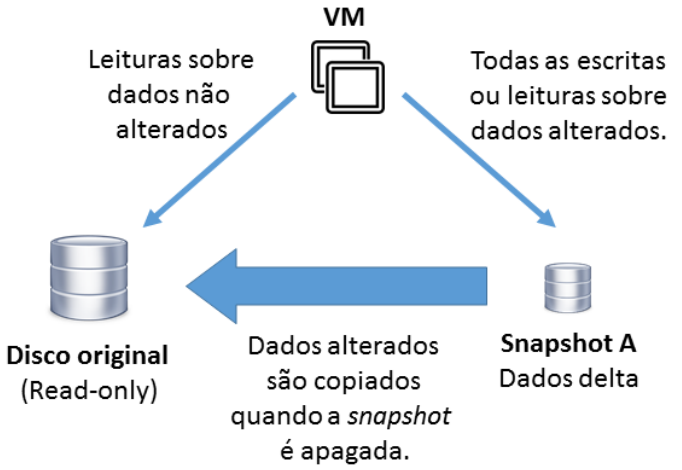


Figura 3.3: Como funcionam as operações de leitura/escrita na presença de uma *snapshot*.

Por exemplo, no caso do hipervisor KVM, os *snapshots* podem ser efetuados de várias formas, o que vai originar diferentes ficheiros [22] [11]:

- ***Snapshots* internos:** Este tipo de *snapshot* requer que a imagem da VM seja do tipo qcow2, que é um formato que permite o *copy-on-write* e pode ser executado com a VM *online* (o libvirt utiliza o comando QEMU 'savevm' neste caso) ou *offline* (o libvirt utiliza o comando QEMU 'qemu-img' para realizar o *snapshot*). É utilizado um único ficheiro tanto para salvaguardar o estado da VM no momento em que o *snapshot* foi efetuado, como o "delta" dos dados que virão a ser alterados desde essa altura.

Os *snapshots* internos podem ainda ser classificados como:

  - **de disco:** Apenas o estado do disco virtual é guardado.
  - ***system checkpoints*:** É guardado o estado do(s) disco(s) e, se a VM estiver a ser executada (*online*) o estado da RAM e dos periféricos. Toda esta informação é guardada num único ficheiro qcow2.

- ***Snapshots* externos:** Neste caso, o estado da VM aquando a execução do *snapshot* e guardado num ficheiro (que se marca como *read-only*) e é criado um novo ficheiro (designado *overlay*) que irá guardar os dados modificados desde o instante do *snaphot*.

  Os *snapshots* externos podem ainda ser classificados como:

  - **de disco:** Apenas o estado do disco virtual é guardado num ficheiro (*read-only*), sendo os dados modificados desde o instante do *snapshot* guardados num novo ficheiro, do tipo qcow2. Podem ser executados com a VM *online* ou *offline*.
  - ***system checkpoints*:** Neste caso, o disco virtual é guardado num ficheiro e a RAM e o estado da VM são guardados num outro novo ficheiro.

- ***Snapshots* do estado da VM:** Neste caso apenas o estado da RAM e dos periféricos é guardado (mas o estado do disco virtual não), para que possam ser repostos mais tarde se necessário. A operação é semelhante à hibernação de um SO, exigindo que o disco virtual não tenha sido modificado até à altura da reposição.

3.1.2 Armazenamento: os desafios da VDI

A representação de forma persistente de uma VM é, como vimos, geralmente efetuada recorrendo a ficheiros, tipicamente um por cada "entidade" - por exemplo, um para o disco virtual, outro para as configurações, outro para os *snapshots* (ou um separado para cada *snapshot*), outro para o estado da memória quando a VM é suspensa, etc..

Sendo o objetivo de uma VDI fornecer *desktops* virtuais aos utilizadores, a forma como estes *desktops* (que são VMs) estão armazenados é um dos pontos principais a ter em atenção. Seguidamente, apresentam-se alguns dos principais desafios que uma VDI enfrenta, na ótica do armazenamento: [28]

- **Custo por posto de trabalho**

  Como é natural, pretende-se que o custo por posto de trabalho seja tão reduzido quanto possível, ao mesmo tempo que se mantêm níveis aceitáveis de performance.

  Quando se planeia uma solução VDI, no que ao armazenamento se refere, existem dois pontos principais a considerar na avaliação do seu desempenho: a taxa de ***Input/Output Operations Per Second*** (IOPS), medida comum para fazer a avaliação da performance de discos (tanto os tradicionais discos magnéticos, como os *solid*

state drives, SSDs), e a quantidade de **espaço de armazenamento** necessária por cada VM.

A taxa de IOPS pode ser aproximada pela seguinte fórmula: **1 / (*seek time* + latência)**. Este valor não têm em conta qualquer operação de escrita ou leitura, mas é um bom ponto de partida para estimar o número de operações de leitura ou escrita que o disco poderá executar por unidade de tempo [31].

A tabela que se segue apresenta valores típicos de IOPS para HDDs em função da sua velocidade de rotação:

RPM	**Latência**	***Seek time***	IOPS
5400	5.5	7 - 14.4	50 - 80
7200	4.2	5.9 - 9.1	75 - 100
10,000	3	3.6 - 5	125 - 150
15,000	2	2.7 - 3.7	175 - 210

Tabela 3.1: Valores médios de IOPS para discos mecânicos [31].

Tendo em conta os valores apresentados, podemos perceber que mesmo utilizando simultaneamente vários discos mecânicos em configurações RAID 0 ou 5, o que se torna uma solução dispendiosa para ambientes com grande número de postos, é difícil oferecer o desempenho adequado, razão pela qual muitos projetos VDI falham, pois os discos magnéticos tradicionais (HDDs), pelas suas partes mecânicas, têm dificuldade em acompanhar os requisitos únicos de VDI; por exemplo, um *desktop Windows 7* gera em média 15 IOPS em funcionamento normal, chegando aos 60 quando se efetua o *boot* [19].

Contudo, se utilizarmos *linked-clones*, em que todas as (instâncias de) VMs criadas a partir de um *template* partilham os mesmos dados (a imagem principal), mesmo sendo a quantidade de pedidos (IOPS) muito elevada, uma vez que a imagem é a mesma e está, total ou parcialmente, em cache, o tráfego que efetivamente se exerce sobre o disco está limitado ao acesso aos *snapshots* individuais das instâncias.

Sabendo-se que uma VM *Windows* pode ter em média 50 GB (com sistema de operação e aplicações instaladas), pode optar-se por utilizar um sistema híbrido que combina discos *Solid State Drive* (cuja a performance em IOPS é da ordem dos 100 000) e HDDs, para não ter que suportar o custo de uma solução totalmente baseada em SSDs (que são muito mais caros e têm menos capacidade que os discos mecânicos).

- ***I/O storms***

  Um I/O *storm* acontece quando o pedido, por parte dos sistemas de uma infraestrutura é de valor (em IOPS) superior ao que a infraestrutura consegue oferecer,

fazendo com que o(s) sistema(s) de armazenamento percam a capacidade de atender os pedidos "em tempo real", provocando atrasos nas respostas aos utilizadores [28].

Numa infraestrutura computacional típica bem desenhada é raro, em condições normais, acontecerem situações que provoquem uma quantidade massiva ou constante de I/O que provoque impacto negativo no ambiente. No entanto num ambiente VDI existem alguns eventos que podem desencadear I/O *storms* que acabam por introduzir degradações no tempo de resposta - por exemplo, os utilizadores terem de esperar 20 minutos somente para efetuarem o *login* no seu *dekstop*, o que diminui o nível de satisfação dos utilizadores e pode ter impacto nos resultados (financeiros) da empresa.

Num ambiente empresarial há várias situações que podem causar I/O *storms* regularmente. Por exemplo:

- **Arranque (*boot*) das VMs (também chamado *boot storm*)**: Acontece quando, ao iniciar um turno de trabalho, vários utilizadores executam simultaneamente o *boot* do seu *desktop*. Nesse momento, todas as VMs estão simultaneamente a realizar várias operações de leitura e escrita sobre o sistema de armazenamento, o que se traduz em maus tempos de resposta e longas esperas pelo fim do processo de arranque.

  Uma solução muito utilizada para contornar este problema consiste na manutenção de uma *pool* de instâncias (de VMs) prontas a serem "consumidas" pelos utilizadores; assim, quando estes querem iniciar uma sessão de trabalho um *broker* seleciona uma VM da *pool*, e o utilizador consegue efetuar imediatamente o seu *login*. À medida que o número de instâncias disponível na *pool* diminui, um processo em *background* efetua o *boot* de mais VMs, de forma diferida, mantendo o número de pedidos de I/O sob controle.
- ***Login* dos utilizadores (também chamado de *Login storm*)**: Logo depois do processo de *boot*, os utilizadores têm de efetuar o *login* no *desktop*. Este processo também resulta num enorme número de I/O em simultâneo de várias VMs num curto espaço de tempo, pois o sistema tenta carregar bastantes ficheiros relacionados com o perfil dos utilizadores.
- **Varrimento de *malware***: Normalmente estes varrimentos são executados num horário em que causem o menor impacto possível; no entanto, pode acontecer que muitos iniciem o varrimento ao mesmo tempo, o que pode ter impacto negativo.
- **Lançamento de aplicações**: Algumas aplicações exigem bastante I/O quando são iniciadas; numa sala de aulas, por exemplo, o professor pede aos alunos para iniciarem uma determinada aplicação e os pedidos serão, muito provavelmente, simultâneos.

  - **As atualizações de terça-feira**: *Patch tuesday* é um termo não oficial utilizado para referir o dia em que normalmente a Microsoft lança as atualizações de segurança nos seus produtos; essas atualizações, se aplicadas em simultâneo a todos os postos, causam uma enorme quantidade de pedidos de I/O.

- **Gestão do armazenamento**

  Sendo o sistema de armazenamento uma das partes cruciais e mais dispendiosas de uma VDI, é importante garantir que este tem o melhor desempenho possível, sem *bottlenecks* e sem espaço desnecessariamente ocupado. [24]

  Um dos problemas que de uma deficiente utilização do espaço de armazenamento designa-se ***VM Sprawl***, e designa o crescimento descontrolado do número de VMs numa infraestrutura virtualizada. Este problema é, em parte, causado pela facilidade com que se pode obter uma nova máquina virtual quando comparada com a dificuldade - em tempo de espera pela aprovação, aquisição, entrega, instalação, etc. - de obter uma máquina física, processo que pode demorar semanas e ter custos financeiros apreciáveis. Afinal, para obter uma VM basta pedi-la ao administrador especificando o *hardware* virtual (CPU, memória, disco) e esperar uns minutos que a VM seja criada.

  Assim, mesmo considerando que VMs inativas não gastam CPU nem memória, a proliferação de VMs pesa de múltiplas formas: gastando espaço em disco e aumentando a complexidade do trabalho de administração da infraestrutura.

  No caso de uma VDI, mesmo que não haja *sprawl*, a gestão do espaço é um desafio especialmente importante, dado o elevado número de *virtual desktops*; tomando o já citado exemplo de 50 GB por cada *desktop* Windows, uma organização com uns meros 100 utilizadores necessita, no mínimo, de 5 TB de espaço de armazenamento, com elevado desempenho para as VMs.

  Existem duas tecnologias que ajudam a minimizar o espaço ocupado: a primeira, designada ***thin disks*** (que podemos traduzir como discos esparsos, se atendermos à sua natureza similar à dos ficheiros esparsos), aplica-se tanto a ambientes virtualizados como físicos; a segunda, recorre a *snapshots* de discos e, embora se possa igualmente aplicar tanto a ambientes virtualizados como físicos, é mais comunemente encontrada nos primeiros.

  Thin disk é uma técnica de criação de discos (volumes) lógicos que utiliza estruturas de dados (bitmaps, árvores, etc.) que apenas utilizam blocos que tenham sido alocados para dados, o que permite fazer *overbooking* do espaço físico de armazenamento, podendo até criar-se a ilusão de que existe mais espaço disponível do que aquele que realmente existe. O conceito *thin* tem várias implementações, muitas vezes apresentadas com nomes distintos: *thin disks* quando ao nível dos volumes lógicos oferecidos por *disk arrays* ou sistemas de armazenamento baseados em objetos;

e *sparse files* (ficheiros esparsos), quando ao nível de ficheiros. A ideia-base é a de que a maioria dos "consumidores" de espaço não irá, simultaneamente, utilizar todo o espaço que lhe foi atribuído, havendo por isso uma melhor partilha dos recursos disponíveis. A alternativa é utilizar a forma tradicional em que o espaço pedido para o volume ou ficheiro é imediatamente alocado e não pode ser utilizado para outros fins.

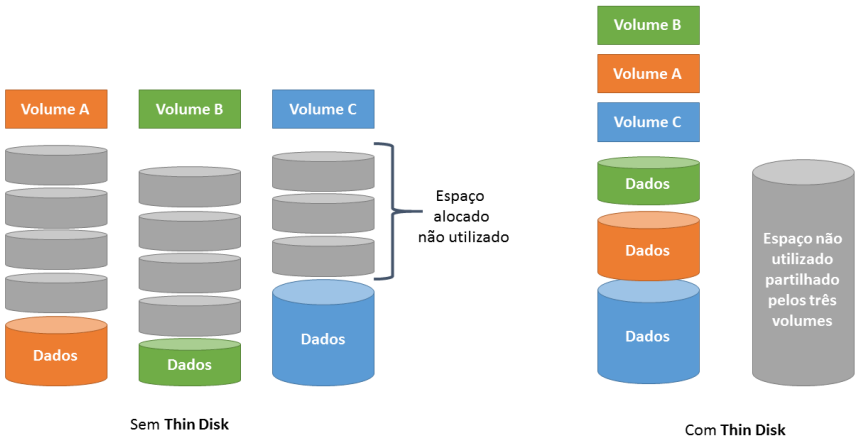


Figura 3.4: Armazenamento com e sem *Thin Disk*

Como seria de esperar, a tarefa de administrar o armazenamento é agora mais complexa, e exige maior atenção.

