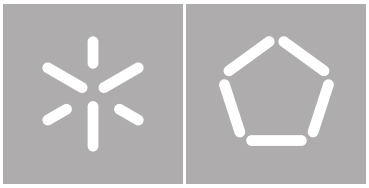
Universidade do Minho
Escola de Engenharia

Marco André Costa Dinis

FlexDeploy

Outubro de 2014



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Marco André Costa Dinis

FlexDeploy

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de

Professor António Luís Sousa

Outubro de 2014

Anexo 3

DECLARAÇÃO

Nome
Marco André Costa Dinis

Endereço electrónico: marcoandredinis@gmail.com Telefone: 912365692 / _____

Número do Bilhete de Identidade: 13981501 2 0ZZ7

Título dissertação □/tese □
Flex Deploy

Orientador(es):
António Luís Sousa

Ano de conclusão: 2014

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:
Mestrado em Engenharia Informática

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;
2. É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE/TRABALHO (indicar, caso tal seja necessário, nº máximo de páginas, ilustrações, gráficos, etc.), APENAS PARA EFEITOS DE INVESTIGAÇÃO, , MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;
3. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Universidade do Minho, 31 / 10 / 2014

Assinatura: Marco André Costa Dinis

Agradecimentos

Ao Nuno pelo conhecimento partilhado, aos Professores António Luís Sousa e José Orlando Pereira pelo acompanhamento ao longo desta dissertação.

À Eurotux por me acolher no decorrer da realização deste projeto que tanto me ensinou.

A todos os que me acompanharam de perto neste percurso académico.

Por último, mas mais importante, à minha família. Ao meu pai, mãe, irmão e avô.

Resumo

FlexDeploy

Atualmente, as TI estão presentes e representam um papel importante em qualquer organização. O aumento do número de computadores numa organização, quer sejam *desktops*, portáteis ou servidores, origina inevitavelmente um aumento nos custos operacionais da organização.

A gestão de *desktops* torna-se então num ponto fulcral para a organização. Uma das possibilidades para otimizar esse processo passa pela criação de um sistema de *deploy* em que os utilizadores têm acesso à sua área de trabalho independentemente da máquina que usam no momento.

Tendo em conta a emergência dos serviços na *cloud*, mas não desvalorizando o poder de computação das máquinas locais, o objetivo desta dissertação de Mestrado consiste em perceber de que forma é possível criar um sistema de virtualização que possa correr a partir da *cloud* ou a partir da máquina local tirando partido dos recursos disponíveis.

Abstract

FlexDeploy

Currently, IT are involved and play an important role in any organization. The increase in the number of computers in an organization, whether desktops, laptops or servers, inevitably leads to an increase in operating costs of the organization.

Desktop management, then, becomes a key point for the organization. One of the possibilities to optimize this process involves the creation of a system for deployment in which users have access to theirs desktop regardless of the machine they use at the moment.

Given the emergence of cloud services, but not devaluing the computing power of local machines, the goal of this Master's thesis is to realize that it is possible to create a virtualization system that can run from the cloud or from the local machine by taking advantage of available resources.

Conteúdo

Nomenclatura

DRBD Distributed Replicated Block Device

JSON JavaScript Object Notation

KVM Kernel-based Virtual Machine

QEMU Quick EMUlator

SPICE Simple Protocol for Independent Computing Environments

VDI Virtual Desktop Infrastructure

Lista de Figuras

Capítulo 1

Introdução

Na maior parte das organizações a heterogeneidade dos equipamentos é uma realidade incontornável, mesmo quando se tenta mantê-los o mais semelhantes possível, pois basta que equipamentos aparentemente iguais sejam comprados em momentos diferentes para que tenham algumas componentes e versões de software ligeiramente diferentes. Isto irá inevitavelmente originar planos de manutenção diferentes e logo um aumento do custo operacional, porque estas situações exigem muitas vezes o crescimento da equipa que garante a manutenção e operacionalidade dos parques computacionais.

Com este projeto, pretende-se desenvolver um produto de aprovisionamento e instalação automática para parques informáticos. Para solucionar este desafio, anteveem-se várias possibilidades, todas elas aproveitando de alguma forma as mais recentes técnicas de virtualização. Virtualização permite tornar os diferentes sistemas mais homogéneos, uma vez que cada utilizador terá uma versão ligeiramente alterada em relação a um *desktop* base. Sempre que possível será utilizado um *hypervisor*, gestor de virtualização, que pode ser utilizado durante o processo de instalação e configuração e que fica ativo depois desta fase a gerir a virtualização e os recursos computacionais disponíveis. Esta solução é adequada quando os recursos da máquina são superiores aos necessários para efetuar as tarefas do utilizador e permite aproveitar a capacidade excedente para outras tarefas, por exemplo serem utilizados em tarefas de cálculo ou processamento de dados. Permite também que fora das horas normais de trabalho sejam disponibilizados

recursos computacionais para as tarefas referidas.

No caso em que os utilizadores fazem uso intensivo e/ou físico de alguns componentes de hardware, por exemplo placa gráfica ou leitor de impressões digitais, é necessário alterar a abordagem. Assim sendo, o hypervisor fica apenas responsável por realizar a instalação e manutenção, mas será desligado durante o processo normal de operação de modo a que o sistema operativo tenha total controlo dos recursos.

1.1 Motivação

Muitas organizações estão a ponderar a utilização de VDI [10] como forma de contornar os custos de manutenção de operacionalidade dos *desktops*. Ao ter na secretária do utilizador, quase exclusivamente, um ecrã e um teclado que se liga a um servidor que fornece o ambiente de *desktop*, reduz-se o custo de manutenção de operacionalidade. Atualmente o custo inicial destas soluções é bastante elevado [1], sendo o investimento comparável ao dos *desktops* mas com funcionalidade reduzida. Nestes casos, a solução proposta neste projeto pretende prolongar a vida útil dos *desktops* existentes ao desenvolver um cliente de VDI gerido pelo hypervisor da máquina e ainda poder utilizar os recursos existentes para outras tarefas. No caso de ambientes fabris que provocam desgaste muito rápido dos componentes móveis dos equipamentos, a utilização deste tipo de solução será o ideal por permitir reduzir ao mínimo a intervenção da equipa de TI. Se, adicionalmente, os *desktops* tradicionais forem progressivamente substituídos por *thin* clientes, esta será a solução ideal para este cenário de aplicação.

Além das diferenças nas tarefas que cada utilizador tem de efetuar, a especificidade dos colaboradores das organizações faz com que nem todos tenham as mesmas necessidades em termos das aplicações utilizadas. Neste cenário, faz todo o sentido que o aprovisionamento e instalação de cada computador tenha em consideração a especificidade do seu utilizador final, quer seja a nível de grupo de utilizadores quer seja do utilizador individual.

Em organizações com alguma dimensão é natural que os recursos computacionais estejam distribuídos por várias localizações. A mobilidade dos colaboradores e dos seus computadores portáteis é também um dos fatores responsáveis por

esta distribuição. Tendo em conta a mobilidade dos utilizadores e a existência de múltiplas localizações da organização, a otimização do processo de instalação aconselha a que os recursos necessários se encontrem o mais perto possível dos utilizadores. Este requisito faz com que os recursos sejam replicados pelas várias instalações da organização. Uma vez que estes recursos tendem a crescer com o tempo, é aconselhável a utilização das mais recentes técnicas de deduplicação para acelerar o processo de replicação dos dados poupando significativamente na largura de banda necessária para a realização desta tarefa.

A instalação simultânea de vários computadores semelhantes coloca uma sobrecarga adicional na rede e servidor de aprovisionamento que vai ter de enviar várias vezes os mesmos dados, uma para cada computador a instalar. Para otimizar esta situação deve ponderar-se a possibilidade de utilização de protocolos de *multicast* para a realização destas tarefas.

A evolução a que se assiste na capacidade de comunicação, com linhas de elevado débito a preços acessíveis, leva alguns decisores a ponderarem a utilização de serviços de *cloud* para suportarem todas as infraestruturas, quer servidor quer *desktop* com soluções de VDI. A pensar neste tipo de situações, está previsto o projeto desenvolver uma solução de *cloud* e que permita fazer a gestão e manutenção também neste cenário. Em alternativa pode considerar-se a *cloud* como o repositório central a partir do qual são replicados os dados para as diversas instalações da organização.

Embora existam diversas soluções de VDI no mercado, quer o custo dos terminais para os utilizadores, quer a infraestrutura de *Datacenter*, tornam a sua adoção proibitiva. Com este projeto pretende-se desenvolver uma solução mais acessível ao mercado das PME, dotando as empresas deste segmento de capacidades (como ambientes empresariais totalmente *stateless*) que normalmente existem nos segmentos mais elevados.

1.2 Objetivos

Para resolver os problemas descritos vai-se usar diferentes abordagens: virtualização remota, virtualização local e utilização nativa.

Virtualização remota Usada no caso em que o utilizador dispõe apenas de um computador com fracas capacidades computacionais, não possuí ou não pode usar o disco, ou a transferência de uma grande quantidade de dados não é aceitável. Esta abordagem permite uma utilização imediata do ambiente de trabalho através de um protocolo de partilha de *desktop*. No entanto, não é possível usar os recursos locais da máquina e o acesso fica restrito ao correto funcionamento da rede.

Virtualização local Implica transferir todo o sistema para a máquina local. Para que tal seja possível, é necessário sincronizar os discos entre o servidor e a máquina local, que levará mais ou menos tempo em função do tamanho do disco. Com este método, o uso do ambiente de trabalho fica tolerante à falha da ligação ao servidor, porque a computação é realizada do lado do cliente. Nesta situação o servidor mantêm-se atualizado face às alterações que o utilizador efetua na máquina local, à exceção da falha da rede. Devido ao tempo necessário para a sincronizar, vai ser usado o primeiro método descrito, para que o utilizador tire partido imediato do seu ambiente de trabalho, até que a sincronização esteja completa.

Utilização nativa Utilizador pode transferir o seu sistema para a máquina local, e após reiniciar o ambiente, poderá usufruir por completo das características da máquina local. Posteriormente, será possível fazer o envio do estado da máquina para o servidor para que as alterações não sejam perdidas.

Como tentativa de otimização do armazenamento e transferência de dados vai ser usada deduplicação. Deduplicação consiste na deteção e remoção de dados repetidos.

Como se pode observar na Figura 1.1 [23], quando existem vários ficheiros com blocos em comum, apenas os blocos únicos são guardados em disco. Esta técnica permite reduzir o espaço real ocupado pelo armazenamento, a taxa de redução varia dependendo do tipo de ficheiros presentes e das técnicas usadas, sendo possível atingir valores na ordem dos 80% [9] no caso de imagens de máquinas virtuais. Este tema será abordado com mais detalhe no Capítulo 2.

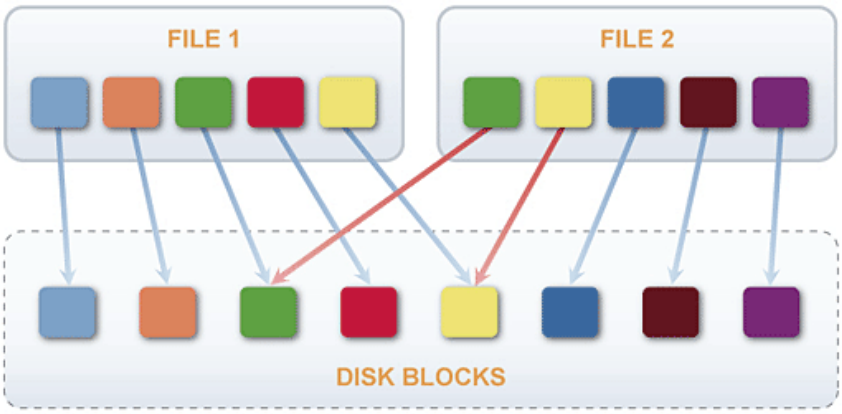


Figura 1.1: Deduplicação

1.3 Estrutura do Documento

No Capítulo 2 será descrita a virtualização, replicação de dados e replicação. Para cada um dos temas, serão abordadas algumas ferramentas fazendo uma comparação entre elas e por fim indicando a que foi escolhida para a realização deste projeto.

No Capítulo 3 é abordada a questão de acesso remoto, a migração e virtualização local.

No seguinte Capítulo 4 será explicado como é possível utilizar o sistema nativamente, e o envio do novo estado para o servidor.

Por fim, no Capítulo 5 será feita a conclusão da dissertação, assim como possível trabalho futuro a realizar neste tema.

Capítulo 2

Estado de Arte

Este capítulo tem como principal objetivo introduzir as tecnologias e conhecimento existentes nesta área. Inicialmente será introduzida a virtualização, quais os tipos de virtualização existentes assim como *hypervisors*.

Em seguida, será abordada a replicação de discos em sistemas distintos, para que seja possível migrar máquinas virtuais entre dois *hypervisors*.

Por último, será abordado o tema da deduplicação e qual o seu papel em conjunto com as outras tecnologias.

2.1 Virtualização

Virtualização consiste na criação de um conjunto de componentes virtuais e isolados capazes de correr um sistema operativo convidado, fazendo uso desses componentes.

Existem níveis de privilégios para que o sistema operativo controle as interações entre as aplicações e o hardware. Esse níveis de privilégio, chamam-se *rings* ou anéis, e estão organizados numa forma hierárquica que começa no anel 0, e que corresponde ao *kernel* do sistema operativo e como tal corresponde ao nível mais privilegiado. Os dois níveis seguintes correspondem aos componentes mais críticos do sistema operativo, como o caso de *drivers*. Sendo o último nível, aquele que engloba as aplicações que o utilizador interage, como o caso de navegadores web e processadores de texto.

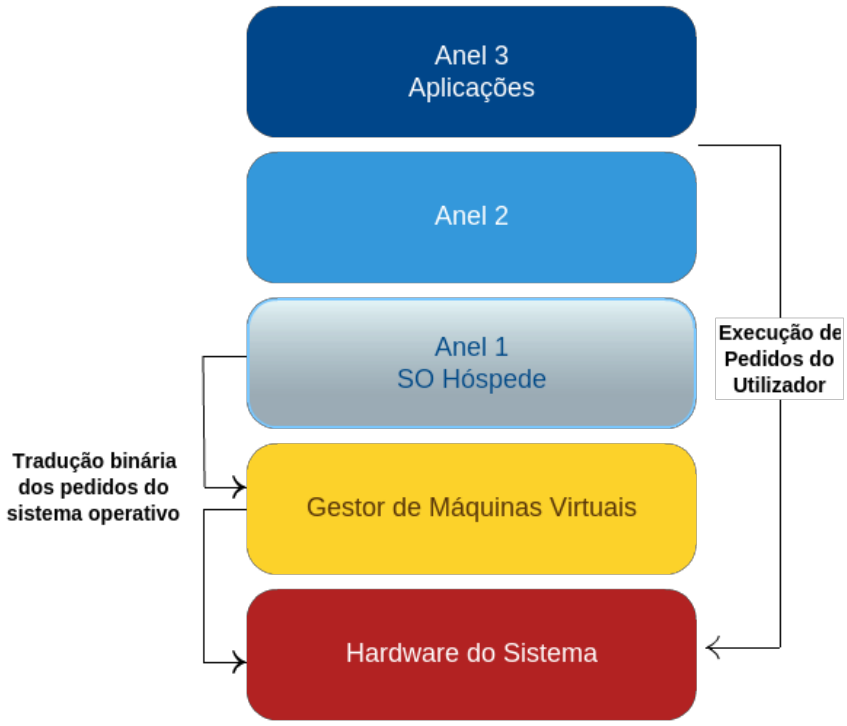


Figura 2.1: Virtualização completa

Existem três tipos de virtualização [25]: virtualização completa, para-virtualização e virtualização ao nível do sistema operativo. Cada tipo de virtualização funciona num conjunto de anéis diferentes.

Virtualização completa Permite ao sistema operativo convidado aceder diretamente ao processador e disco. Cada convidado é independente entre si, não tendo conhecimento da existência de outros convidados. O anfitrião é responsável por gerir todo o sistema, incluindo alocar recursos para cada um dos sistemas operativos convidados. Sistema de virtualização opera no anel 0, já o sistema operativo convidado opera no anel 1. Este sistema é conhecido como *Hypervisor*. Ver Figura 2.1.

Para-virtualização Relativamente semelhante com a virtualização completa, mas neste caso os sistemas operativos convidados necessitam de modificações ao

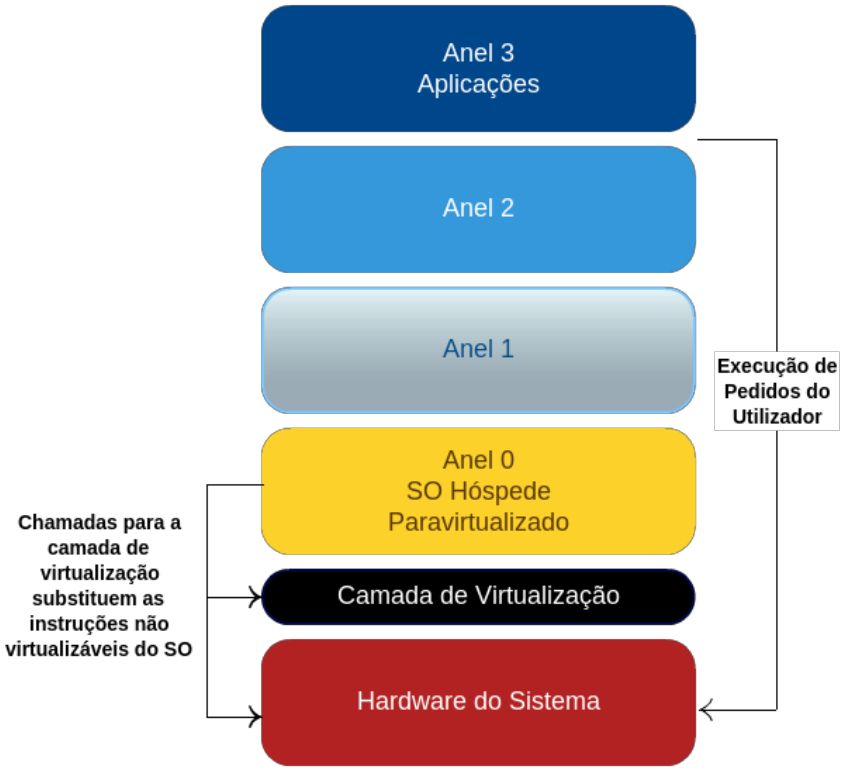


Figura 2.2: Para-virtualização

nível do sistema operativo, para que as operações não virtualizáveis possam ser chamadas a partir da interface do *hypervisor*. O anfitrião tem uma menor carga computacional, uma vez que cada convidado tem acesso total ao processador. Como são necessárias modificações ao núcleo do sistema operativo, esta técnica sofre de portabilidade e impossibilita que seja o utilizador a escolher livremente qual o *desktop* a usar. O sistema operativo convidado opera no anel 0. Ver Figura 2.2.