Um outro conceito, igualmente interessante e também capaz de contribuir para um melhor aproveitamento do espaço é o *snapshot*. A VMware (uma das empresas líder em produtos para ambiente virtualizado) recomenda que os *snapshots* sejam apagados em menos de 24 horas e que não sejam feitos mais do que 3 *snapshots* interligados (apesar dos seus produtos suportarem mais de 30 interligações). Também não recomenda efetuar *snapshots* em VMs que tenham I/O intensivo.

- **Planeamento do subsistema de armazenamento**

  Os desafios ao nível do subsistema de armazenamento para uma VDI são, como já se disse, de dois tipos: dotar o sistema de **a)** capacidade suficiente, e **b)** desempenho adequado.

  Em ambientes de TI típicos, com servidores físicos que correm SGBDs e aplicações cujo comportamento é estável, conhecido, sem picos relevantes de carga, etc., dimensionar os subsistemas de armazenamento, muitas vezes internos aos próprios servidores, é uma tarefa geralmente simples: utiliza-se a capacidade disponível e se for preciso mais, faz-se um *upgrade* ao servidor. Mas mesmo nestes ambientes há soluções que, ao serem introduzidas para melhorar alguns aspetos, têm como

consequência tornar o dimensionamento mais complexo; um exemplo é a introdução de dispositivos de armazenamento externo (*disk arrays*) que, ao permitir gerir melhor o espaço disponível e criar as condições para fornecer alta disponibilidade e tolerância a faltas, tornam o dimensionamento mais complexo tanto na perspetiva do espaço como do desempenho.

A virtualização, ao introduzir mais uma camada de *software* na infraestrutura vem naturalmente tornar o dimensionamento ainda mais complexo: as VMs não acedem, geralmente, a discos/volumes (físicos ou lógicos - LUNs), mas a ficheiros que representam esses discos. Tais ficheiros existem em volumes formatados e geridos por um sistema de ficheiros o que apresenta, do ponto de vista do desempenho, maiores desafios: é preciso caracterizar o desempenho do *hardware* e ainda do *software* - o sistema de ficheiros e a camada de virtualização de I/O.

No caso das VDIs, um fator para o aumento da complexidade advém da grande variação no número de pedidos efetuados ao longo de um dia de trabalho ao sistema de armazenamento, consequência do número de eventos de *boot* e *shutdown*, muitas vezes acompanhados de criação e destruição de instâncias de VMs, quando estas são transitórias e criadas apenas a pedido [24]. E outro fator, igualmente importante, decorre da heterogeneidade de sistemas de ficheiros e protocolos usados na infraestrutura. Por exemplo, para armazenar as VMs podem usar-se sistemas de ficheiros genéricos (Linux ext3 ou ext4 [17], XFS[27]), ou especializados (VMFS [29], NFS [2]); todavia os primeiros só podem ser usados em infraestruturas com um único nó (servidor) enquanto que os segundos suportam *clusters* de múltiplos nós. Mas, nem uns nem outros são adequados para armazenar os dados dos utilizadores, que muitas vezes são partilhados com outros utilizadores e grupos de utilizadores, sendo de preferência armazenados em sistemas de ficheiros CIFS.

Em resumo, o dimensionamento de um sistema de armazenamento para uma VDI é uma tarefa muito complexa e que requer um planeamento cuidadoso.

3.1.3 As soluções

De forma a melhorar a eficiência e utilização de recursos em ambiente VDI, propõem-se neste trabalho duas possíveis formas de armazenar máquinas virtuais; destas, a primeira, armazenamento de VMs em sistemas de ficheiros será experimentada nesta tese, estando a outra a ser alvo de estudo separado noutra tese paralela.

- **Sistema de ficheiros**

  A utilização de sistemas de ficheiros é o método tradicional para armazenar VMs. Um sistema de ficheiros (SF) é utilizado para gerir a forma como se armazena e (posteriormente) acede à informação nos dispositivos de armazenamento. Existem muitos tipos diferentes de sistemas de ficheiros, com funcionalidades únicas, mais

ou menos opções de segurança/privacidade, mais ou menos controlo da integridade dos dados, e com diferentes taxas de desempenho; não existe um SF que seja, em qualquer situação, melhor que todos os outros, é preciso encontrar um que seja adequado para uma situação particular, ou para um conjunto de situações.

Sendo as VMs representadas por ficheiros, o SF que as armazena tem um enorme papel tanto na performance da VM como na sua eficiência, e deve também ser robusto e capaz de garantir a integridade dos dados. O papel do sistema de ficheiros responsável por armazenar as VMs é tão importante, que empresas como a VMWare criam o seu próprio SF para armazenamento de VMs, o Virtual Machine File System (VMFS)
. O VMFS [33] é um caso particular no panorama dos SFs porque é um sistema de ficheiros para clusters de disco partilhado [LOPES, pCFS], o que significa que um disco pode ser montado simultaneamente por vários nós, e foi desenhado para um fim específico, o armazenamento de VMs. Infelizmente o VMFS é proprietário, o que significa que apenas pode ser usado em sistemas com o hipervisor de tipo I (nativo, ou *bare-metal*) ESXi da VMware.

Neste trabalho de tese propomo-nos utilizar a funcionalidade de *snapshot* do próprio sistema de ficheiros, presente em alguns dos sistemas de ficheiros que foram concebidos mais recentemente, como o BTRFS e ZFS, para criar *linked-clones* como alternativa aos *linked-clones* criados pelo próprio *software* de virtualização. A ideia que preside a esta proposta é a de que a forma mais eficiente para criar um *snapshot copy-on-write* num SF é usar os mecanismos do próprio SF, em vez de criar estruturas de dados externas genéricas, aplicáveis a qualquer SF, que a realizem.

Para além de investigar essa possibilidade, e quais as suas vantagens e desvantagens em relação aos atuais *linked-clones* realizados pelo hipervisor, interessa também conhecer o desempenho do SF escolhido quando este é utilizado para armazenar VMs. Medir o desempenho de um sistema de ficheiros para armazenamento de VMs é algo que pode ser visto em várias fases. Numa primeira fase, pode medir-se o desempenho do SF segundo os parâmetros tradicionais de largura de banda (em acesso sequencial), e número de operações de I/O realizadas por segundo (IOPS, em acesso aleatório) e o desempenho do SF em operações sobre metadados - tipicamente número de operações de criação e remoção de ficheiros e diretorias. Numa segunda fase, os mesmos testes podem ser efetuados sobre *snapshots* de estruturas do SF: de volumes, de diretorias e de ficheiros. Depois, podem efetuar-se os testes de desempenho que são mais representativos de um ambiente virtualizado: tempo de arranque (*boot*) e paragem (*shutdown*) de uma VM; idem, para uma "rampa" com uma taxa de crescimento de arranque (*boot*) e paragem (*shutdown*) de x VMs/segundo, etc.. E finalmente, fazer testes de desempenho "intra-VMs", i.e., correndo os testes nas próprias VMs, exercendo cargas de I/O sobre o sistema de ficheiros onde os discos virtuais das VMs residem.

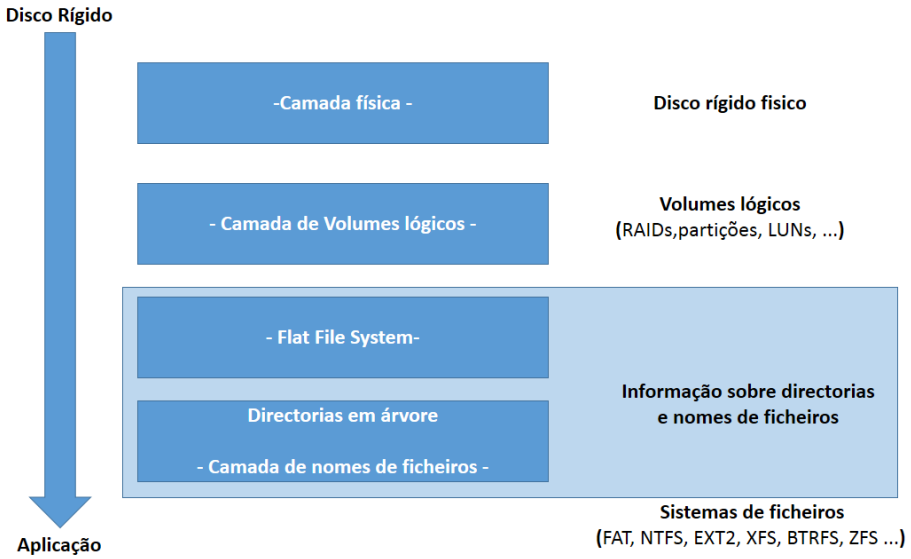


Figura 3.5: Camadas de um sistema de ficheiros.

Um sistema de armazenamento de ficheiros pode ser dividido em cinco camadas:

- **Camada física** que corresponde ao disco em si;
- **Camada de volumes lógicos** encarregue de criar uma abstração do *hardware* através da criação de volumes lógicos (LV) resultantes de partições, volumes RAID, etc.. Os LVs exibem ao SO os LVs como simples vetores de blocos.
- ***Flat File System*** camada que cria e gere os metadados necessários para contabilizar os blocos livres/ocupados no volume, os blocos que pertencem a cada ficheiro e a forma de o identificar (num espaço único de identificadores);
- **Camada de diretorias e nomes**, é responsável por estruturar espaço de identificadores dos ficheiros em diretorias no formato de uma árvore, ou outra organização considerada adequada, fazendo corresponder aos nomes externos vistos pelo utilizador os identificadores dos ficheiros;

- **Armazenamento *Object-storage***

  Uma outra alternativa para armazenar VMs, seria utilizar um sistema de armazenamento de objetos, (*object-based storage*, OBS).

  Um sistema OBS pode ser posicionado na camada FFS do modelo apresentado acima, uma vez que mapeia identificadores de objetos (OIDs) em objetos, que são pares (dados, metadados) nos quais os dados são estruturados, ou não, de acordo com as informações constantes nos metadados. Os sistemas OBS são sistemas distribuídos, i.e., utilizam múltiplos nós - clusters - com capacidades computacionais, para além

das capacidades de armazenamento. Um exemplo de um sistema OBS é o Ceph [CITAR], que oferece uma camada de objetos, o RADOS [CITAR] e, sobre esta, um sistema de ficheiros, CephFS [CITAR].

Nos OBS típicos não existe limite no tipo ou tamanho destes *metadados* associados ao objeto, o que os torna altamente versáteis; por exemplo, é possível incluir informações sobre classificação de segurança dos dados armazenados no objeto, ou outra informação relevante para uma dada aplicação que os vai manipular.[14].

Arquitetura de um sistema de armazenamento de objetos

A figura abaixo apresenta, em traços gerais, a arquitetura de um sistema de armazenamento de objetos.

A camada inferior engloba os dispositivos de armazenamento, que podem ser discos magnéticos (*hard disk drives* ou HDD), ou de estado sólido (SSD), ou outros dispositivos.

O acesso a esta *pool* de armazenamento pode ser feita diretamente ao nível do disco físico, como exemplificado na parte esquerda da figura 3.1.3, ou a do disco virtual criado por um sistema de armazenamento - por exemplo, um armário de discos (*disk array*) acessível numa infraestrutura de rede de armazenamento *(Storage Area Network*, SAN).

A camada seguinte é a do controlador dos objetos e é responsável por executar a criação, destruição, escrita e leitura de objetos, mapeando-os sobre os dispositivos de armazenamento. A comunicação com esta camada é feita através de APIs, sendo cada vez mais comuns as de tipo ReST (sobre HTTP) que oferecem, pelo menos, as quatro operações CRUD (***C**reate*, ***R**ead*, ***U**pdate*, ***D**estroy*).

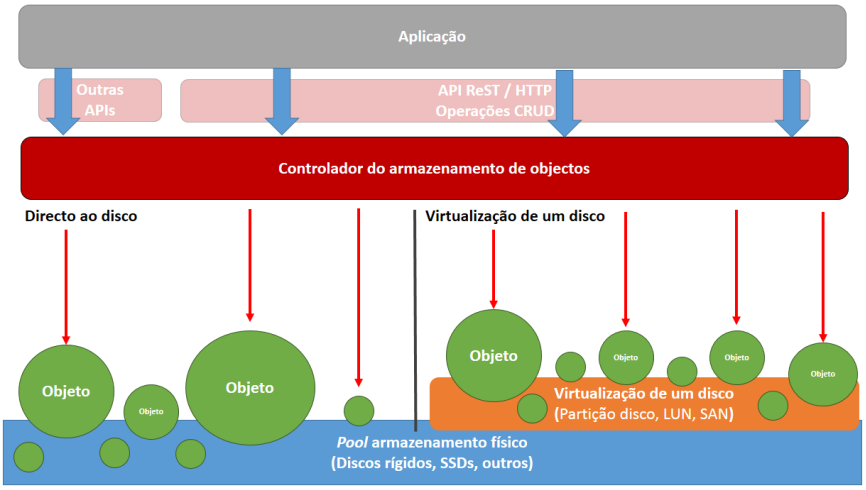


Figura 3.6: Arquitetura de um sistema de armazenamento de objetos.

Características

Os sistemas de armazenamento baseados em objetos apareceram recentemente e são construídos para oferecer desempenhos e escalabilidades elevadas (da ordem dos muitos terabytes ou petabytes de informação). Estes sistemas podem ter centenas a milhares de nós e oferecem alta disponibilidade por via da redundância nos dados, tolerando por vezes múltiplas falhas: ao nível da rede, da energia, dos nós, dos discos, etc. [14].