Virtualização assistida por hardware Neste contexto não existe conceito de *Hypervisor*. Existe um *kernel* comum e partilhado por todas as instâncias em vez de cada instância ser independente. Este método também é conhecido por *containers virtualization*. Cada *container* equivale a uma instância do sistema operativo base, não sendo assim possível ter um sistema operativa da família *Windows* tendo como base um *kernel* Linux. Esta limitação proíbe o uso deste

modelo para o objetivo proposto, uma vez que é pretendido usar *desktops* de famílias distintas, e independentes do sistema operativo base. Sistema operativo convidado situa-se no anel 0. Ver Figura 2.3.

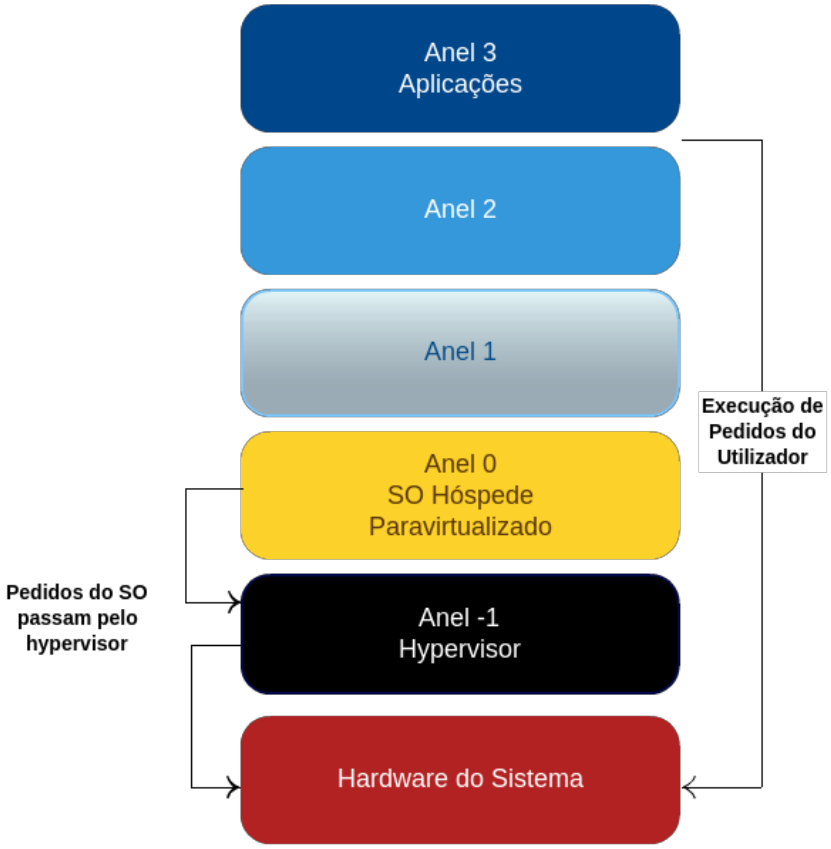


Figura 2.3: Virtualização assistida por hardware

2.2 Replicação de disco

Um dos objetivos definidos é a possibilidade de migração entre um servidor e o posto do cliente, para que tal seja possível é necessário que o disco tenha o mesmo conteúdo em ambos *hypervisors* (nodos).

Existem dois principais modos de replicação: síncrona e assíncrona.

No caso de replicação síncrona, um dado bloco só é dado como escrito quando este é escrito nos dois nodos. Um modelo totalmente síncrono poderá aumentar

demasiado a latência na escrita dos dados por parte do sistema operativo convidado. No entanto, permite a migração do sistema operativo convidado porque os discos têm cópias exatamente iguais. Ver Figura 2.4.

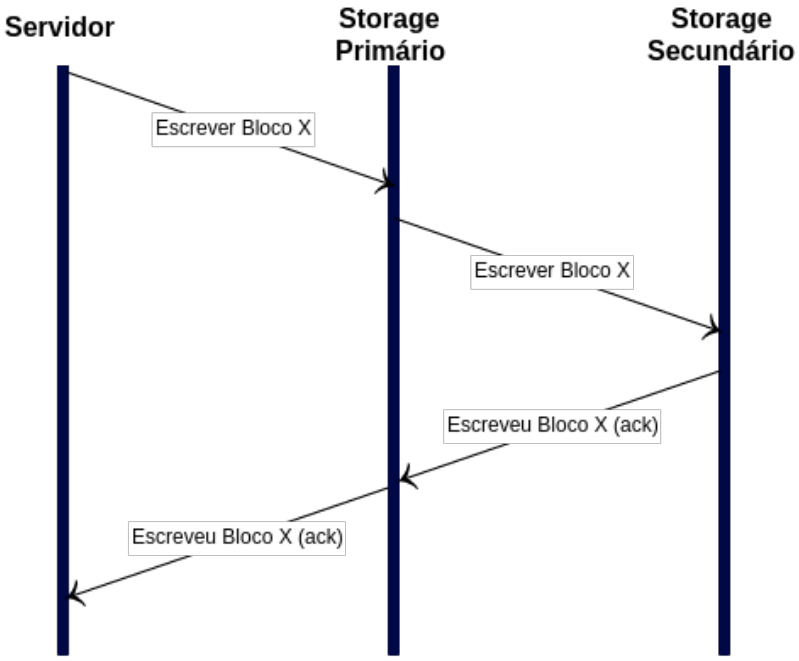


Figura 2.4: Replicação Síncrona

Replicação assíncrona permite uma menor latência uma vez que dado bloco é dado como escrito após a escrita no nodo principal. Não há garantias que o sistema num dado momento esteja sincronizado, apenas que, inevitavelmente, os discos irão convergir a menos de falha de rede ou de algum dos nodos. Ver Figura 2.5.

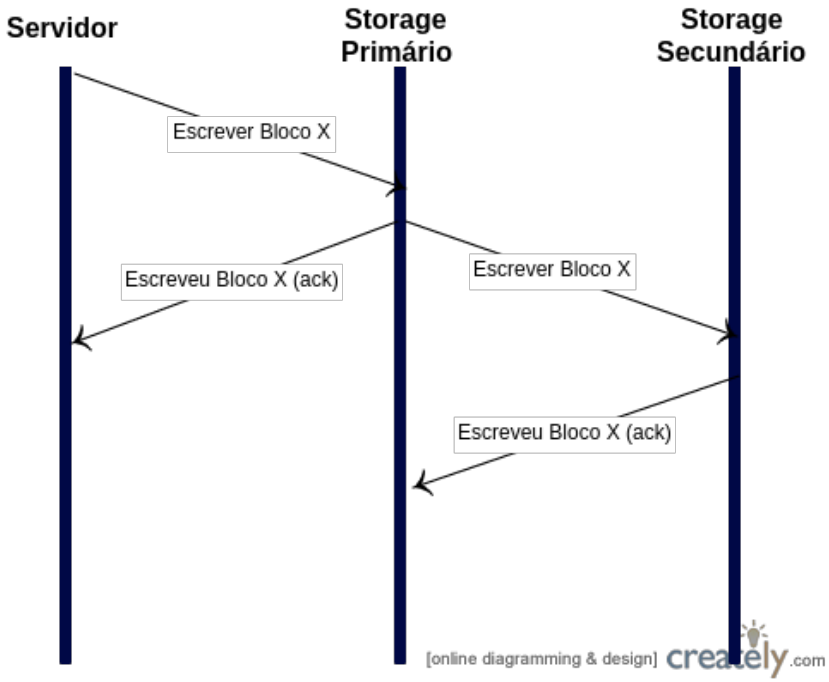


Figura 2.5: Replicação Assíncrona

2.2.1 Storage

A utilização de uma *storage* permite a migração de uma máquina virtual entre dois *hypervisors*. No entanto, a arquitetura destes sistemas obriga o utilizador a ter acesso constante à *storage*. Isto torna-se uma limitação uma vez que o utilizador a dado momento deverá ser capaz de usar o seu posto local e não ficar dependente de outras estruturas.

2.2.2 DRBD

DRBD[13] é um replicador distribuído de blocos do disco, ou seja, o seu funcionamento não é o mesmo que outros sistemas de sincronização que funcionam em função dos ficheiros. Com o DRBD é possível ter cópias exatas [14] de discos entre diferentes postos (ou nodos), obtendo assim uma homogeneidade necessária para o correto funcionamento da migração.

Existem três diferentes tipos de funcionamento [7].

Protocolo A - Assíncrono Ação completa quando as escritas locais acontecem e as remotas são colocadas no *buffer* de rede. É o modo que permite maior velocidade na replicação dos dados, mas no caso de falha da rede, os nodos ficam dessincronizados. Apenas pode existir um nodo principal.

Protocolo B - Semi-síncrono Ação completa quando as escritas locais acontecem e as remotas chegam ao computador destino. Este modo é mais lento relativamente ao Protocolo A pois requer que os dados sejam recebidos no nodo secundário. Não permite a existência de dois nodos principais (*master*).

Protocolo C - Síncrono Ação completa quando as escritas locais acontecem, assim como as remotas. Este é o modo mais lento porque exige que os dados sejam escritos no nodo secundário, aumentando a latência. Permite que hajam dois nodos principais, possibilitando que duas aplicações distintas possam interagir com o mesmo disco estando cada uma em postos distintos.

O DRBD é a ferramenta adotada uma vez que permite sincronização dos discos com relativo baixo custo. Um outro ponto a favor é permitir alguma flexibilidade no que diz respeito aos modos de sincronização, sendo possível adaptar o sistema para ambientes em que as condições da rede são menos favoráveis.

2.3 Deduplicação

Ao usar um sistema base para vários utilizadores replica-se muita da informação presente no sistema operativo.

Deduplicação é uma técnica de compressão de dados que elimina cópias de repetição de dados. Ver Figura 2.6.