Capítulo 4

Projeto iCBD: caracterização e contribuições

4.1 Introdução

O projeto iCBD (*Infrastructure for Client-Based Desktops*) é um produto de *software* em desenvolvimento na Reditus S.A. com a colaboração de uma equipa de docentes do DI-FCT/NOVA (Professores Nuno Preguiça, Paulo Afonso Lopes e Pedro Medeiros) que tem por objetivo criar uma infraestrutura de suporte à execução de *desktops* virtualizados nos postos de trabalho; tal infraestrutura pode, do ponto de vista mais abstrato, ser vista na figura 6.2.

4.1.1 Funcionamento (do ponto de vista do utilizador)

Na iCBD, as imagens das VMs (*desktops* virtuais que constituem os postos de trabalho dos utilizadores pretendem utilizar), estão guardadas num sistema de armazenamento de imagens, sendo este constituído por um ou mais servidores. Quando o utilizador liga o seu posto de trabalho, recebe via rede um menu de opções que lhe mostra os ambientes que pode utilizar (figura 4.2). *Grosso modo* podemos subdividir esses ambientes em dois tipos: os que correspondem à execução nativa de um sistema de operação (por exemplo, Ubuntu 16.04, ou Fedora 21) e os que correspondem à execução virtualizada de um sistema de operação (por exemplo Windows). Não nos interessa aqui abordar a execução, nativa, mas sim a virtualizada (note-se que nada nos impede de fazer uma execução virtual de um sistema Linux).

Quando o utilizador escolhe executar um SO virtualizado, o posto de trabalho começa por descarregar da rede a imagem de um sistema Linux "quase" mínimo que é executado no posto de trabalho (figura 4.1.1); esse sistema está dotado de um hipervisor (KVM ou

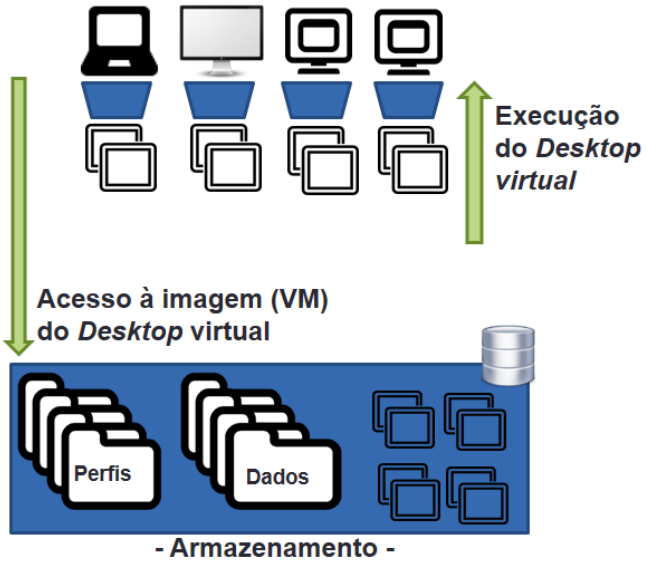


Figura 4.1: Esquema de uma VDI baseada em clientes (postos de trabalho).

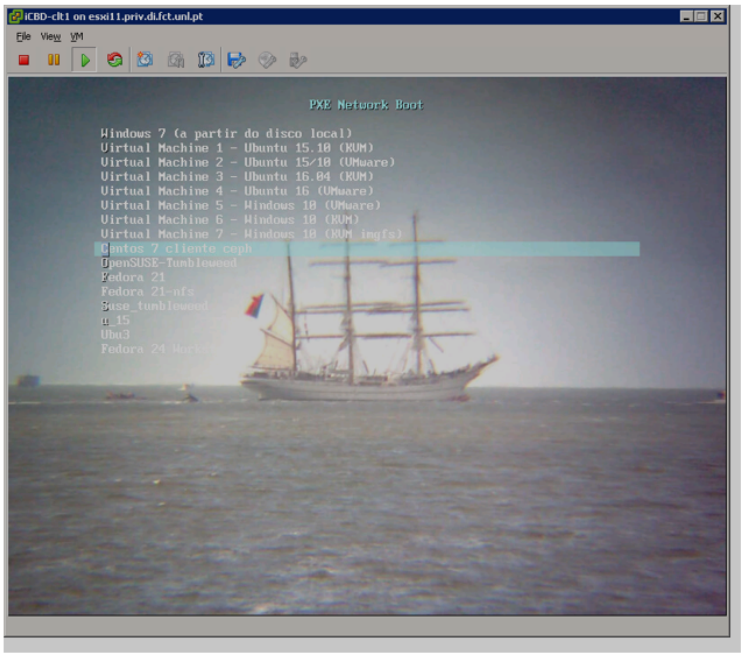


Figura 4.2: a) Menu de opções de escolha do ambiente a executar.

VMware Player) que: i) é imediatamente invocado; ii) "recebe" do sistema de armazenamento de imagens a imagem de uma VM; e iii) executa essa VM - após *login* o utilizador tem acesso ao seu *desktop* , como mostra a figura 4.1.1.

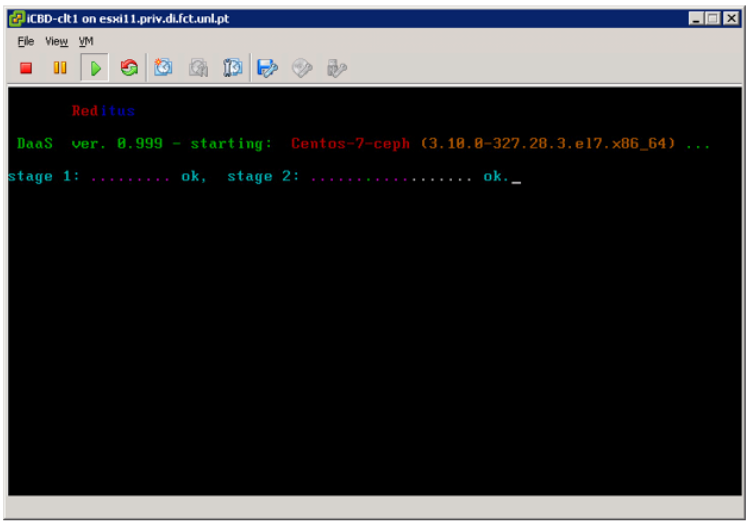


Figura 4.3: b) Carregamento de um *kernel* Linux contendo um hipervisor.

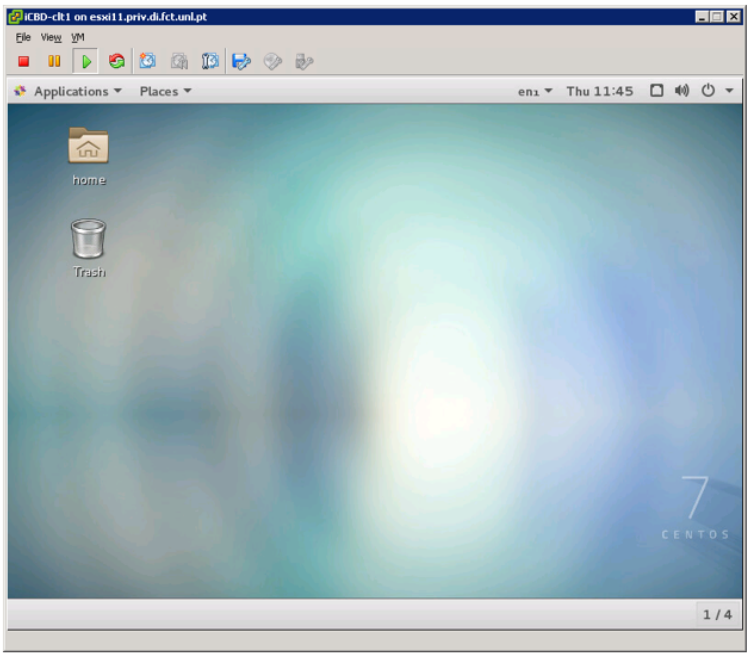


Figura 4.4: c) *Desktop* pronto-a-usar.

4.1.2 Arquitetura e funcionamento da iCBD

Os aspetos fundamentais da arquitetura iCBD são: a) **armazenamento das imagens**; b) suporte ao ***boot* remoto**; e c) suporte à **execução de instâncias**. A figura 4.5 mostra uma realização possível para esta arquitetura, na qual o suporte ao *boot* remoto é efetuado pelos servidores HTTP, TFTP e DHCP, e o servidor de imagens "controla" o armazenamento de *templates* (de VMs) e "produção" das suas instâncias.

O processo de comunicação entre o posto de trabalho e o servidor é feito utilizando

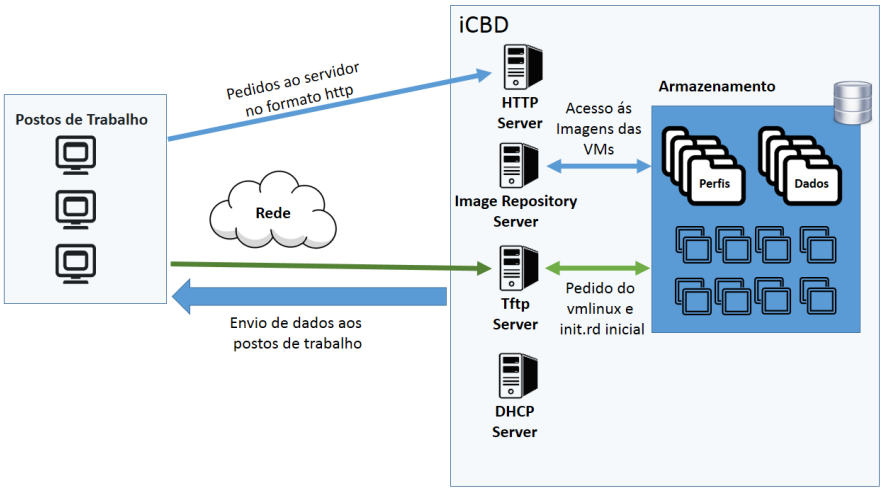


Figura 4.5: Arquitetura física do iCBD.

mensagens com o protocolo HTTP, permitindo flexibilidade e eficiência na comunicação das mensagens. Ao iniciar-se o processo de arranque do posto de trabalho, os seguintes passos são executados:

- É feito um **PXE/TFTP *boot*** de um *template* Linux (que é, de fato, aqui tratado como uma imagem);
- É carregado e executado o *kernel* e o *initd*;
- É executado um *script* para identificação da(s) NIC(s) existentes no do posto de trabalho e carregar os *drivers* correspondentes;
- Esse mesmo *script* utiliza HTTP para executar *wgets* que irão descarregar os restantes *scripts* iCBD necessários (estes *scripts* podiam ser incluídos no *init*, mas de forma a torná-lo o mais pequeno possível e não ter de se editar tantas vezes durante o processo de desenvolvimento, optou-se por esta opção).
- É executado o *script rc0* que cria o *setup* do ambiente que vai montar a imagem (entenda-se, disco virtual) escolhida. Este *mount* é, nesta fase, em *read-only* e pode ser feito tendo como alvo uma imagem acessível por NFS, iSCSI ou outro protocolo que venha a ser suportado.
- Por último é criada uma instância do *template* que constitui a imagem a montar em *read/write*, tal como se faz quando se monta um disco (físico) de sistema nos últimos passos do arranque. Isto é necessário para que possam existir múltiplos postos a executar simultaneamente o mesmo *template*, mas não a mesma imagem - já que esta é modificada durante a execução em cada um dos postos.

É fácil apresentar um exemplo da necessidade de instâncias: suponha-se um grupo de utilizadores que desempenham uma mesma função numa organização: quando usam o

seu posto de trabalho iCBD, "executam" todos um mesmo *template* - imaginemos, um que inclui Windows e Office. Só que, de facto, o *template* não pode ser o mesmo, porque: **a)** por questões de eficiência na gestão de "imagens" não se executam *templates*, mas sim *clones* de *templates*; **b)** alguns hipervisores (e.g., VMware) proíbem mesmo a execução de *templates*; e, finalmente, **c)** mesmo que fosse possível executar um *template*, se o mesmo estivesse a ser simultaneamente executado em vários postos, o seu conteúdo seria destruído.

4.2 Desafios da iCBD ao nível do armazenamento

Como ficou referido no Capítulo 3, a escolha do tipo de armazenamento tem um enorme impacto em toda a infraestrutura VDI. Para uma organização, interessa reduzir os custos totais de uma solução, e isso passa por reduzir não só os custos de investimento em *hardware* e *software*, mas também os custos de operação e de gestão - e estes incluem não só os recursos computacionais mas os custos dos recursos humanos afetos à gestão e manutenção da solução.

Focando-nos no armazenamento de dados, as quatro questões essenciais são o **volume de dados armazenado**, o **volume de dados comunicados,** a **eficiência computacional do serviço** e a **gestão da infraestrutura**.

4.2.1 Eficiência do volume de dados armazenado

Um sistema de armazenamento com a capacidade e desempenho necessários para suportar uma VDI é oneroso, pelo que a rentabilização da capacidade instalada é fundamental.

Um ponto de partida para esta rentabilização é reconhecer que, geralmente, os utilizadores não chegam a utilizar nem 50% da espaço de armazenamento atribuído ao seu *desktop*. Numa VDI, esse espaço não utilizado por ser rentabilizado se se utilizar, por exemplo, **thin disks** (descritos anteriormente em 3.1.2) para constituir os discos dos *templates*. Se à constituição de *templates* baseados em tecnologia de **thin disks** juntarmos a utilização de *linked-clones* (estes por sua vez baseados em *snapshots*), conseguimos reduzir apreciavelmente o consumo de espaço numa VDI.

Adicionalmente pode ainda usar-se **de-duplicação** de dados, assunto sobre o qual não nos alongaremos por estar fora do âmbito deste trabalho. Há que referir, contudo, que a de-duplicação pode trazer benefícios consideráveis em duas áreas: no armazenamento dos *templates* (por exemplo, ao reduzir o espaço utilizado para armazenar dois *templates* com uma base comum - um de Windows 7, e outro de Windows 7 com Office); e no armazenamento de dados dos próprios utilizadores (é muito comum a existência de um mesmo ficheiro guardado múltiplas vezes pelo mesmo utilizador e copiado para vários utilizadores...). A de-duplicação pode ser efetuada a nível dos ficheiros ou dos blocos. A nível dos ficheiros, simplesmente remove os duplicados de um mesmo ficheiro; ao nível de bloco, pode utilizar algoritmos de *hash* para identificar os blocos semelhantes,

aumentando bastante a eficiência em termos de espaço recuperado. No entanto, também necessita de muito mais poder de processamento.