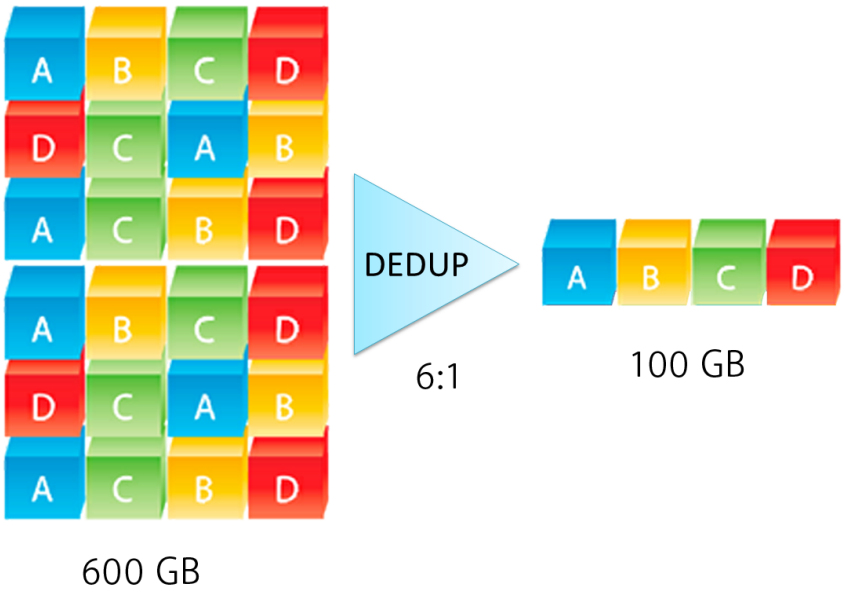


Figura 2.6: Deduplicação

Usando deduplicação em imagens de máquinas virtuais é possível obter uma redução de 80% [2] [27].

Deduplicação de dados pode funcionar ao nível do ficheiro ou bloco. Cada ficheiro ou bloco recebe uma assinatura (ver Figura 2.7) que corresponde ao cálculo de um algoritmo de *hash* sobre os bytes presentes.

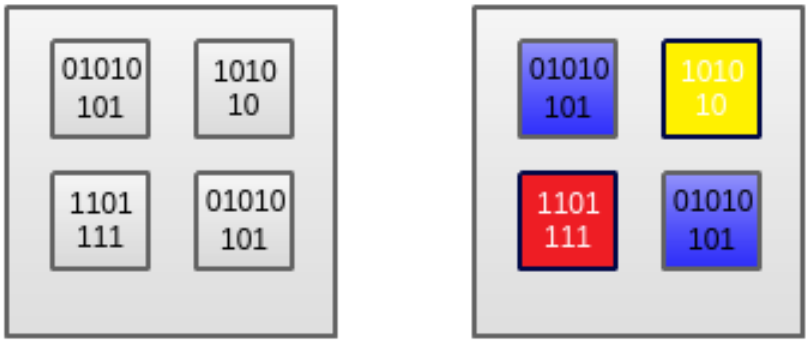


Figura 2.7: Deduplicação - Exemplo - Análise

Cada assinatura é guardada num índice, que poderá estar em RAM para rápido acesso. Sempre que é encontrada uma assinatura igual será necessário apenas indicar que o seu conteúdo é uma cópia de dados já existentes, e indicar a localização desses dados. Ver Figura 2.8.

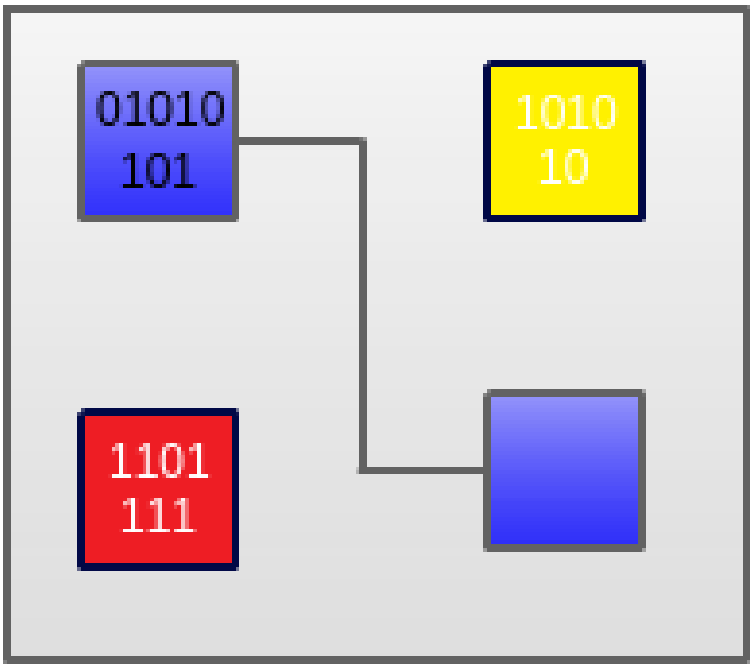


Figura 2.8: Deduplicação - Exemplo - Pós-deduplicação

Podem-se considerar os seguintes cinco pontos como chave na implementação da deduplicação: granularidade, localidade, tempo, indexação e técnica [15] [19]. Cada um destes pontos de decisão vai assumir valores diferentes mediante o contexto onde se pretende implementar deduplicação.

2.3.1 Granularidade

Granularidade [20] define o tamanho do segmento de dados considerado como unidade na eliminação de duplicados. Tamanho elevado leva a um conjunto de possíveis duplicações mais reduzido possibilitando que todos estes índices estejam

guardados na RAM, no entanto cada indexação e posterior comparação será mais lenta por unidade. Por outro lado, tamanho reduzido possibilita um processamento mais rápido por unidade mas provoca um maior número de possíveis duplicações, sendo que nem todos os índices poderão estar presentes em RAM.

2.3.2 Localização

Existem dois tipos de localização: temporal e espacial.

Localização temporal significa que duplicações vão aparecer várias vezes num pequeno período de tempo. Partindo deste pressuposto, uma das possíveis otimizações passa por criar uma lista com as duplicações mais usadas, e colocar essa lista numa zona de rápido acesso.

Localização espacial significa que duplicações vão aparecer pela mesma ordem em diferentes fluxos de dados. Por exemplo, se um segmento A é seguido do segmento B e C, quando A aparecer novamente é provável que B e C o sigam. Sempre que A apareça, é carregado para a RAM as assinaturas de B e C para posterior verificação de duplicados.

2.3.3 Tempo

Outro fator a considerar é quando devem ser as duplicações detetadas e removidas. Existem duas soluções: *in-line*, e *off-line*.

Deduplicação *in-line* consiste em detetar duplicações antes de estas serem escritas. Este método requer a interceção dos pedidos de escrita ao disco, calculando a assinatura e deteção de duplicação. Um dos principais pontos contra este método é o tempo de processamento necessário para que seja processada uma escrita.

Deduplicação *off-line* passa por completar os pedidos de escrita assim que possível, posteriormente todos os novos segmentos são examinados e é efetuado o processo de deduplicação.

2.3.4 Indexação

Cada segmento pode ser sumariado usando dois diferentes métodos: assinaturas por identidade e por similaridade. Ao usar assinaturas por identidade é possível procurar duplicações exatas, no entanto leva a um uso elevado de recursos computacionais. Assinaturas por similaridade causam um maior número de índices mas permitem um menor uso de computação.

Posterior ao cálculo da assinatura, é necessário indexar os valores. Existem três abordagens neste sentido: indexação completa, indexação dispersa e indexação parcial.

Com uma indexação completa, todas as assinaturas são indexadas causando uma entrada por cada segmento único no armazenamento, permitindo encontrar todos os candidatos a duplicação, mas o tamanho do índice torna-se um obstáculo para performance e escalabilidade.

Indexação dispersa consiste em mapear um conjunto de segmentos, que tenham alguma semelhança, para um índice único, tornando assim a lista de índices mais pequena e permitindo que seja mantida na RAM.

Com indexação parcial, apenas são mantidos no índice um sub-conjunto de todas as assinaturas dos segmentos, obtendo um reduzido valor de índices mas sacrificando a quantidade de deduplicação.

2.3.5 Técnica

Existem duas principais técnicas no que diz respeito a deduplicação: *aliasing* e *delta-encoding*[15].

Aliasing consiste no cálculo exato da assinatura do segmento de dados. Este método requer menos processamento e tem um mais rápido tempo de restauro porque os segmentos não necessitam de nenhum cálculo para serem usados.

Delta-encoding elimina dois segmentos similares, mas não totalmente iguais. Apenas um segmento é totalmente armazenado, sendo que os segmentos similares são recuperáveis usando uma diferença entre a base e o segmento pretendido, usando assim menos espaço em relação ao *aliasing*.

Sumário

Neste capítulo foram explicados os conceitos básicos e que tipos de virtualização existem. Em seguida, foi abordado o DRBD como ferramenta de replicação de disco. Por último, foi descrito o que é a deduplicação abordando diferentes aproximações mediante as configurações disponíveis na implementação.

Com o estado de arte estudado, é necessário escolher quais os componentes a interligar e adaptar para que os objetivos propostos no Capítulo 1 possam ser alcançados.

Capítulo 3

Acesso Remoto e Migração

Este capítulo descreve as ferramentas utilizadas, como é possível virtualizar um sistema e dar acesso a alguém que se ligue remotamente, assim como migrar a máquina para a estação do cliente.

Inicialmente, será descrita a interação realizada entre o posto cliente e o servidor para que o utilizador tenha acesso ao seu *desktop*. Em seguida, vão ser abordadas as ferramentas de virtualização, nomeadamente o sistema de replicação de blocos DRBD, e o sistema de virtualização QEMU usando KVM. A secção seguinte vai descrever como é possível sincronizar os discos, para que, após sincronizados, seja possível enviar o *desktop* para o posto do cliente ficando este encarregue de o virtualizar.

3.1 Problema

É pretendido que o sistema seja capaz de virtualizar o *desktop* do utilizador num servidor remoto. O utilizador deverá ter acesso assim que o mesmo seja iniciado.

Caso exista um disco no posto do cliente, a sincronização deverá iniciar. Deverá ser possível interromper a sincronização a qualquer momento, permitindo ao utilizador uma gestão personalizada dos recursos de rede. Após sincronizado, o utilizador poderá migrar o sistema virtualizado para o seu posto. A migração a partir do posto local para o servidor deverá também ser possível.

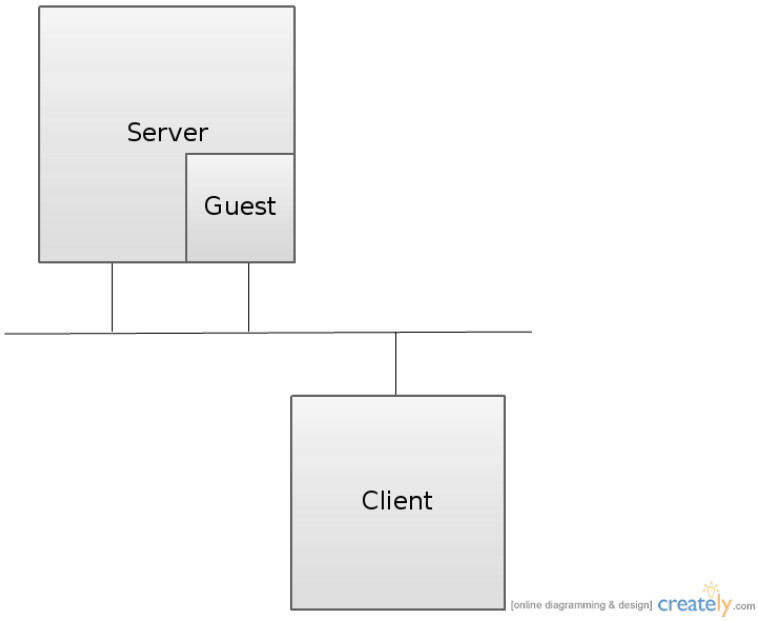


Figura 3.1: FlexDeploy - Virtualização remota

3.2 Arquitetura

O utilizador deverá ter acesso ao servidor através de uma conta previamente criada, bem como ter pelo menos uma máquina virtual associada a essa conta.

Após fazer *login* e escolher uma máquina, o utilizador têm acesso ao seu *desktop*. O sistema global é o indicado na Figura 3.1. Este modelo vai ser designado por virtualização remota, uma vez que a máquina virtual é executada no servidor e a interação é feita remotamente.

Após sincronizar os discos do servidor e do cliente, e efetuar a migração, o sistema resultante é o indicado na Figura 3.2. Ficando agora o posto cliente com os custos da virtualização do *desktop*. Este modelo vai ser designado por virtualização local.

3.3 Ferramentas

As ferramentas utilizadas na resolução destes objetivos são as seguintes:

- Python + PyQt [16] - linguagem usada para criar o sistema base assim

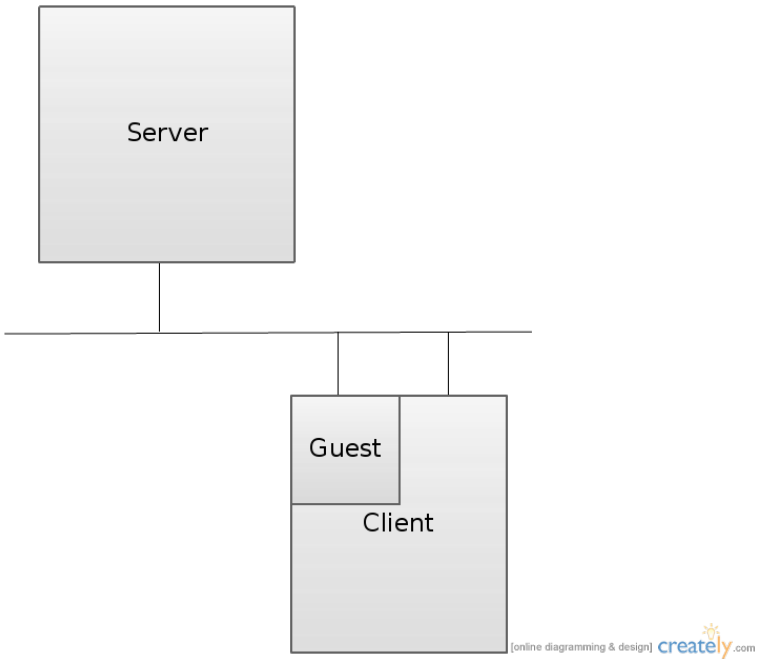


Figura 3.2: FlexDeploy - Virtualização local

como a interface com o utilizador

- DRBD [13] - sincronização dos discos
- QEMU/KVM [17] - sistema de virtualização utilizado
- SPICE [21] - permite interação com sistemas virtualizados

Para garantir uma comunicação mais fiável, esta é feita com a ajuda de objetos JSON [11]. É enviado o tamanho e só depois o objeto, isto garante que o recetor não vai tentar interpretar o objeto antes de este estar completamente do seu lado. Pode-se ver pelo excerto de código seguinte as funções que recebem e enviam objetos.

```
1 def readObj( self ):
2     size = self._msgLength( )
3     data = self._read( size ).decode()
4     frmt = "@%ds" % size
5     msg = struct.unpack( frmt, data.encode() )
6     msgDict = json.loads( msg[0].decode() )
```

```
7       self.newMsg.emit( msgDict )
8
9   def sendObj( self, obj ):
10      print('SendObj', obj)
11      msg = json.dumps( obj )
12      frmt = "@%ds" % len( msg )
13      packedMsg = struct.pack( frmt, msg.encode() )
14      packedHdr = struct.pack( '@I', len(packedMsg) )
15
16      self._send( packedHdr )
17      self._send( packedMsg )
```

3.4 Acesso Remoto

Após o utilizador escolher uma máquina (ver Figura 3.3), vai ser iniciada a virtualização no servidor e interação no posto do cliente.

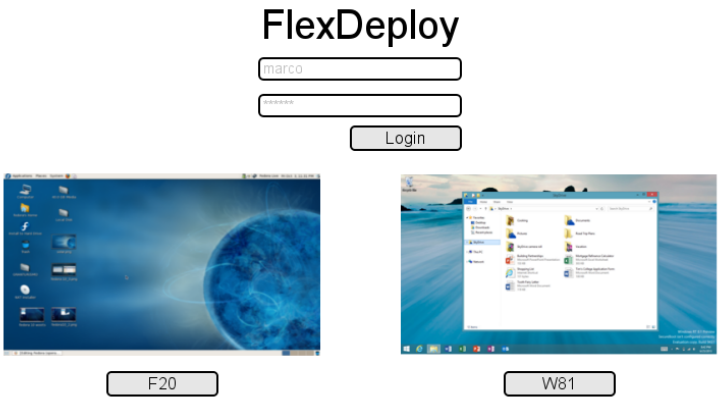


Figura 3.3: FlexDeploy - Login

3.4.1 Servidor

O papel do servidor neste processo passa por iniciar a máquina e dar os dados de acesso ao cliente.

Acesso

Na configuração do protocolo de acesso à máquina virtual existem duas opções: VNC[26] [18] e SPICE [21]. Estes protocolos permitem a interação entre o utilizador e a máquina.

Algumas das funcionalidades do SPICE são[22].:

- *streaming* de vídeo - heuristicamente identifica *streaming* de vídeo e transmite num formato mais compacto, nomeadamente, M-JPEG
- compressão de imagem - vários métodos para compressão de imagens, que posteriormente serão transmitidas ao cliente
- *live migration* - permite ao sistema migrar o sistema operativo convidado com um *downtime* muito reduzido
- driver gráfico - permite ao utilizador definir uma resolução que melhor se adequa ao monitor presente no posto do cliente
- multiplos monitores - permite o uso de vários monitores no posto do cliente
- audio - permite ao cliente reprodução de audio do sistema operativo convidado
- partilha de dispositivos USB - ainda em fase experimental, esta funcionalidade permite que o sistema operativo convidado tenha acesso a dispositivos USB presentes no posto do cliente

O seu funcionamento consiste em 3 distintos componentes. Um dispositivo apresentado ao sistema operativo convidado, um servidor presente no *hypervisor* e um cliente presente no posto do utilizador. Caso o sistema operativo convidado não possua os *drivers* relativos ao dispositivo, este será usado como um dispositivo *standard* de vídeo, VGA. O servidor fica responsável por aceitar pedidos de ligação provenientes do cliente. O cliente liga-se ao servidor, recebe a imagem do sistema operativo convidado e gere a utilização do rato e teclado.

O funcionamento é ilustrado na Figura 3.4.

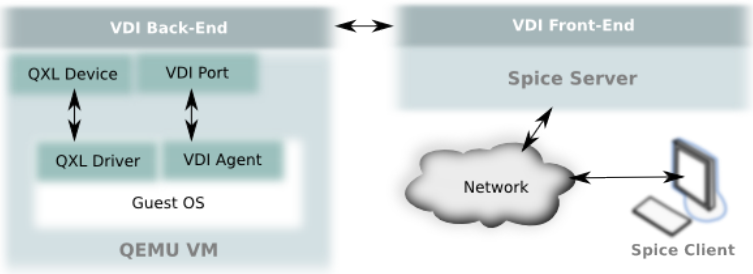


Figura 3.4: SPICE Schema

Optou-se por escolher o protocolo SPICE pois este tem algumas vantagens relativamente ao VNC[8], nomeadamente na reprodução de audio e partilha de dispositivos USB.