4.2.2 Eficiência do volume de dados comunicados

Implementado o sistema VDI *client-based*, a maioria do tráfego de dados entre o *desktop* e os servidores será, para ambientes típicos de utilização, no arranque de uma VM e nos instantes que se seguem - após as primeiras transferências de dados (*login* do sistema, transferência de dados do utilizador e arranque inicial de algumas aplicações) o tráfego na rede deverá reduzir-se a algumas leituras/escritas esporádicas.

4.2.3 Eficiência computacional dos serviços

A utilização de VDI implica por parte da organização um grande investimento em *hardware* para o centro de dados, para que todas as VMs necessárias possam ser executadas com uma performance semelhante à de um *desktop* físico. Numa VDI *client-based*, o investimento vai na sua maior parte para o subsistema de armazenamento e infraestrutura de rede, já que o poder computacional necessário para suportar os serviços de *boot* remoto é muito baixo. No entanto é necessário que os postos de trabalho tenham memória e CPU suficientes para executar eficientemente uma VM.

Esta situação contrasta com a das VDIs *server-based*, para as quais o investimento nos postos de trabalho é pequeno, mas em contrapartida o investimento em *hardware* e *software* para o centro de dados é enorme, tanto ao nível dos servidores (computação) como do armazenamento (*disk-arrays*) e rede. E, em casos em que os utilizadores realizam tarefas que exigem boa capacidade de processamento de imagem e/ou som (ver videos, por exemplo), o investimento nos servidores pode ser enorme.

4.2.4 Eficiência da gestão da infraestrutura

Gerir uma infraestrutura com centenas ou milhares de postos de trabalho pode ser um verdadeiro pesadelo para muitos administradores; uma VDI fornece algumas ferramentas para melhor gerir a infraestrutura, de forma centralizada e mais rapidamente.

Em particular, no caso da iCBD, o conceito de máquina virtual é pervasivo a todos os níveis da infraestutura e em todos os casos de uso. Assim,

- Um *template* é uma VM;
- Uma instância (de um *template*) é uma VM;
- Um instância criada para uma execução nativa de Linux num posto é extraida de um *template*;
- Do ponto anterior decorre que os *templates* são as únicas entidades a administrar numa iCBD;

Tal como acontece em infraestruturas não virtualizadas, numa VDI os dados dos utilizadores não são, geralmente, guardados no *desktop*, razão pela qual este é "descartável"; no caso de uma VDI, os *desktops* são então (geralmente) transitórios, o que torna o processo de actualização de *softwate*, (*patching* e/ou *updating*) extremamente simples. Na iCBD o processo é efetuado da forma que se segue:

- Clona-se um *template* criando uma nova VM;
- Atualiza-se a VM;
- Promove-se a VM a *template* e marca-se como corrente.

Aos utilizadores basta reiniciar os seus *tdesktops* para acederem imediatamente à versão mais atual do *software*.

A criação de postos de trabalhos (correspondentes a instâncias transitórias) é uma tarefa muito simples porque, formalmente, não existe! Apenas há que: i) definir se um dado dispositivo físico (PC, *tablet*, etc.) tem acesso à iCBD, o que se faz indicando o seu *MAC address*; e, ii) criar, se for o caso, um novo utilizador.

Em caso de falhas de um *desktop* virtual, ou mesmo do posto de trabalho físico, é muito fácil obter uma nova instância (VM) e, se necessário, o utilizador pode simplesmente trocar de dispositivo.

Em resumo, apesar de existir alguma manutenção por parte do administrador da infraestrutura, essa manutenção é mínima.

4.3 Contribuições

É na criação das instâncias que se insere este trabalho de dissertação; e, especificamente, na criação suportada por mecanismos de *snapshot* nativos do sistema de ficheiros onde os *templates* - que darão origem às instâncias - estão armazenados.

A criação das instâncias faz-se usualmente, já o dissemos, usando o próprio hipervisor: seleciona-se o *template* ou VM origem, especifica-se se a (instância da) VM a criar como sendo um *clone* de tipo *full* ou *thin*, e executa-se a operação.

Sabendo que há sistemas de ficheiros que suportam nativamente a criação de *snapshots* e que, nestes sistemas de ficheiros, para um processo que acede aos ficheiros um *snapshot* é indistinguível do "ficheiro-original", propomo-nos então criar instâncias usando os mecanismos de *snapshot* do sistema de ficheiros e, seguidamente, entregar tais instâncias aos hipervisores para que as executem. Resolvemos assim o problema atrás enunciado: poder ter um mesmo *template* (i.e., as suas múltiplas instâncias) a ser "executado" em diferentes postos de trabalho.

Mas, como vimos na secção anterior, não basta conseguir criar as instâncias, é preciso que estas usem eficientemente o espaço de armazenamento disponível - leia-se, usem o mínimo possível - e possam ser eficientemente executadas e acedidas (i.e., o acesso a

ficheiros de instâncias *thin* não seja muito mais lento que o acesso aos mesmos ficheiros em instâncias *thick* (ou *full*).

No próximo capítulo analisam-se dois sistemas de ficheiros capazes de suportar a execução nativa de de *snapshots*: o Btrfs e o ZFS.

Capítulo 5

Sistemas de Ficheiros com suporte para *Snapshots*

Há muito poucos sistemas de ficheiros (SF) que ofereçam a possibilidade de realizar *snapshots*; destes, dois destacam-se ou pelas funcionalidades disponíveis, pelo desempenho, ou ainda pela quota de utilizadores que conseguiram granjear.

Um desses sistemas de ficheiros é o Btrfs, criado em 2007 nos laboratórios da *Oracle* por Chris Manson; o outro é o ZFS, originário da Sun Microsystems (agora integrada na Oracle), que além das funcionalidades oferecidas ao nível dos *snapshots*, goza de uma reputação de excelente performance, em especial em testes executados em ambientes virtualizados [4].

Neste capitulo apresentam-se, de forma resumida, o Btrfs e o ZFS, faz-se uma análise preliminar de cada um e elabora-se um quadro comparativo que inclui não só estes, mas também dois outros: o ext4 [17], um sistema de ficheiros muito conhecido em ambientes Linux e que descende do ext2 e do ext3, e o VMFS [29] um SF para *clusters* de disco partilhado [16], ambos sem suporte nativo de *snapshots*.

5.1 Btrfs

O Btrfs (*B-tree file system*, pronunciado em inglês como "*butter* eff ess") [13] é um sistema de ficheiros construido de raiz, inicialmente desenhado pela *Oracle Corporation* em 2007 sendo, neste momento, um projeto sob a licença GPL com contribuições de diversas empresas e indivíduos. A partir de 2014 foi considerado um projeto estável, sendo distribuído no *Mainline Linux Kernel*, e a SUSE, uma das distribuições mais usadas no mundo empresarial, a par da Red Hat, adotou o Btrfs como sistema de ficheiros por omissão na sua distribuição.

Chris Mason, um dos principais *developers* do Btrfs, afirmou que o objetivo deste sistema de ficheiros seria dotar o Linux de um SF capaz de gerir a capacidade de armazenamento, da ordem dos PetaBytes, que já se encontra disponível (e cresce de dia para dia) assim como de uma interface limpa e confiável capaz de gerir tais volumes de dados[12]. Assim, desde 2014 que Chris Mason e alguns *developers* do Btrfs estão a trabalhar em conjunto com o facebook para ajudar a desenvolver o sistema de ficheiros.

5.1.1 Destaques da arquitetura e realização

O Btrfs utiliza **árvores B+** como estrutura principal para armazenar os dados: os nós internos são índices (onde são armazenadas as chaves) e apontam para folhas, nas quais se encontram armazenados os dados associados a essas chaves. Tudo neste sistema de ficheiros, desde os *inodes*, aos dados, entradas de diretorias, *bitmaps*, etc., é um objeto na árvore B+. As modificações aos dados são feitas num regime ***copy-on-write*** (CoW), em que quando é necessário alterar o conteúdo de um bloco, em vez de se alterar o bloco existente, cria-se uma nova cópia do anterior, com as alterações aplicadas, mantendo-se, durante o tempo necessário, ambas as versões, funcionando como um mecanismo de *journal* para qualquer tipo dados (incluindo, naturalmente, metadados), que mantém o sistema de ficheiros recuperável em caso de *crash* do sistema. As *B+trees* têm, normalmente, as suas folhas conectadas, mas no Btrfs, por causa do CoW isto não é possível. Quando há alterações, é necessário mudar todas as referências que apontam o bloco modificado, e se as folhas estivessem conectadas essas mudanças iriam propagar-se por toda a árvore [13].

Uma chave na árvore deste sistema de ficheiros é composta por três campos:

- ***objectid:*** Para cada ficheiro lógico no sistema de ficheiros, referenciado por um **inode**, é atribuído um identificador único (de objeto);
- ***type:*** O tipo de objeto; este pode indicar, por exemplo, metadados (informação sobre o tamanho do ficheiro, proprietário, permissões, etc.), entradas de diretoria, dados, etc.;
- ***offset:*** Deslocamento de um atributo num objeto; é utilizado para, por exemplo, num objeto do tipo "entrada" de uma diretoria, indicar a posição do *hash* do nome;

5.1.2 Funcionalidades mais relevantes

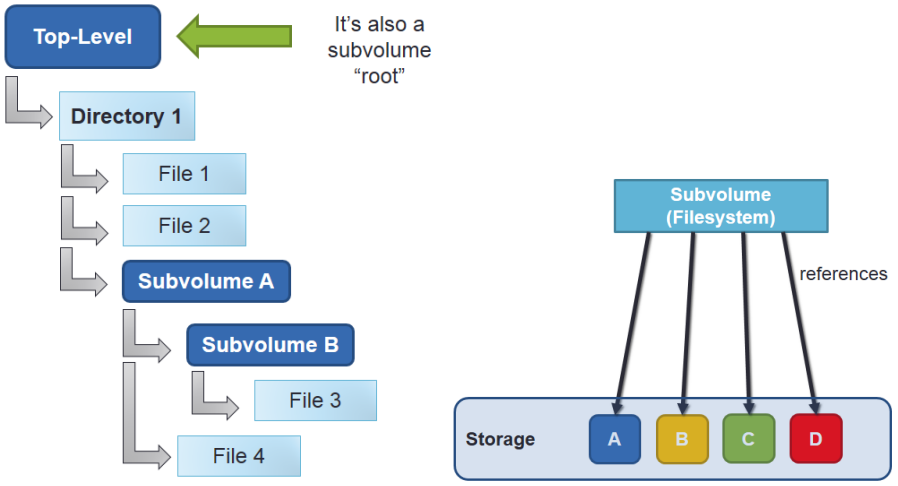
O *Btrfs* contém várias das funcionalidades presentes na generalidade dos sistemas de ficheiros mais atuais como o suporte para múltiplos dispositivos de armazenamento ou *checksums* para detetar erros que passam o crivo dos mecanismos *hardware* sem ser detetados; outras funcionalidades importantes incluem uma arquitetura que tenta dispensar a camada de volumes lógicos oferecida por sistemas como o LVM (Logical Volume Manager), a possibilidade de criar cópias por simples "clonagem" dos metadados (e portanto,

copiar ficheiros de forma quase instantânea) e as funcionalidades *send* e *receive* que permitem exportar e importar sub-volumes (ver abaixo) entre nós diferentes de uma "rede" de sistemas Btrfs, de forma a que o tráfego seja minimizado passando apenas as diferenças. Mas como já referimos, a funcionalidade mais interessante deste sistema (e menos comum entre os sistemas de ficheiros), é a sua capacidade de criar *snapshots* de diretorias e ficheiros utilizando sub-volumes.

Sub-volumes

Um **sub-volume** é, do ponto de vista do utilizador, totalmente idêntico a um volume, i.e., pode ser montado e desmontado (usando *mount/unmount*) como qualquer volume, em qualquer ponto da árvore de diretorias do SF. Do ponto de vista da realização, um sub-volume é um espaço de nomes (*namespace*) POSIX separado, ao qual é atribuído um nome, realizado com uma nova árvore B+ e que contém ficheiros e/ou diretorias; um sub-volume pode ser criado em qualquer parte da hierarquia dos sistema de ficheiros.

Podem também ser criados sub-volumes aninhados dentro de outros sub-volumes, aparecendo como sub-diretorias do sub-volume a que pertencem como pode ser visto na figura 5.1. Todos os sistemas de ficheiros Btrfs começam com um sub-volume por omissão (o *top-level subvolume*).



(a) Estrutura dos sub-volumes.

(b) Referências aos blocos em armazenamento.

Figura 5.1: Sub-volumes no sistema de ficheiros *Btrfs*

Snapshots

No Btrfs, um ***snapshot*** é idêntico a um sub-volume, mas o seu bloco raiz é inicialmente partilhado com um outro sub-volume (no Btrfs apenas se podem criar *snapshots* de sub-volumes, nunca de diretorias ou ficheiros isolados); ou seja, inicialmente quando é efetuado um *snapshot*, apenas os meta-dados do sub-volume são atualizados (o contador de referências do bloco raiz é incrementado), como se pode verificar na figura 5.2. Como resultado, criar um *snapshot* é praticamente instantâneo. Os *snapshots* podem ser criados

com o atributo *read-only* em vez do usual *read-write*.

Cada bloco tem um contador de referências para que seja fácil detetar quando é que os dados ou meta-dados desse bloco já não são necessários e podem ser libertados (ou seja, quando o contador é zero). Quando o valor do contador é superior a zero, os blocos que fazem parte dessa *snapshot* não são alterados ou apagados, mantendo sempre o estado original na altura do *snapshot*.