Replicação de Dados

O disco usado pela máquina virtual é gerido pelo DRBD. Para iniciar a sincronização é necessário adicionar um novo nó na configuração do DRBD. As configurações do DRBD são ficheiros numa localização específica. Cada ficheiro corresponde a um *resource*. Um *resource* tem várias informações relativas à configuração do disco [6] [4]. Segue um exemplo de uma configuração usada para uma máquina virtual:

```
1  resource rF20 {
2      net{
3          protocol A;
4          after-sb-1pri discard-secondary;
5      }
6      device /dev/drbd1;
7      disk{
8          size 20G;
9      }
10     on server {
11         disk      /dev/sdb1;
12         meta-disk internal;
13         address   10.10.3.23:7789;
14     }
15     on live {
16         disk      /dev/sda;
```

```
17          meta-disk internal;
18          address   10.10.3.22:7789;
19      }
20  }
```

Nesta configuração estão definidas as seguintes propriedades:

- *net*
  - *protocol A* - protocolo inicial é A (assíncrono)
  - *after-sb-1pri discard-secondary* - após um *split-brain* (processo em que existe uma divergência entre pelo menos dois nodos) o nodo que deverá ser considerado como contendo os dados mais atualizados será o que estiver definido como primário.
- *device* - dispositivo apresentado ao sistema como sendo o disco a utilizar
- *on <location>* - definições para a localização do nodo (deverá ser o *hostname* da máquina física)
  - *disk* - disco ou partição usada pelo DRBD para guardar os blocos de informações escritos em *device*
  - *meta-disk* - poderá ser *internal* e nesse caso os meta-dados são guardados em *disk*; ou então especificado um disco ou partição
  - *address* - endereço da máquina e porta a usar pelo serviço

Após o inicio da máquina, o servidor envia para o cliente a configuração do DRBD (o ficheiro deverá ser igual em ambos os postos) assim como os dados necessários para que o cliente se ligue via *SPICE*.

```
1   #...
2   data = {'action': 'drbdInit',
3           'conf': self.vm.getDRBDConf()}
4
5   self.js.sendObj(data)
```

```
def startRemoteFull(self):
    remoteFullConfig = {
        'action': 'spice',
        'IP': self.myIP,
        'Port': self.currentVMInfo['spicePort'],
        'tcpPort': self.currentVMInfo['tcpPort']
    }

    self.js.sendObj(remoteFullConfig)
```

3.4.2 Cliente

O cliente é responsável por apresentar ao utilizador as máquinas a que este tem acesso (como se pode ver na Figura 3.3). Após a escolha de uma delas comunica com o servidor de modo a receber a informação necessária para iniciar o cliente *SPICE*. O cliente *SPICE* é iniciado em modo *fullscreen* para que, do ponto de vista do utilizador, não exista nenhum outro elemento além do seu *desktop*.

```
def handleConnectSpice(self, data):
    args = "spicy -f -h " + data['IP'] + " -p " + str(data['Port'])
    self.spicePort = data['Port']
    self.hg.setTcpPort(data['tcpPort'])
    self.startProcessWithWindow(args)
```

O utilizador tem, assim, acesso à sua máquina podendo utilizá-la no imediato.

3.5 Sincronização

Para que seja possível migrar a máquina do servidor para o cliente é necessário sincronizar os discos. Existindo um disco no computador do cliente, vai ser iniciado o serviço de *DRBD* com as configurações definidas no servidor. O serviço inicia e a sincronização começa logo que possível.

O modo inicial do *DRBD* é o A (assíncrono) que permite ao servidor ter um menor tempo de resposta aquando das escritas por parte da máquina virtual.

A sincronização é um processo demorado, é percorrido todo o disco e vai sendo criado um conjunto de meta-dados. Para otimizar o processo, sempre que o cliente se desliga os meta-dados são guardados. Quando o cliente se ligar novamente, esses dados são usados. Nem sempre será possível tirar partido, por vezes as diferenças entre os dois pontos são tais que o serviço se recusa a usar esses meta-dados e começa a sincronização de novo.

```
1 def saveClientMD(self):
2     self.vm.downResource()
3     md = self.vm.getMD()
4     data = {'action': 'saveClientMD',
5             'data': md}
6     self.js.sendObj(data)
```

```
1 def restoreMD(self, md):
2     mdLocation = "/tmp/md" + self.vmName
3     with open(mdLocation, "w+") as f:
4         f.write(md)
5     #...
```

Para que o utilizador consiga parar (ver figura 3.5) e retomar, assim como saber qual o estado da sincronização, foi criado um agente que deverá estar instalado em todas as máquinas virtuais.

Ver Figura 3.6 [1].

Este agente usa uma porta série [12] para comunicar com o servidor. É utilizada uma porta série em vez de uma ligação TCP, para tornar o sistema mais resistente a falhas e/ou diferentes configurações por parte do *desktop* do utilizador.

3.6 Migração

Usando o agente presente na máquina virtual, o utilizador poderá iniciar a migração, bastando para isso clicar na seta direcionado para o computador. Ver

[1] Pode ser observado nesta imagem algumas funcionalidades que ainda não foram abordadas. *Reboot + native* vai ser abordado no Capítulo 4. *Share HW* é uma funcionalidade que permite partilhar hardware presente no posto cliente para a máquina virtual, no entanto encontra-se numa fase muito inicial pelo que não foi documentada. Existem ainda mais duas opções (*Log* e *Dev*) presentes apenas para ajudar no desenvolvimento da aplicação.

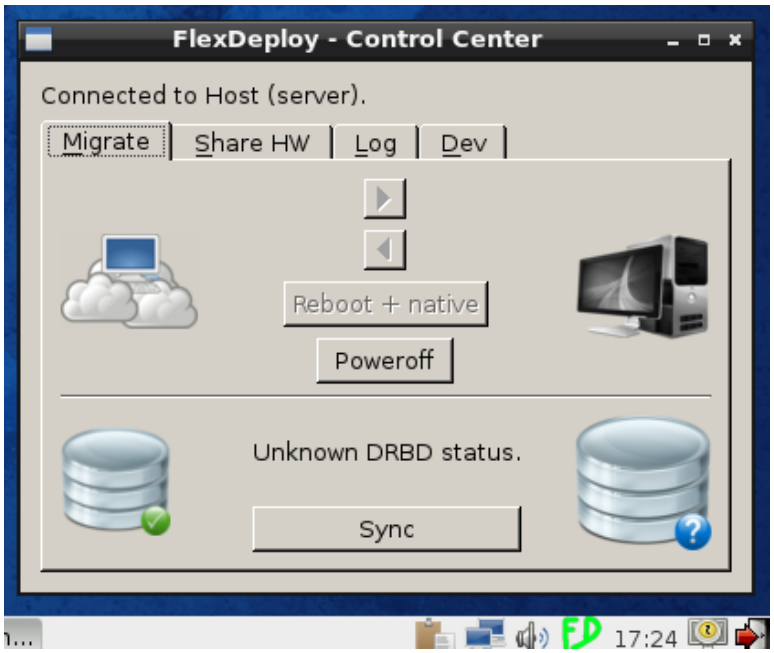


Figura 3.5: FlexDeploy - Agente (servidor) - Sincronização desligada

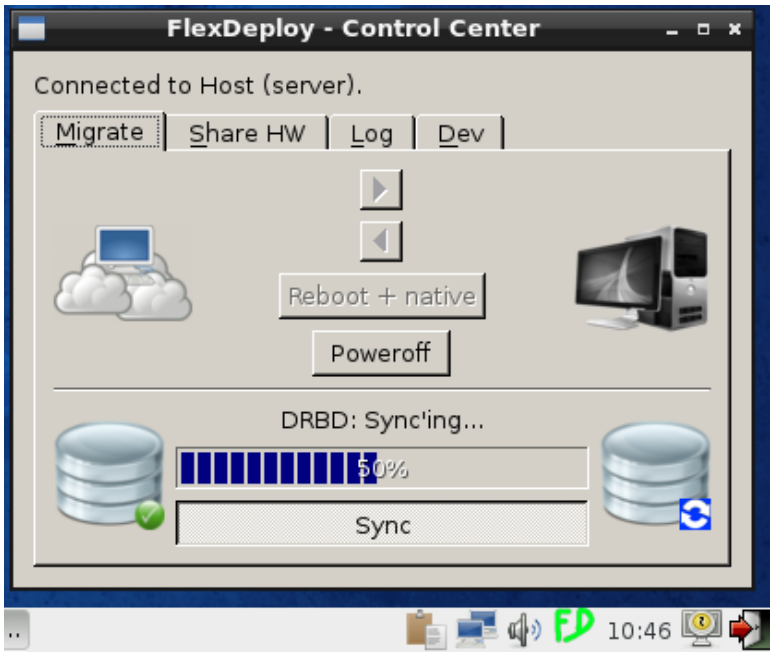


Figura 3.6: FlexDeploy - Agente (servidor) - Sync

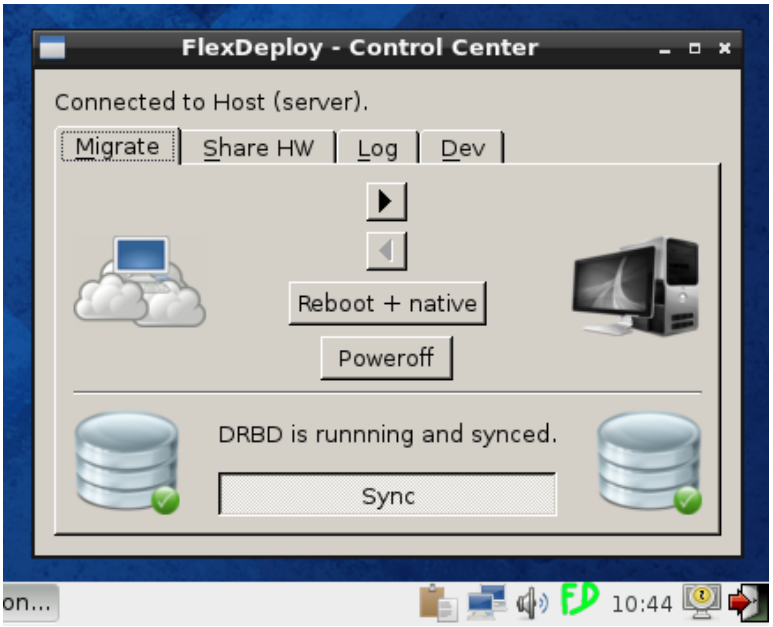


Figura 3.7: FlexDeploy - Agente (servidor) - Sincronização completa

Figura 3.7. A migração acontece quase em tempo real e com mínima interrupção para o cliente.

O processo de migração exige dois discos ativos, no entanto o *DRBD* em modo assíncrono só permite um disco activo. É então necessário mudar o protocolo para modo C (síncrono).

```
1  def startMigration(self):
2      #we need Primary/Primary
3      #change to sync protocol (C) and allow 2 primaries
4      args = "drbdadm␣net-options␣--protocol=C␣--allow-two-
           primaries=yes␣r" + \
5              self.currentVMInfo['name']
6      ProcessRunner(args).runWaitFinish()
7
8      #dual primary
9      data = {'action': 'dualPrimary'}
10     self.js.sendObj(data)
```

Após a alteração para modo síncrono, é necessário que os blocos em espera sejam escritos antes de iniciar a migração.

A migração é iniciada após sincronização completa.

```
def liveMigrate(self):
    #...
    args = 'virsh migrate --verbose --live ' + self.vmName + \
           ' qemu+ssh://' + self.ipClient + '/system --p2p --
              tunnelled'
    self.pri = PRI(args)
    self.showPercentMigrating = True
    self.pri.processEnded.connect(self.handleMigrateCallResult)
    self.pri.start()
    self.getMigrationProgress()
```

Após terminado, o processamento da máquina virtual passa completamente para o posto cliente. Esta alteração permite uma maior fluidez na interação do utilizador com o seu *desktop* devido a uma utilização muito reduzida da rede.

O serviço de replicação passa novamente a ter apenas um disco primário (localizado no cliente) e modo assíncrono para permitir uma melhor experiência para o utilizador.

Fica então concluída a migração do servidor para o cliente.

3.7 Virtualização Local

Para o utilizador, a localização do sistema que está a virtualizar o seu *desktop* é transparente, à exceção do agente que o informa se este está no posto do cliente ou no servidor.

Tal como na situação em que a máquina virtual está a ser processada no servidor, também na virtualização local é possível parar a sincronização.

A qualquer momento (desde que esteja sincronizado) o utilizador pode migrar novamente a máquina para o servidor. O processo é relativamente igual ao descrito anteriormente. Essa ação pode ser iniciada clicando na seta direcionada para o servidor, presente no agente da máquina virtual. Ver Figura 3.8.

Quando a máquina virtual está presente no cliente e existe uma falha de ligação o sistema pode ficar instável. É necessário reestabelecer a ligação antes de desligar o posto do cliente. Reiniciando a ligação vai permitir ao sistema recuperar o serviço *DRBD* e começa novamente a sincronização.

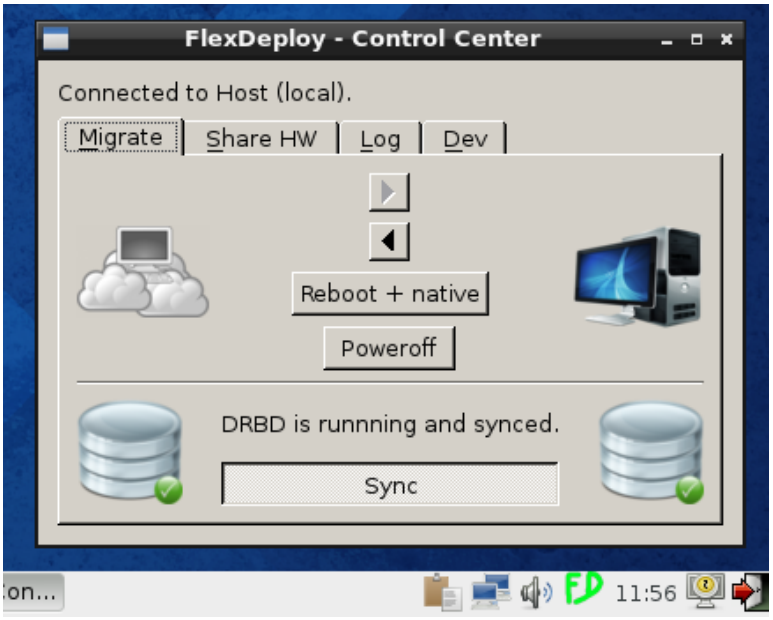


Figura 3.8: FlexDeploy - Agente (cliente) - Sincronização completa

Sumário

Este capítulo detalhou a implementação de objetivos delineados inicialmente: virtualização remota e virtualização local.

Em primeiro lugar foi descrito o problema, a arquitetura da solução assim como as ferramentas utilizadas. Por fim foi documentado o estado inicial e os passos necessários para atingir o desejado estado final: virtualização local.

Capítulo 4

Utilização Nativa

Utilização nativa do sistema implica conseguir arrancar a máquina virtual num sistema real e livre de qualquer virtualização.

Este capítulo vai descrever como é possível atingir esse objetivo. Assim como, após fazer alterações, replicá-las para o servidor.

4.1 Problema

O utilizador deverá ser capaz de utilizador o seu *desktop* do mesmo modo que usaria um posto normal. Todo o processamento será realizado diretamente no posto físico. Para que tal seja possível é necessário sincronizar o disco virtual no disco do posto, de modo a posteriormente ser possível iniciar o posto com o conteúdo deste.

Após utilização nativa do sistema, o utilizador deverá ser capaz de propagar as alterações para o servidor. Num próximo arranque, estas alterações deverão estar presentes.

4.2 Sincronização

O serviço de replicação usado, e seguindo a configuração apresentada em 3.4.1 usa o esquema presente na Figura 4.1 para armazenar os meta-dados necessários para manter a sincronização.

Figura 4.1: FlexDeploy - Nativo - Problema

Como pode ser observado, o disco além dos dados da máquina virtual, contém outros dados. O pretendido é arrancar o sistema pelo disco da máquina virtual. No entanto existe um problema com esta configuração. O sistema apenas vai detetar os meta-dados e não vai ser possível arrancar a máquina virtual.

Quando o computador iniciar e tentar arrancar pelo disco, este vai tentar ler o primeiro sector [3], no entanto a informação aí presente é referente aos meta-dados.

O serviço *DRBD* suporta colocar meta-dados num disco diferente do que é usado para armazenar os dados [5].

Uma possível solução para esta situação passaria por adicionar mais um disco ao posto do cliente. Esta solução, apesar de eficaz, pode nem sempre ser possível devido a limitações físicas, económicas e/ou outras.

A solução seguida passa por usar discos virtuais criadas na memória RAM. Ver Figura 4.2. Para as aplicações do posto cliente, nomeadamente o serviço *DRBD*, estes discos virtuais têm as mesmas características que discos físicos. Na criação da imagem do sistema operativo para o posto cliente é necessário indicar o tamanho que o *ramdisk* vai ter. Usar um disco para colocar meta-dados implica

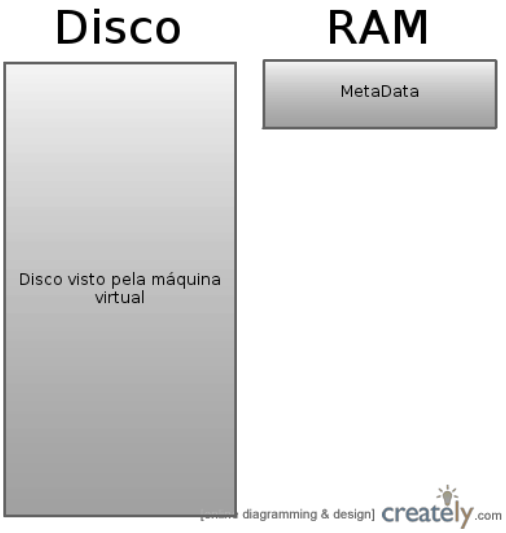


Figura 4.2: FlexDeploy - Nativo - Usar *ramdisk*

que este tenha um tamanho mínimo de 128MB (permite sincronizar um volume com um tamanho máximo de 4TB) [4]. Acrescenta-se, então, o seguinte parâmetro na linha de arranque da máquina virtual

```
ramdisk_size=134857 #valor em bytes, ligeiramente superior a 128 MB
```

O ficheiro de configuração é ligeiramente alterado de modo a usar a nova localização dos meta-dados.

```
resource rF20 {
    net{
        protocol A;
        after-sb-1pri discard-secondary;
    }
    device /dev/drbd1;
    disk{
        size 20G;
    }
    on server {
        disk      /dev/sdb1;
        meta-disk internal;
        address   10.10.3.23:7789;
    }
```

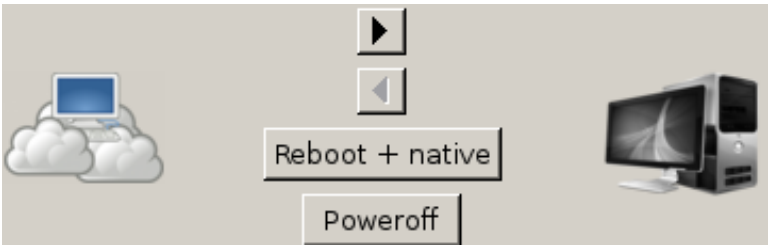


Figura 4.3: FlexDeploy - Agente (servidor)

```
15      on live {
16          disk       /dev/sda;
17          meta-disk /dev/ram0; #antes: meta-disk internal;
18          address    10.10.3.22:7789;
19      }
20  }
```

4.3 De Virtual a Nativo

Com as alterações efetuadas, o processo de sincronização desenrola-se do mesmo modo que anteriormente, à exceção que agora o disco da máquina virtual é igual ao disco presente na máquina física.