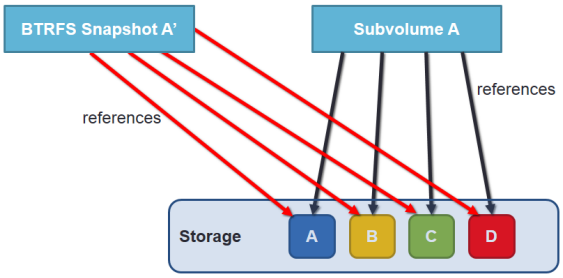


Figura 5.2: Exemplo do funcionamento de um snapshot em *BTRFS*.

Após isto, o sistema CoW garante que as alterações que sejam feitas ao *snapshot* não são vistas pelo sub-volume que lhe deu origem ou vice-versa. É fácil e muito rápido fazer um *rollback* para um determinado *snapshot*, simplesmente descartando o *snapshot* indesejado e renomeando aquele ao qual se pretende regressar de forma a ficar com o mesmo nome do que foi descartado.

O *Btrfs* tem ainda outras funcionalidades e características tais como **a)** um novo tamanho máximo para um ficheiro (16 *exabytes* [1] para o sistema *Btrfs*, limitados a 8 *exabytes* em Linux), **b)** compressão de dados (*zlib* e LZO), **c)** otimizações para SSDs, **d)** *backups* incrementais, **e)** desfragmentação do sistema de ficheiros, e **f)** deduplicação de dados *out-of-band* (realizada após concluídas as escritas, e não durante os próprios *writes*).

Suporte para múltiplos dispositivos de armazenamento

Um dos principais objetivos do Btrfs é suportar a adição e remoção de dispositivos (i.e., discos lógicos ou físicos) com o SF ativo. O princípio que está na base desta funcionalidade é o seguinte: quando um dispositivo é adicionado ao Btrfs, este organiza o espaço livre no dispositivo em múltiplos pedaços (*chunks*). Cada pedaço pode ser uma parte diretamente mapeada de um dispositivo da *pool*, uma parte do dispositivo que é *mirrored* para outra localização ou uma combinação de várias partes de vários dispositivos usando uma técnica RAID [13]. Para além disso o Btrfs suporta, usando o mecanismo de *seeding*, numa única árvore e simultaneamente, dispositivos em modo *read-only* e *read-write*.

O Btrfs suporta os níveis RAID 0, RAID 1, RAID 10, RAID 5 e RAID6 [1].

Integridade dos dados

De forma a identificar erros que possam passar sem ser detetados pelos mecanismos *hardware* dos sistemas de discos, o Btrfs calcula *checksums* para toda a informação: dados e metadados. Para isso utiliza um algoritmo CRC32 para calcular os *checksums* (mas o

[1] 1 *exabyte* = 1000 *petabytes* = 1.000.000 *terabytes*

formato está preparado para aceitar outros tipos de algoritmos de *checking*), e cada bloco da árvore tem um *checksum* no seu cabeçalho. Para além disso, o Btrfs tem os metadados duplicados.

O Btrfs pode iniciar uma verificação sob todo o sistema de ficheiros executando um processo (*scrub job*) que verifica a integridade de todo o sistema e relata e tenta automaticamente reparar os blocos danificados que encontrar. Se detetar que existe um erro de *checksum* enquanto lê um bloco, tenta obter uma cópia desse bloco noutro dispositivo (caso se esteja a utilizar técnicas de RAID com redundância - níveis superiores a 0).

A funcionalidade de *scrubing* é utilizada para prevenir a corrupção silenciosa de dados (erros que passam despercebidos ao *firmware/hardware* do disco e ao sistema de operação, que podem ter múltiplas origens, sejam estas *hardware* - variações de tensão nas fontes de alimentação, vibrações externas, bombardeamento da RAM por partículas alfa - ou *software - kernel crashes*, etc.).

5.2 ZFS

O sistema de ficheiros Z (*Z File System*) ou ZFS [9], foi desenvolvido na Sun Microsystems e incluído no sistema de operação Solaris. Posteriormente, a Sun promoveu o desenvolvimento de um *port* para Linux, sendo a primeira versão estável datada de Março de 2013. Com a aquisição da Sun pela Oracle é criado um projeto *open-source*, o OpenZFS. Tal como o Btrfs, também o ZFS utiliza ***copy-on-write*** e permite criar **snapshots** do sistema de ficheiros.

O ZFS tem três objetivos principais: [7]

- **Integridade dos dados:** Baseada em *checksums* calculados aquando das escritas no sistema de ficheiros. Quando há uma leitura, o *checksum* é calculado outra vez, e comparado com o valor armazenado anteriormente na escrita do item de dados; se os valores não coincidirem, o ZFS tenta corrigir automaticamente a situação recorrendo a informação redundante;

- **Armazenamento *pooled*:** O ZFS mantém uma *pool* de armazenamento formada pelas contribuições dos diversos dispositivos físicos e lógicos (o ZFS inclui uma camada de gestão de volumes lógicos). O espaço da *pool* está disponível para todo o sistema de ficheiros e pode ser aumentado por adição de mais dispositivos.

- **Performance:** O ZFS oferece múltiplos mecanismos de cache para aumentar o desempenho: a) o **ARC** é uma cache avançada baseada em memória, b) o **L2ARC** é uma cache de leitura baseada em disco e c) o **ZIL** é uma cache de escrita síncrona baseada em disco;

5.2.1 Destaques da arquitetura e realização

O ZFS foi desenhado para garantir integridade dos dados utilizando *checksums* baseados em memória. Num sistema de ficheiros tradicional com *checksums* estes apenas protegem contra *bit rot* [2] pois o *checksum* é armazenado no próprio bloco. Mas no ZFS o *checksum* é armazenado no apontador do bloco em vez de no próprio bloco (juntos aos dados). Assim, além de proteger contra *bit rot*, também previne problemas como: **a)** escritas fantasma (escrita anterior não ficou armazenada em disco), **b)** leituras e escritas desorientadas (o disco acede ao bloco errado), **c)** erros de paridade em *Direct Memory Access* (erros ao aceder à memória RAM), **d)** bugs do controlador ou **e)** re-escritas acidentais [8].

É um sistema de ficheiros baseado em transações CoW onde as operações são atómicas, ou seja, a operação é executada por completo com sucesso, ou nada é executado de todo. Isto significa que em caso de faltas graves (*crash*, falha de energia, etc.) o sistema pode facilmente recuperar da corrupção de dados [9].

O ZFS não utiliza volumes como um sistema de ficheiros tradicional; em vez disso o sistema partilha uma *pool* de armazenamento que consiste num ou mais dispositivos físicos de armazenamento, que até podem ser de diferentes tecnologias - a ***ZFS Hybrid Storage Pool*** (HSP) permite, por exemplo, combinar RAM, SSDs e discos magnéticos numa única *pool* de armazenamento.

As *pools* são constituídas por dispositivos virtuais chamados *vdevs* que podem ser físicos (um disco rígido por exemplo) ou lógicos (um grupo de vdevs físicos). [26]

5.2.2 Funcionalidades mais relevantes

- ***Adaptive Replacement Cache* (ARC)**

  A ARC é uma cache avançada de alta velocidade suportada em memória RAM, onde são inicialmente armazenados os dados provenientes de operações de escrita, segundo uma política MRU (*Most Recently Used*); é naturalmente também o primeiro local onde procurar dados para satisfazer uma operação de leitura [20].

- ***Level-two ARC* (L2ARC)**

  A L2ARC é uma cache de nível 2, a nível do disco, e funciona como uma extensão da ARC. Todos os dados que seriam colocados no ARC mas não o são por esta estar cheia, são colocados na L2ARC. Para ser eficiente, convém que o disco que suporta a L2ARC tenha uma boa performance (por exemplo, um SSD) [20].

- ***ZFS Intent Log* (ZIL)**

  O ZIL é utilizado para gerir operações de escrita **síncronas**, cujo protocolo requer que os dados sejam colocados num dispositivo de armazenamento não volátil antes das operações serem reconhecidas pelo sistema como já tendo terminado. Um

[2] Quando um bit é invertido devido à deterioração do disco

exemplo são as escritas em NFS ou CIFS (a.k.a SMB); outro, é o dos sistemas transacionais, nos quais uma transação só pode avançar após outra, anterior, ter terminado e armazenado os dados de forma persistente.

Numa arquitetura ZFS com ARC (em RAM) e ZIL (que por questões de desempenho deve estar suportado em SSDs), todas as operações de escrita (síncronas ou assíncronas) são primeiro enviadas para a ARC (armazenamento volátil), mas as escritas síncronas são também enviadas ao ZIL antes de serem dadas por terminadas, reduzindo a latência das operações quando comparadas com a latência que seria observada sem ZIL, situação na qual os dados seriam escritos nas *pools*.

Apenas são feitas operações de leitura ao ZIL caso exista *crash* do sistema ou falhas de eletricidade, caso este em que o ZFS vai ler os dados ao ZIL e escrevê-los no disco, que é a sua localização natural; quando as operações finalmente chegam ao disco magnético, o conteúdo do ZIL é tornado irrelevante.

- ***Copy-on-Write e Snapshots***

  O zfs utiliza CoW, o que lhe permite criar *snapshots* do sistema de ficheiros quase instantaneamente. A forma como estes são criados é semelhante à do Btrfs, pois é baseada no CoW e nas referências aos blocos, mantendo em disco os blocos marcados como sendo de *snapshots*, que de outra forma seriam descartados.

O ZFS inclui ainda: **a)** o seu próprio *software* para RAID, designado RAID-Z, e que é semelhante ao esquema de distribuição por paridade do RAID5); **b)** sendo um sistema de 128-bit em vez de 64 bit como o Btrfs, o ZFS pode endereçar muito mais informação; **c)** inclui de-duplicação de dados; e **d)** permite cifrar o sistema de ficheiros.

5.3 Quadro comparativo

Apresenta-se na figura 5.3 um o quadro comparativo, onde se resumem algumas funcionalidades dos sistemas de ficheiros Btrfs e ZFS. Optou-se por incluir também o sistema de ficheiros VMFS3 da VMware e o Ext4, a versão mais recente do sistema de ficheiros mais utilizado no sistema operativo Linux, para conseguir uma panorâmica mais geral de sistemas de ficheiros muito usados para suportar máquinas virtuais - que é, afinal, o objetivo do nosso trabalho.

Sistema de Ficheiros	Tamanho máximo do nome dos ficheiros	Tamanho máximo de um ficheiro	Tamanho máximo de um volume	Listas de controlo de acesso	Checksum
Btrfs	255 bytes	16 exabytes	16 exabytes	Sim	Sim
Zfs	255 bytes	16 exabytes	2^{18} exabytes	Sim	Sim
VMFS3	128 bytes	2 terabytes	64 terabytes	Não	Não
Ext4	255 bytes	16 gigabytes	1 exabyte	Sim	Não

Sistema de Ficheiros	Block Journaling	Metada Journaling	Snapshotting	Cifrado	De-duplicação
Btrfs	Sim	Sim	Sim	Não, mas planeado	Sim
ZFS	Sim	Não	Sim	Sim	Sim
VMFS3	Sim	Sim	Não	Não	Não
Ext4	Sim	Sim	Não	Sim, experimental	Não

Figura 5.3: Quadro comparativo dos sistemas de ficheiros [34].

Capítulo 6

Implementação e testes de funcionalidade

6.1 Utilização da técnica de *snapshotting* em Máquinas virtuais

Atualmente existem, nas várias camadas de uma infraestrutura VDI, formas distintas de criar *snapshots* (de máquinas virtuais): usando o hipervisor (também utilizam estruturas do tipo CoW), usando funcionalidades disponibilizadas por muitos dos *arrays* de discos RAID disponíveis no mercado (implementadas por *software add-ons* geralmente muito dispendiosos), ou por gestores de volumes (o LVM pode criar *snapshots* das partições que cria [6]).

A nossa tese é a de que o sistema de ficheiros que é responsável por armazenar as VMs, possa ele mesmo fornecer essa funcionalidade de criar *snapshots* e, com isso, a) ao retirar essa responsabilidade ao hipervisor e, b) que o desempenho seja melhorado. Intuitivamente (a) é conseguido; já (b) pode ser defendido com a seguinte argumentação: uma formulação, realizada ao nível do hipervisor, que seja genérica para todos os sistemas de ficheiros não poderá, pensamos, ser tão eficiente como uma que seja específica de um dado SF.

Dos dois candidatos anteriormente estudados por permitirem criar *snapshots* ao sistema de ficheiros, Btrfs e ZFS, para a presente prova de conceito, selecionou-se o Btrfs; optou-se também por disponibilizar a funcionalidade de *snapshots* nativos ao hipervisor KVM, sem detrimento desta poder também vir a ser integrada, no futuro, noutros hipervisores. Caso a integração seja efetuada com sucesso, o objetivo seguinte é usá-la como alavanca para disponibilizar *linked-clones* baseados nos referidos *snapshots* nativos ao sistema de ficheiros.

Pensamos que o seguinte cenário ajuda a clarificar os nossos propósitos: existe um *template* de VM com as configurações básicas estabelecidas; quando for necessário criar uma instância desse *template* (i.e., uma nova VM), a) faz-se um *snapshot* ao diretório onde

está armazenado o *template*, b) alteram-se as configurações da instância de forma a que estas reflitam a natureza da "nova VM" criada, e c) executa-se a VM no posto de trabalho que a requisitou. Quando o utilizador fizer *log off*, a instância é destruída (assim como o *snapshot* que lhe deu origem). Pretende-se portanto, criar VMs não-persistentes.