Após sincronização completa, o utilizador pode usar o agente para reiniciar a máquina e entrar em modo nativo. Ver Figura 4.3. Ao reiniciar a máquina, esta deverá iniciar pelo disco usado pelo *DRBD*.

O utilizador poderá assim tirar partido completo do posto.

4.4 Envio do Estado Atual

Após usar a máquina, esta poderá ser novamente sincronizada para o servidor.

Para tal, é necessário iniciar a máquina novamente usando o cliente e fazer *login*. Após *login* é dada a possibilidade de usar uma máquina previamente criada, ou de fazer upload do posto atual para uma das máquinas do servidor. Selecionando a segunda opção, o sistema cria as configurações para o serviço *DRBD* e começa a sincronizar com o servidor.

Durante a sincronização o posto cliente inicia a virtualização da máquina e permite o acesso desta ao utilizador. É assim possível ao utilizador o seu *desktop* enquanto este é sincronizado para o servidor.

Após o término da sincronização, é possível migrar a máquina para o servidor, continuar a execução da mesma ou desligar por completo o sistema (máquina virtual e cliente).

Este processo torna possível a criação de novas imagens. Para tal, o utilizador deverá iniciar o cliente na máquina que pretende tornar a sua imagem e escolher a opção de envio do estado atual. Após terminar o processo, o posto atual fica disponível para uso em outros postos cujo utilizador tem acesso.

Sumário

Este capítulo demonstrou como é possível transpor uma máquina virtual para um posto físico.

Inicialmente foi apresentada a solução para a necessidade de mais um disco no posto do cliente. Tornando assim possível criar uma cópia exata do disco da máquina virtual, na máquina física. Foram também explicados os procedimentos necessários para que o utilizador possa iniciar o processo que torna a máquina virtual em real.

Por último, é descrito como se pode enviar o estado modificado da máquina para o servidor, não perdendo qualquer alteração efetuada durante a utilização nativa da máquina.

Fazer *deploy* de um sistema operativo para uma máquina por rede, normalmente requer arranque por *PXE* o que implica acesso à *layer* 2 da rede (camada de ligação de dados). Este método permite fazer *deploy* de máquinas cujo acesso ao servidor não é feito em *layer* 2 mas num nível superior, nomeadamente ao nível da camada de transporte - *layer* 4.

Capítulo 5

Conclusão e Trabalho Futuro

5.1 Conclusão

O aumento do número de computadores no contexto organizacional eleva os custos operacionais da organização. Atendendo a este facto, esta tese tem como objetivo criar um sistema capaz de possibilitar ao utilizador o seu *desktop* em qualquer momento, desde que possua um computador com acesso ao servidor. Posteriormente, poderá usar o seu *desktop* instalado no posto onde se encontra, tirando, assim, partido dos recursos disponíveis deste.

Os principais objetivos desta tese foram alcançados. Foi construído um sistema em que o utilizador é capaz de ter acesso ao seu *desktop*, migrá-lo para o posto local e possibilidade de o usar nativamente. O protótipo está funcional e o agente está disponível em dois ambientes diferentes: Windows e Linux.

Existem ainda alguns possíveis melhoramentos que podem ser explorados e que irão ser descritos em seguida.

5.2 Trabalho Futuro

Um conjunto de outras funcionalidades e melhoramentos podem ser realizados.

Permitir escolher disco para o qual se pretende sincronizar Atualmente o sistema faz uso do primeiro disco que encontrar no posto do utilizador. Este

comportamento pode ser problemático no caso de o utilizador desejar que o seu disco não seja alterado. A proposta de melhoramento deste aspeto passar por pedir autorização para usar o disco, e permitir que ele escolha qual deve ser utilizado no caso de existirem múltiplos.

Permitir não iniciar a sincronização O utilizador poderá não desejar sincronizar o seu *desktop* para o seu posto físico. Deverá ser questionado sobre tal ação, possivelmente aquando da interação referida no ponto anterior.

Gestão centralizada do sistema Como forma de tornar o sistema possível de ser utilizado numa organização, este deverá ter serviço que permita gerir utilizadores assim como as máquinas destes. Este serviço poderá ainda albergar gestões mais complexas de utilizadores, nomeadamente no que diz respeito a criação de grupos de utilizadores com privilégios diferentes, criação e manutenção de máquinas virtuais entre outras funções.

Deduplicação Apesar de abordado inicialmente, não foi possível implementar nenhum sistema de deduplicação. Uma possível abordagem passaria por implementar um sistema que trabalhasse em conjunto com o *DRBD*.

Partilha de hardware através da Internet Permitir que o utilizador consiga usar *hardware* a partir do posto, estando o seu sistema a ser virtualizado no servidor seria um dos próximos passos a abordar. Usando o projeto USBIP [24] foi possível, de forma experimental, partilhar alguns dispositivos USB. No entanto, devido a ter ainda algumas falhas não foi possível tornar o seu funcionamento fiável.

Implementar outros sistemas de Remote Desktop Existem vários protocolos otimizados para diferentes *desktop*. Para ambientes Microsoft Windows pode ser usado o protocolo RDP, que permite uma maior performance. Para ambientes Linux existe o protocolo NX que trabalha sob o servidor de *X*. Qualquer um destes protocolos trabalha após o arranque o sistema operativo, não permitindo ao utilizador ter acesso imediatamente após este ter iniciado (poderá ser útil para

diagnosticar algum problema durante o arranque). Uma abordagem, que chegou a ser testada, seria usar o protocolo otimizado e quando não possível voltar para o protocolo *SPICE*.

Bibliografia

[1] Art Wittmann. Calculating The True Cost Of VDI, 2013. URL http://www.networkcomputing.com/next-generation-data-center/commentary/storage/calculating-the-true-cost-of-vdi/240152954.

[2] Austin T. Clements, Irfan Ahmad, Murali Vilayannur, and Jinyuan Li. Decentralized Deduplication in SAN Cluster File Systems.

[3] DEW Associates Corporation. The Master Boot Record (MBR) and Why is it Necessary? URL http://www.dewassoc.com/kbase/hard_drives/master_boot_record.htm.

[4] DRBD. drbd.conf, . URL http://www.drbd.org/users-guide/re-drbdconf.html.

[5] DRBD. DRBD Internals: External meta data, . URL http://www.drbd.org/users-guide/ch-internals.html#s-external-meta-data.

[6] DRBD. Configuring your resource, . URL http://www.drbd.org/users-guide/s-configure-resource.html.

[7] DRBD. Replication modes, 2014. URL http://www.drbd.org/users-guide/s-replication-protocols.html.

[8] Martin Hagström. Remote desktop protocols: A comparison of Spice, NX and VNC.

[9] K. Jin and E. L. Miller. The Effectiveness of Deduplication on Virtual Machine Disk Images. *Proceedings of International Systems and Storage Conference (SYSTOR)*.

[10] Jorge Orchilles. Virtualization: The benefits of VDI, 2013. URL http://technet.microsoft.com/en-us/magazine/dn170431.aspx.

[11] JSON. URL http://json.org.

[12] libvirt. Domain XML format. URL http://libvirt.org/formatdomain.html#elementCharSerial.

[13] LINBIT. DRBD, 2014. URL http://www.drbd.org/.

[14] M. Tim Jones. High availability with the Distributed Replicated Block Device, 2010. URL http://www.ibm.com/developerworks/library/l-drbd/.

[15] João Paulo and José Pereira. A Survey and Classification of Storage Deduplication Systems.

[16] PyQt. URL http://www.riverbankcomputing.co.uk/software/pyqt/intro.

[17] QEMU/KVM. URL http://wiki.qemu.org/KVM.

[18] Tristan Richardson. URL http://www.realvnc.com/docs/rfbproto.pdf.

[19] Dilip Simha. RPE: The Art of Data Deduplication. .

[20] Dilip Simha. Demystifying Data Deduplication. .

[21] SPICE, . URL http://www.spice-space.org/.

[22] SPICE, . URL http://www.spice-space.org/features.html.

[23] StarWindSoftware. Data Deduplication, 2014. URL http://www.starwindsoftware.com/features/data-deduplication.

[24] USB/IP Project. URL http://usbip.sourceforge.net/.

[25] VMware. Understanding Full Virtualization, Paravirtualization, and Hardware Assist.

[26] VNC. URL http://www.hep.phy.cam.ac.uk/vnc_docs/index.html.

[27] Wei Zhang, Hong Tang, Hao Jiang, Tao Yang, Xiaogang Li, and Yue Zenf. Multi-level Selective Deduplication for VM Snapshots in Cloud Storage.