6.1.1 Snapshots de VMs no Btrfs

Os *snapshots* realizados utilizando o sistema de ficheiros Btrfs com o propósito de criar *linked-clones* são executados unicamente ao disco virtual do *template* e, consequentemente, quando este está *off-line* (tanto quanto nos foi possível perceber, o KVM não tem *verdadeiros templates*, não no sentido, por exemplo, do VMware - onde um *template* não pode, de forma nenhuma, ser executado). Para executar a *snapshot*, é necessário que o disco da VM (e apenas os eu disco) esteja contida num sub-volume criado previamente. Após esse passo, é criado o disco virtual que a VM irá utilizar futuramente (que teoricamente pode ser de qualquer tipo, raw, qcow2,...) nesse sub-volume.

É feita a instalação da VM normalmente indicando o *path* do disco virtual previamente criado.

Finalmente, sempre que for necessário criar um *snapshot*, apenas é necessário utilizar o comando btrfs correspondente tal como indicado em **??**. Nesse momento, existiram duas pastas com dois *paths* diferentes para o mesmo ficheiro, a pasta original e a nova pasta/subvolume que contém o mesmo nome da *snapshot*.

Ambos podem ser escritos (pode ficar *read-only* se indicarmos o argumento quando executamos o comando de *snapshot*) e para restaurar qualquer um deles,com a VM *offline* alterarmos o *path* para o disco virtual no ficheiro de configuração XML correspondente à VM.

Após o arranque da VM, esta irá utilizar o disco virtual escolhido, sem que o hipervisor saiba se está a aceder a uma *snapshot* ou não. Ou seja a execução do *snapshot* em si é feita e tratada apenas pelo sistema de ficheiros, sem qualquer influencia do hipervisor que se limita a chamar um *script* ou API que é responsável pelo *snapshot*, tal como demonstrado na figura 6.1, sendo este, o objetivo principal desta dissertação.

Para criar a *snapshot* manualmente, os seguintes passos têm de ser executados:

- Requisitos -

a) Ter um sistema operativo Linux com KVM/QEMU e Libvirt instalados;

b) Conter pelo menos uma partição formatada em Btrfs, onde serão colocados os discos virtuais da VM;

c) Permissões de acesso para edição e utilização de máquinas virtuais;

- Preparação do sistema -

Antes de se poder criar snapshots, é necessário preparar o ambiente. Para isso têm de ser feitas algumas preparações:

a) Criar um subvolume no Btrfs onde será colocado o ficheiro do disco virtual da VM Para isso utiliza-se o seguinte comando:

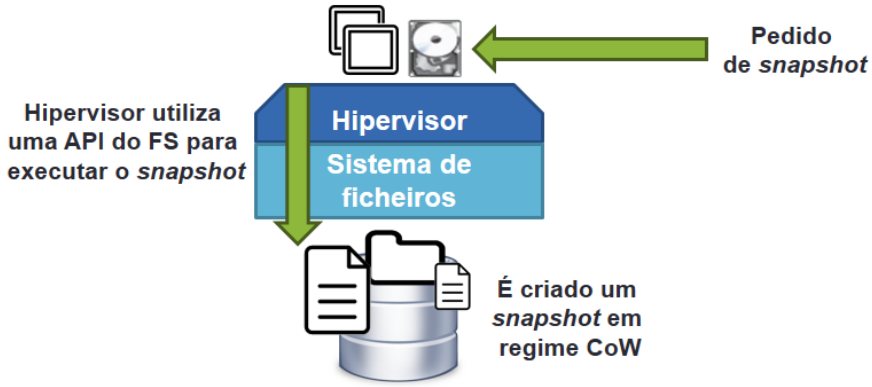


Figura 6.1: Esquema da execução de um *snapshot*.

```
1 sudo btrfs subvolume create [<dest>]/<nome>
```

Se não for indicado o <dest> então o subvolume será criado na diretoria corrente.

b) Criar manualmente o disco virtual, utilizando o QEMU Para criar um ficheiro de disco virtual utilizando o QEMU, utiliza-se o seguinte comando:

```
1 qemu-img create -f raw image_file 12G
```

Qualquer tipo de disco virtual (raw, qcow2, ...)é aceitável, uma vez que será o sistema de ficheiros a criar as *snapshots* e não o hipervisor.

c) Instalar a máquina virtual, indicando como disco principal o ficheiro de disco virtual criado no passo anterior.

Para este passo, qualquer *software* de gestão de máquinas virtuais é aceitável, desde que permita indicar o disco virtual. Para este tutorial, foi utilizada a ferramenta *virt-manager* que contém um GUI que pode facilitar as operações de instalação, *start* e *shut-down* de VMs.

- Criar e utilizar *snapshots* em Btrfs - Após estarem cumpridos os requisitos e os passos acima, o sistema está pronto a criar *snapshots* ao disco virtual da VM. Para se criar uma *snapshot*, utiliza-se o comando do Btrfs:

```
1 btrfs subvolume snapshot <source> [<dest>]/<name>
```

Este comando irá criar um novo subvolume, *snapshot* do volume indicado em <source> . A partir deste momento existem duas diretorias que incluem o ficheiro de disco virtual da VM, cujos blocos estão partilhados entre os dois subvolumes. Ambos podem ser reescritos e as modificações de um, não são visiveis no outro.

Podem ser criados tantos *snapshots* quanto se querem, sendo que para cada um, será criado um novo diretório. É possível apagar *snapshots* utilizando o comando:

```
1 btrfs subvolume delete [<path>]/<name>
```

Para remover um *snapshot*, simplesmente apagamos o subvolume que o "contém". Não é possível apagar subvolumes que contenham outros subvolumes, para fazer isso é necessário apagar os restantes primeiro.

Após isto, para se poder mudar de um *snapshot* para outro, modificamos o XML criado pelo Libvirt, correspondente à VM de que criamos o *snapshot*. Para editar o XML podemos utilizar o utilitário virsh com o seguinte comando:

```
1 virsh edit <domain>
```

No XML, pretendemos alterar o *path* do disco virtual no nó **<source>** para o disco virtual que pretendemos (seja o original, seja a *snaspshot*, indicamos o *path* do subvolume e no fim o ficheiro de disco, por exemplo:

```
1 /home/nuno/Image_Data/VM_Test/lubuntu.img
```

O utilitário virsh utiliza o **vi** para editar o ficheiro XML. Para conhecer que domínios (VMs) estão instalados, utiliza-se o comando *sudo virsh list –all*

Por fim, salvamos o ficheiro e utilizamos, por exemplo, o *virt-manager* para arrancar a VM que utilizará o disco virtual indicado pelo XML. Uma vez que ambos pertencem à mesma VM, o sistema KVM não será capaz de dizer se está a aceder ao disco original ou a uma *snapshot*.

Como dito anteriormente, ambos o disco virtual original e a *snapshot* são editáveis e as modificações não vão ser refletidas nos outros subvolumes. Existe a vantagem da transparência e da facilidade e rapidez com que os *snapshots* são criados. No entanto, podem existir problemas a nível de *performance* em resultado do processo como o Btrfs coloca os dados em cache dos blocos partilhados pelas várias *snapshots* do Btrfs.

6.1.2 Exportar em NFS

Outro objectivo será disponibilizar a imagem do disco virtual a nível de uma rede de computadores. Para alcançar esse objetivo, escolheu-se utilizar o utilitário NFS (um protocolo de um sistema de ficheiros distribuído), que permite exportar diretórios locais para que estes sejam acessíveis a partir de uma rede de computadores, como se esse diretório estivesse em armazenamento local.

Para exportar para NFS, primeiro é necessário que este esteja instalado no sistema Linux onde se encontra o disco virtual (servidor nfs) e que as devidas configurações sejam efetuadas (permissões do NFS, firewalls, etc). De seguida, as seguintes configurações são necessárias, de forma a que uma máquina virtual consiga aceder ao disco virtual exportado e para que possa utiliza-lo.

No ficheiro de configuração */etc/exports* (sudo vi /etc/exports) utiliza-se a seguinte configuração:

```
1 /home/nuno/Image_Data/  192.168.1.104(rw,sync,no_root_squash,no_subtree_check)
```

- **/home/nuno/Image_Data** = Diretoria partilhada com permissões de leitura e escrita;
- **192.168.1.104** = Endereço IP do cliente NFS. Em vez do IP, pode colocar-se um *, que permite que qualquer IP aceda ao directorio exportado;
- **rw** = Permitir operações de leitura e escrita na directoria exportada;
- **sync** = responder aos pedidos de acesso apenas depois das escritas anteriores terem sido confirmadas em armazenamento;
- **no_root_squash** = Permitir que um utilizador root numa máquina cliente NFS tenha as mesmas permissões na diretoria exportada, que um utilizador root no servidor NFS onde se encontra a diretoria partilhada;
- **no_subtree_check** = Quando um servidor NFS exporta um subdirectorio de um sistema de ficheiros local, mas os restantes ficheiros e diretórios não são exportados, o servidor NFS deve verificar se cada pedido NFS corresponde a ficheiros apenas no directorio exportado. Esta verificação chama-se *subtree_check*; aconselha-se, por razões de desempenho, a sua não utilização.

Nota importante, garantir que não há espaços entre as virgulas e os argumentos, caso contrário durante o *export* surgirá uma mensagem de erro.

Por fim, executar o comando nfs *exportfs -a* para partilhar as diretorias indicadas no ficheiro de configurações. Para verificar se a diretoria foi exportada, utilizar o comando *showmount -e 127.0.0.1*, o *output* indicará a diretoria exportada e que IPs lhe podem aceder.

Se for necessário alterar as configurações do ficheiro *exports*, utilizar os comandos *service nfs stop* e *service nfs restart* para reiniciar o nfs. De seguida, do lado do **cliente**, é necessário primeiro criar a diretoria onde será montada a diretoria exportada. Utiliza-se o comando:

```
1 mkdir -p /var/nfs_share
2 chmod 777 /var/nsf_share
```

As permissões *777* dadas pelo *chmod*, devem ser especificadas para o utilizador especifico que vai utilizar a diretoria, mas não esquecer que neste caso são necessárias permissões de *root* e que têm de ser as mesmas que estão na directoria exportada.

De seguida, montar a diretoria exportada na que foi criada, por exemplo:

```
1 mount  192.168.1.112:/home/nuno/Image_Data/ /var/nfs_share
```

Por último, é necessário alterar o ficheiro XML da VM para procurar o disco virtual na diretoria exportada. Editamos o ficheiro com *sudo vi /etc/libvirt/qemu/lubuntu.xml* e em *Devices/Disk/souce* colocamos:

```
1 'var/nfs_share/VM_Test/lubuntu-img'
```

Com todas as preparações efetuadas, apenas é necessário arrancar o *virt-manager* (ou outro de preferência) e arrancar a máquina virtual. Caso surja uma mensagem de erro durante a ativação que a rede virtual *default* não está ativa, basta utilizar o comando *virsh net-start default* (*default* é neste caso o nome da rede virtual criada, dependendo de como foi criada, poderá ter outro nome).

6.2 iCBD - conceito do produto

O projeto iCBD (*Infrastructure for Client-Based (virtual) Desktop (Computing)*) é um produto de *software* criado na Reditus S.A. que tem por objetivo criar uma infraestrutura do tipo VDI para execução de *desktops* virtualizados nos postos de trabalho. Segue-se uma descrição conceptual do produto.

6.2.1 Funcionamento

As imagens das VMs (*desktops* virtuais) que os utilizadores pretendem utilizar), estão armazenadas do lado do servidor que utiliza o sistema de ficheiros Btrfs. Quando o utilizador inicia o seu posto de trabalho, um sistema operativo Linux mínimo é executado a nível local. Este sistema inicial, oferece ao utilizador através de um menu o(s) ambiente(s) que este pode executar (por exemplo, *Windows*, Fedora, Ubuntu) e permite que este escolha o qual quer lançar.

Após a escolha, é iniciado o processo de *boot* da VM (descrito mais adiante) que irá criar/trazer a VM para o posto de trabalho onde será feita a execução da mesma, baseado num conceito chamado *Virtual Desktop Infrastructure*, *client side* tal como descrito na figura 6.2.

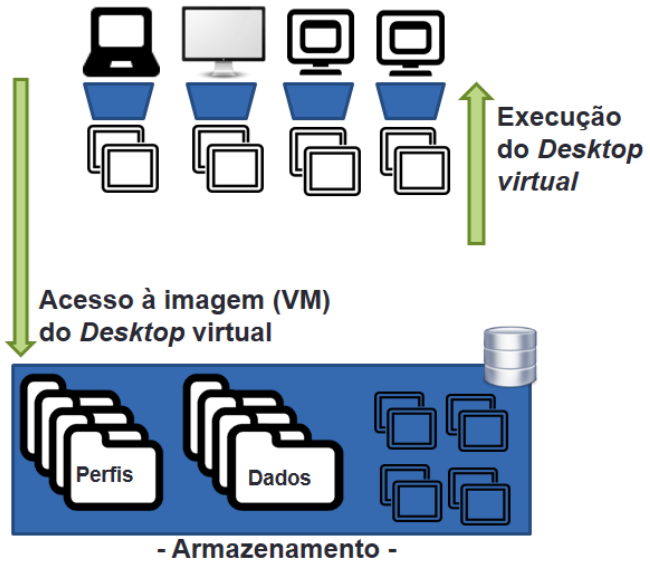


Figura 6.2: Esquema de uma VDI baseada em cliente.

6.2.2 Implementação

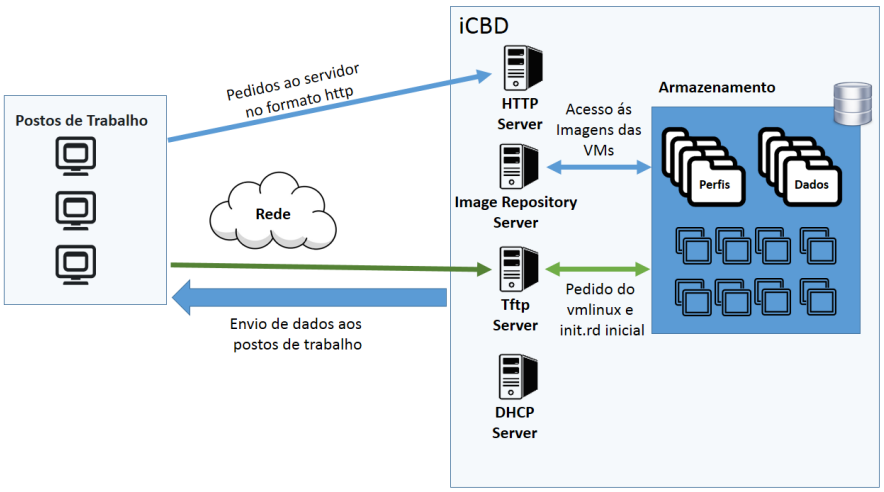


Figura 6.3: Arquitetura física do iCBD.

O processo de comunicação entre o posto de trabalho e o servidor é feito utilizando mensagens com o protocolo HTTP, permitindo flexibilidade e eficiência na comunicação das mensagens. Ao iniciar-se o processo de *boot* de uma VM, os seguintes passos são executados (cf. Capítulo 2):

- É feito um **PXE *boot*** e o utilizador escolhe uma opção no menu que recebeu; da imagem Linux selecionada, extrai-se o *kernel vmlinux* e o *initrd* que é enviado ao cliente por TFTP;
- É executado o *VMkernel* eo *initrd*;
- É executado o *init* para identificação das NICs e carregar os *drivers* correspondentes ao *hardware* específico do posto de trabalho.
- Esse mesmo *init* utiliza HTTP para executar *wget* para descarregar os restantes *scripts* iCBD necessários (estes *scripts* podiam ser incluídos no *init*, mas de forma a torná-lo o mais pequeno possível e não ter de se editar tantas vezes durante o processo de desenvolvimento, optou-se por esta opção).
- É executado o *rc0* que cria o *setup* do ambiente que vai correr a imagem (entenda-se, disco virtual) escolhida. Este processo, é nesta fase feito em regime *read-only* e pode ser feito através de NFS, iSCSI ou outro que venha a ser suportado.
- Por último é criada uma estrutura de dados em forma de árvore por cliente (onde o *entry point* será o *Mac Address* do cliente), cujo o objetivo é tornar a imagem *read and write*.

Para tornar a imagem *rw* ao mesmo tempo que se mantém os dados da imagem original intactos, é utilizada uma funcionalidade do Btrfs chamada ***device - seeding***. O objetivo original desta funcionalidade seria criar uma camada de sistema de ficheiros semelhante ao *unionfs/aufs*, mas neste contexto vai ser utilizado para criar um *linked-clone* (escritas *rw*) a partir de um *template* (*device read-only*) [5].

Para se utilizar *device seeding*, utiliza-se a linha de comandos para marcar com uma *flag* um *device* Btrfs como sendo uma *seed*. A partir de agora, este *device* tornou-se *read-only* e os seus dados não podem ser alterados. Após isso, ainda utilizando a linha de comandos, podemos adicionar outros Btrfs *devices* à *seed*, para que as operações de leitura são feitas sobre tanto sobre a *seed* como sobre o *device* "descendente", mas as operações de escrita são feitas apenas sobre o descendente (que é *rw*).

```
1  #Initial device setup for testing
2
3  losetup /dev/loop0 /mnt/gentoo/btrfs-test-1
4  losetup /dev/loop1 /mnt/gentoo/btrfs-test-2
5
6  # Mount loop0 as /mnt/test/ and creation of two files on loop0
7  mount /dev/loop0 /mnt/test/
8  echo a > /mnt/test/a
9  echo b > /mnt/test/b
10 umount /mnt/test
11
12 # Adding Seed flag to device loop0. Loop0 is now a Seed Device
13 btrfstune -S 1 /dev/loop0
14
15 mount /dev/loop0 /mnt/test
16 -> mount: block device /dev/loop0 is write-protected, mounting read-only
17
18 #Adding another device (loop1) pointing at loop0 (the seed)
19 btrfs device add /dev/loop1 /mnt/test
20
21 # listing the seed device directory contents
22 ls /mnt/test
23 -> a  b
24
25 # If a write operation os executed on seed, a Read-Only error is returned
26 echo c > /mnt/test/c
27 -> bash: c: Read-only file system
28
29 # After adding a new device to the seed, a remount is needed for the changes
       to take effect
30 mount -o remount,rw /mnt/test
31 # Or
32 umount /mnt/test
33 mount /dev/loop1 /mnt/test
34
```

```
35  # After the remount, it's possible to execute write operations again on the
        device
36  echo c > /mnt/test/c
37  echo d > /mnt/test/d
38
39  # To show what is happening, the seed device is remounted again
40  umount /mnt/test
41  mount /dev/loop0 /mnt/test
42  -> mount: block device /dev/loop0 is write-protected, mounting read-only
43
44  # If the seed directory contents are listed, only the original files can be
        seen (at the time when the device was flagged as a seed)
45  ls /mnt/test
46  -> a  b
47  # But if the pointing device is mounted again (loop1) and then if we list
        its
48  # directory contents, we notice that not only the original files of the seed
         device can be
49  # seen (loop0), but also the files created at the pointing device can be
        seen (loop1)
50  umount /mnt/test
51  mount /dev/loop1 /mnt/test
52  ls /mnt/test
53  -> a  b  c  d
```

Ou seja, obtemos desta forma uma estrutura semelhante à necessária para criar *linked-clones*, utilizando a *seed* para guardar a imagem original da VM, e adicionando novos *devices* que podem guardar os "deltas" das operações de escrita. As operações de leitura, têm em conta o *device* adicionado e a *seed* para a qual aponta (sendo que vários *devices rw* podem ter um único *device* como *seed*), oferecendo a possibilidade de vários *linked-clones* em simultâneo.

A árvore acima descrita irá, para cada cliente (numa subdiretoria identificada pelo seu MAC *address*), indicar onde está/estão o(s) seu(s) ambiente(s) modificado(s), podendo portanto existir múltiplas instâncias simultaneamente. Estando esta parte da árvore exportada por NFS, seria simples ao cliente montar a imagem da instância e executá-la. Infelizmente verificamos que tal conduz a uma sobre-utilização da RAM do servidor de imagens.

A razão dessa sobre-utilização prende-se com o facto do sistema de cache do Btrfs não partilhar os blocos entre as *snapshots* (subvolumes). Ou seja, se existirem várias *snapshots* que apontam para os mesmos dados e vários clientes lhes tentarem aceder, por cada cliente os mesmos dados são transportados para a cache, resultando numa ocupação excessiva da RAM. Assim, resolveu-se o problema fazendo o *seeding* e os *loop devices* ao nível de cada cliente.

Por fim, o iCBD oferece ainda uma **ferramenta de administração** das imagens (*templates*). Ao arrancar a imagem utilizando esta ferramenta, o administrador pode fazer as

alterações que quiser à imagem (por exemplo, *updates*, instalação de *software*, modificação de configurações).

Quando fizer *shutdown* da imagem, é oferecida ao administrador 3 hipóteses: **1)** descartar alterações, em que todas as alterações será esquecidas, **2)** *update*, onde as alterações irão afetar a imagem utilizada para criar os *linked-clones*, que a partir desse momento irão utilizar essa imagem modificada como base, ou **3)** guardar alterações para utilização mais tarde, aqui as configurações são guardadas, mas os *linked-clones* continuarão a utilizar a imagem base antiga.

Este processo de administração é implementado utilizando o sistema de *snapshoting* do Btrfs. Pode ser utilizado aqui, porque apenas o administrador vai utilizar este sistema, não havendo problemas de performance causados pela cache.

Para cada imagem existe um índice de *snapshots* de 1..*i* (a que correspondem diretorias nomeadas de 1..i) e uma diretoria *w* que é o utilizada para executar os *linked-clones*. Quando se pretende aplicar alterações à imagem *w*, é criada uma nova *snapshot* i+1 (e por consequência uma nova diretoria i+1).

Após isto, os novos *linked-clones* utilizam a imagem *w* para se basearem, no entanto é importante manter as versões antigas (disponíveis nas diretorias 1..i) para os *linked-clones* que ainda não foram desligados desde o *update* poderem utilizar, ou mesmo caso o próprio administrador precise uma versão anterior da imagem.

Estes *snapshots* podem ser apagados se necessário, sendo que apenas a diretoria *w* é necessária para funcionamento de novos *linked-clones*.

6.3 Testes de funcionalidade

Sendo o iCBD um produto VDI baseado em cliente, a parte do processamento e execução das aplicações é feita do lado do posto de trabalho. O servidor não necessita portanto, de alta capacidade de recursos físicos como a memória RAM ou o CPU, uma vez que estes apenas são necessários para administração das VMs, *boot* inicial dos ambientes e posterior comunicação com os postos de trabalho.

Nestes moldes, a maior carga de "trabalho" do lado do servidor será feita sobre o armazenamento das VMs, ou seja, nos discos rígidos físicos do servidor que contêm os discos virtuais das VMs sendo aqui que se vai encontrar o principal *bottleneck* da performance.

Assim como foi escrito no capítulo 3 (Armazenamento - o problema), quando se fala de performance a nível de armazenamento existem dois pontos principais de medida: 1) os IOPS (*Input/Output Operations Per Second*) e 2) o espaço de armazenamento.

Sobre o segundo ponto, o espaço de armazenamento vai depender principalmente do investimento financeiro por parte da organização que utilizar a solução VDI. Este investimento é feito tendo em consideração tanto o número de imagens de VMs disponível, como o número de VMs que são criadas a partir destas imagens, pois apesar do espaço ocupado pelas VMs ser mínimo (começando com alguns MBs) estas podem atingir várias GBs no caso de VMs persistentes.

Algo igualmente importante sobre o espaço de armazenamento, é ter em conta que quanto maior for o armazenamento de um disco rígido, mais barato fica (preço por *Gigabyte*), mas também menos poder de performance irá ter (a nível de IOPS), sendo o contrário também verdade. Quanto maior a performance do disco (por exemplo, os SSDs), mais dispendioso será o preço por *Gigabyte* e menor será o espaço disponível.

Assim sendo, o pressuposto de uma infraestrutura VDI seria, a utilização de discos físicos com uma maior disponibilidade de espaço mas menor performance para armazenamento das imagens dos discos virtuais e a utilização de discos físicos com menor espaço disponível mas maior performance (como é o caso dos SSDs) para efeitos de *cache* dos discos anteriormente referidos.

Sobre o primeiro ponto, além da performance do próprio *hardware* do disco físico (e da forma como são organizados, NAS, SAN, LUNs) é também importante ter em conta aquilo que é suposto ser lido e escrito dos discos físicos, ou seja, os discos virtuais das VMs. Existem diversos formatos em que se pode guardar os discos virtuais, desde o formato RAW, o já falado QCOW2 da QEMU que permite *Copy-on-Write*, o VHD da *Microsoft*, o VDI da *Virtualbox* ou o VMDK da *VMware*, cada um deles com diferentes características, vantagens e desvantagens.

Sendo que foi eliminado o requisito de utilizar um determinado tipo de disco virtual que permita criar *snapshots* (deixam de ser necessários os ficheiros do tipo qcow2 para criar *snapshots, uma vez que essa funcionalidade é oferecida nativamente pelo sistema de ficheiros*), torna-se bastante importante criar testes sobre a infraestrutura VDI que permita ao administrador escolher qual o tipo de disco virtual que lhe oferece melhor performance.

6.3.1 Testes ao iCBD

Quando uma infraestrutura VDI é implementada corretamente e executando os passos certos, é possível retirar várias vantagens da mesma, como já foi visto nos capítulos anteriores. No entanto, se for implementada incorretamente ou não cumprindo os passos todos necessários, VDI pode originar deslizes orçamentais, utilizadores finais insatisfeitos com a performance do seu *desktop* ou uma complexidade muito maior na gestão da infraestrutura do que seria de esperar.

Assim sendo, e para evitar estes casos, existe um último passo muito importante em VDI que mitiga (e pode mesmo prevenir) muitos destes problemas, que estão na génese de muitos projetos VDI caírem por terra ainda na fase de lançamento, a execução de testes de performance da infraestrutura.

O iCBD, sendo não só um projeto novo mas também ele baseado em VDI, necessita de uma base de testes que verifiquem a sua performance e robustez em ambiente real. Infelizmente, não houve tempo (nem *hardware* disponível) para executar os testes em ambiente real, mas segue-se um possível plano de testes e sugestões de ferramentas para os executar, quando num futuro próximo for possível executa-los:

6.3.1.1 Identificar os casos de uso

Para melhor interpretar se os resultados conseguidos num determinado plano de testes são ou não positivos para um determinada aplicação, é fundamental existirem e serem bem documentados casos de uso específicos que reflitam os requisitos dessa aplicação. Por exemplo, os casos de uso (e por consequência requisitos) de uma VDI numa universidade serão sempre diferentes dos casos de uso numa organização, que serão diferentes de um hospital ou de um sistema de emergência dos bombeiros.

Uns projetos irão requerer uma enorme capacidade de armazenamento para os dados, outros que o sistema tenha uma performance acima da média para não haver qualquer tipo de atrasos ou que seja possível executar o *login* de um grande número de utilizadores em pouco tempo. Qualquer que seja o requisito, se este não for cumprido é necessário reavaliar o equipamento que se adquiriu, ou a infraestrutura de rede ou procurar onde se pode melhorar ou modificar para que esse requisito seja cumprido.

À data de escrita desta dissertação, ainda não foi possível ter acesso ao *hardware* necessário para executar os testes propriamente ditos. Assim sendo, seguidamente propõem-se um conjunto de testes que poderiam medir a performance do produto e por consequente poder efetuar decisões, por exemplo, tanto a nível da infraestrutura de rede (oferecer maior largura de banda) como dos tipos de discos virtuais escolhidos (por exemplo, escolher o tipo RAW em detrimento do VMDK) ou mesmo do *hardware* dos discos físicos.

6.3.1.2 Performance da infraestrutura

Para melhor obter conclusões sobre os testes, propõem-se que se faça a comparação dos diferentes tipos de discos virtuais tendo como parâmetros de comparação a **largura de banda**, **latência** e **IOPS** necessários quando se efetua os seguintes tipos de operações: 1) leitura sequencial, 2) leitura aleatória, 3) escrita sequencial, 4) escrita aleatória [10].

Todos estes tipos de operações são necessários e importantes em diferentes processos da rede VDI (leitura aleatória para arranque simultâneo de várias VMs ás 09h00 da manhã numa organização por exemplo).

Para executar os testes, seria utilizada uma ferramenta de ***IO Micro-Benchmark*** chamada ***fio*** *(Flexible IO)*, uma ferramenta *open-source* que tem por objetivo criar com facilidade testes de *benchmark* para dispositivos de armazenamento.

Com a ajuda desta ferramenta, será possível simular vários tipos de *IO workload* que nos permita testar a performance dos diferentes tipos de discos virtuais e daí tirar conclusões.

A *fio* permite afinar vários parâmetros como tipo de IO, o tamanho dos blocos utilizados nesse IO, a quantidade de informação que será escrita ou lida, o tipo de operação (escrita e leitura normais ou assíncronas, mapeamento de memória de um ficheiro, etc), número de ficheiros e número de *threads* que vão processar este IO.

Como *output*, será possível obter para cada *workload* (e por cada tipo de disco virtual) o tamanho da informação processada, a média da largura de banda utilizada, o número

médio de IOPS, o tempo de latência para submeter o pedido de IO, o tempo de latência para completar o pedido de IO, a percentagem de largura de banda utilizada e a utilização do CPU.

Afinando estes parâmetros, será possível obter um guia que permita ao administrador perceber se a infraestrutura iCBD tem boa performance e onde pode melhorar a estrutura de *hardware*.

Capítulo 7

Conclusões e trabalho futuro

Em conclusão, os objetivos traçados foram cumpridos (a menos dos testes de desempenho): foi possível criar os *linked-clones* utilizando a capacidade nativa de *snapshoting* do Btrfs através de *scripts e/ou linha de comandos*, de forma transparente para o hipervisor. No entanto, devido à forma como o Btrfs usa a cache do Linux, a utilização naïve de *linked-clones* assim criados teria um desempenho bastante mau em termos de utilização da RAM do servidor de armazenamento da VDI, uma vez que cada instância (VM) criada nada ou quase nada partilharia em cache com outras instâncias (o que é natural) mas, também nada partilharia em cache com o *template* a partir do qual foi criada. Ainda assim, utilizando uma outra funcionalidade do Btrfs, o (*seeding*), é possível criar múltiplos *linked-clones* de um mesmo *template*, pensando-se que, desta forma, a ocupação da cache não irá sofrer um aumento significativo.

Para se saber se a infraestrutura poderá comportar em ambiente real um número significativo de VMs numa organização, será necessário executar diversos testes de performance nomeadamente ao sistema de armazenamento, que pode representar o *bottleneck* do iCBD. Vários tipos de afinações sobre alguns parâmetros (tipo de disco virtual, dimensão dos blocos em disco, dimensão das mensagens que passam na rede) poderão afetar em larga medida a performance da infraestrutura, sendo que apenas após vários testes se poderão ter dados suficientes para definir estratégias para obter melhores desempenhos.

De momento, pode-se concluir que o objetivo principal da dissertação foi cumprido à exceção da integração com o hipervisor KVM (ou com o *libvirt*), para o qual não houve tempo disponível para analisar o código e conseguir integrar a funcionalidade de *snapshoting* com sucesso; contudo, a integração por via de *scripts* é, do ponto de vista da operacionalidade, completamente adequada, dispensando alterações (no código do hipervisor) que comprometem sempre a sua portabilidade e capacidade de o manter atualizado em linha com os *updates* fornecidos pela distribuição Linux em uso. E obtém-se ainda uma

vantagem adicional: a solução conseguida é aplicável a hipervisores cujo código não é aberto estando já, por exemplo, a ser usada com o VMware Player.

7.1 Implementação e testes de funcionalidade

7.1.1 *Rationale* e esboço de testes de desempenho

O objetivo do projeto é o desenvolvimento de uma infraestrutura computacional que suporta a execução, de uma forma não intrusiva, de *desktops* virtuais nos PCs que normalmente se podem encontrar numa organização. A característica fundamental deste projeto reside precisamente no aspeto não intrusivo, que o distingue de outras soluções anteriormente tentadas (por exemplo, pela Citrix). Aqui o disco (e o *software* neste instalado) são totalmente preservados, não havendo lugar à instalação de qualquer *software* adicional, sendo o processo de execução desencadeado por arranque e/ou acesso remoto (*remote boot*) de todo o software e quaisquer outros dados necessários.

O foco deste trabalho, no panorama geral do projeto, está no armazenamento (*storage*) das VMs: das suas imagens, dos seus *snapshots*, e dos ficheiros e estruturas de dados que suportam as suas instâncias quando em execução. Em particular, esta tese debruça-se sobre a utilização de sistemas de ficheiros com suporte nativo para *snapshots* que alberguem, de forma eficiente, persistente ou temporária, os *itens* anteriormente referidos: imagens, *snapshots*, ficheiros de suporte à execução.

7.1.2 ANEXO

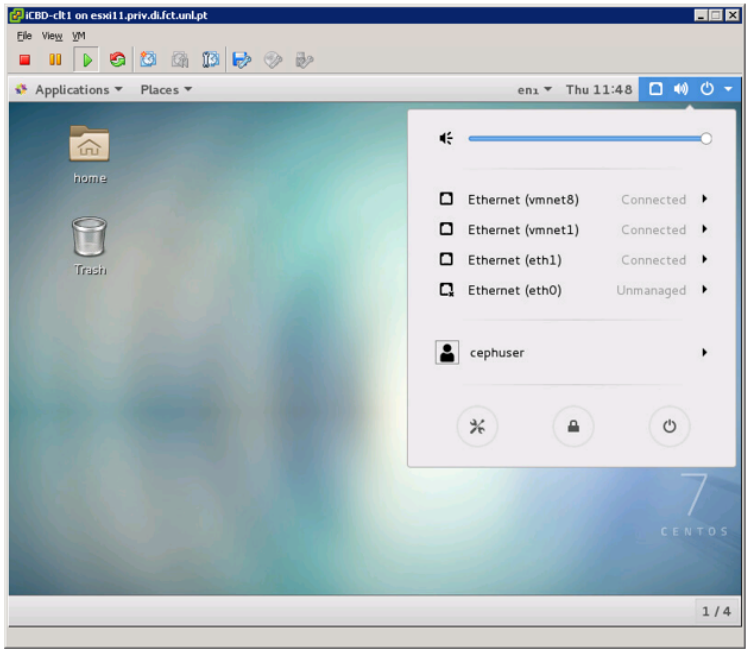


Figura 7.1: Máquina Virtual pronta a utilizar.

A primeira fase, que compreende as partes a) e b), é realizada de forma totalmente idêntica a outras utilizações de "*remote boot*" para arranque de sistemas.

Contudo, na segunda colocam-se aos administradores de sistema duas opções: a mais tradicional, utilizada desde a década de 90, recorre a um servidor NFS que exporta uma diretoria, abaixo da qual reside toda a árvore ficheiros da versão/distribuição que se está a utilizar, árvore essa que é montada no cliente.

A outra opção recorre ao acesso a volumes (*block devices*) que contêm sistemas de ficheiros cujo conteúdo é a árvore ficheiros da versão/distribuição que se está a utilizar, sendo que os volumes são montados pelo cliente da mesma forma como se monta um disco físico tradicional. Há portanto, na segunda fase uma escolha entre utilizar acessos a um sistema de ficheiros exportado por NFS, ou utilizar *block devices*, que podem ser acedidos pelos clientes através do uso de protocolos apropriados – sendo o mais conhecido destes o iSCSI, que é geralmente utilizado sobre redes locais *Ethernet* e TCP/IP.

Bibliografia

[1] *BTRFS Documentation*. (acedido Fevereiro 8, 2016). URL: https://btrfs.wiki.kernel.org/index.php/Main_Page.

[2] Callaghan, B. "NFS Illustrated. Professional Computing Series". Em: (2000).

[3] Chrobak, P. ""Implementation of Virtual Desktop Infrastructure in Academic laboratories"". Em: *Computer Science and Information Systems* (2014).

[4] Danti, G. *ZFS, BTRFS, XFS, EXT4 and LVM with KVM - a storage performance comparison*. (acedido Fevereiro 3, 2016). URL: http://www.ilsistemista.net/index.php/virtualization/47-zfs-btrfs-xfs-ext4-and-lvm-with-kvm-a-storage-performance-comparison.html.

[5] documentation, B. *Talk:Seed-device*. (acedido 12 Setembro, 2016). URL: https://btrfs.wiki.kernel.org/index.php/Talk:Seed-device.

[6] Documentation, C. *4.5. Creating Snapshot Volumes*. (acedido Fevereiro 9, 2016). URL: https://www.centos.org/docs/5/html/Cluster_Logical_Volume_Manager/snapshot_command.html.

[7] FreeBSD. *Chapter 19. The Z File System (ZFS)*. (acedido Fevereiro 8, 2016). URL: https://www.freebsd.org/doc/handbook/zfs.html.

[8] Heger, D. A. ""Workload Dependent Performance Evaluation of the Btrfs and ZFS Filesystems"". Em: *CMG International Conference* (2009).

[9] Jeff Bonwick, B. M. *ZFS The Last Word In File Systems*. (acedido Fevereiro 6, 2016). URL: http://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf.

[10] Joshi, G. "Empirical Study of Virtual Disks Performance with KVM on DAS". Em: *IEEE* (2014).

[11] kashyapc. *Snapshotting with libvirt for qcow2 images*. (acedido Agosto 22, 2016). URL: https://kashyapc.com/2011/10/04/snapshotting-with-libvirt-for-qcow2-images.

[12] Kerner, S. M. *A Better File System for Linux?* (acedido Fevereiro 5, 2016). URL: http://www.internetnews.com/dev-news/article.php/3781676/A+Better+File+System+for+Linux.htm.

[13] Kára, J. ""Ext4, btrfs, and others"". Em: *SUSE Labs, Novell* (2009).

[14] Leon, Y. e Piscopo, T. *Object Storage versus Block Storage: Understanding the Technology Differences*. (acedido Fevereiro 5, 2016). URL: http://searchstorage.techtarget.com/definition/data-deduplication.

[15] Lopes, P. “"Proposta de Candidatura ao programa P2020"”. Em: (2015).

[16] Lopes, P. A. e Medeiros, P. D. “pCFS vs. PVFS: comparing a highly-available symmetrical parallel cluster file system with an asymmetrical parallel file system”. Em: (2010), pp. 131–142.

[17] Mathur, A., Cao, M., Bhattacharya, S., Dilger, A., Tomas, A. e Vivier, L. “The new ext4 filesystem: current status and future plans”. Em: 2 (2007), pp. 21–33.

[18] Myers, B. “A Brief History of Human Computer Interaction Technology”. Em: *ACM Interactions* 5 (1998), pp. 44–54.

[19] NEC. “"Comparing the TCO of Physical PCs, VDI, and NEC’s Cloud-hosted Desktops as a Service (DaaS)"”. Em: *White paper* (2012).

[20] Nexenta. “"ZFS File System Features Technical Overview"”. Em: *White paper* (2014).

[21] Popek, J., G., Goldberg, e e P., R. “"Formal requirements for virtualizable third generation architectures"”. Em: *Communications of the ACM, 17(7):412–421* (1974).

[22] RedHat. *Snapshots: Where they are stored and how to use them?* (acedido Agosto 21, 2016). URL: https://www.redhat.com/archives/libvirt-users/2013-October/msg00018.html.

[23] Shalabh Agarwal Rana Biswas, A. N. “"Virtual Desktop Infrastructure in Higher Education Institution : Energy Efficiency as an application of Green Computing"”. Em: *Fourth International Conference on Communication Systems and Network Technologies* (2014).

[24] Siebert, E. “"Top 7 VMware Management Challenges"”. Em: *Veeam White paper* (2012).

[25] Srivastava, A. “"vDaaS"”. Em: *IEEE* (2011).

[26] Sun Microsystems, I. “"ZFS On-Disk Specification"”. Em: *White paper* (2006).

[27] Sweeney, A., Doucette, D., Hu, W., Anderson, C., Nishimoto, M. e Peck, G. “Scalability in the XFS File System.” Em: 15 (1996).

[28] tegile. “"Overcoming five unique storage challenges that fave VDI implementations"”. Em: *White paper* (2013).

[29] Vaghani, S. B. “Virtual machine file system”. Em: *ACM SIGOPS Operating Systems Review* 44.4 (2010), pp. 57–70.

[30] VMware. “"VDI: A New Desktop Strategy"”. Em: *White Paper* (2006).

[31] VMWare. *Getting The Hang Of IOPS v1.3*. (acedido Fevereiro 1, 2016). URL: http://www.symantec.com/connect/articles/getting-hang-iops-v13.

[32] VMWare. *vSphere 4 - ESX and vCenter - Working with Templates and Clones*. (acedido Fevereiro 1, 2016). URL: https://pubs.vmware.com/vsphere-4-esx-vcenter/index.jsp?topic=/com.vmware.vsphere.vmadmin.doc_41/vsp_vm_guide/deploy_vms_from_templates_and_clones/c_working_with_templates_and_clones.html.

[33] VMWare. *Virtual Machine File System (VMFS)*. (acedido Fevereiro 2, 2016). URL: https://www.vmware.com/products/vsphere/features/vmfs.

[34] Wikipédia. *Comparison of file systems*. (acedido Fevereiro 10, 2016). URL: https://en.wikipedia.org/wiki/Comparison_of_file_systems.