João Gonçalo Póvoa Tavares Oliveira

Licenciatura em Engenharia Informática

Sistema de Replicação para VMs Armazenadas em OBS

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Paulo Lopes, Professor,
Universidade NOVA de Lisboa

Júri

Presidente: Doutor Ricardo João Rodrigues Gonçalves
Vogais: Doutor José Henrique Pereira São Mamede
Doutor Paulo Orlando Reis Afonso Lopes

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Fevereiro, 2021

Sistema de Replicação para VMs Armazenadas em OBS

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

Agradecimentos

A realização deste trabalho teria sido impossível se não existisse o constante apoio das pessoas envolvidas neste projecto.

Em primeiro lugar gostaria de deixar um agradecimento ao Departamento de Informática e à Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, que permitiu que todo este trabalho fosse realizado em conjunto com o grupo da SolidNetworks – Business Consulting, Lda da empresa Reditus S.A. e financiado pelo programa PORTUGAL2020 para o projecto iCBD (POCI-01-0247-FEDER- 011467).

Gostaria de reconhecer e agradecer o acompanhamento do meu orientador Professor Paulo Lopes que forneceu um apoio essencial para realizar este trabalho, mostrando-se sempre disponível para discutir e resolver eventuais problemas, fornecendo uma motivação inigualável.

Também é necessário reconhecer o trabalho realizado pelo Dr. Miguel Martins da Reditus S.A., que foi fundamental em toda a fase prática do projecto. As várias reuniões realizadas proporcionaram uma grande fonte de conhecimento e ideias, e permitiram levar avante toda a implementação deste projecto.

É importante também destacar e agradecer o apoio dado pelo Mestre Luís Silva, que se mostrou sempre disponível para apoiar o desenvolvimento do trabalho.

Por fim gostaria de agradecer à minha família, e amigos que foram sempre um pilar importante neste percurso incentivando-me a chegar mais longe e a terminar este trabalho.

Resumo

Cada vez mais as tecnologias de virtualização tendem a ser usadas, nas organizações, pelos benefícios que podem trazer; destes destacamos, a título de exemplo, a redução de infraestruturas (e custos) conseguida pela via da consolidação: de servidores, de armazenamento e de rede. Nesta linha, o projeto iCBD (*Infrastructure for Client-Based Desktops*) visa a implementação de uma VDI (*Virtual Desktop Infrastructure*) para fornecer ambientes de trabalho virtualizados aos seus utilizadores. Ao contrário de outras VDIs, a iCBD tira partido dos recursos do posto de trabalho (PC, *laptop*) do utilizador para executar as máquinas virtuais (VMs), permitindo um melhor aproveitamento dos recursos disponíveis e reduzindo consideravelmente os custos.

A iCBD pretende ser uma infraestrutura escalável, capaz de suportar eficientemente tanto pequenas organizações, constituídas por uma única localização (*site*), como grandes organizações, geograficamente dispersas por múltiplas localizações. Uma tal infraestrutura necessita, para oferecer aos utilizadores um desempenho adequado, assim como tem de exibir alta disponibilidade, i.e., ser capaz de continuar a oferecer os seus serviços apesar de eventuais indisponibilidades ao nível das comunicações inter-*sites*. Tal exige que cada *site* possa integrar um serviço de replicação, capaz de o manter a funcionar se se encontrar desconectado da restante infraestrutura.

A motivação para o presente trabalho é dupla: a) dotar a iCBD de um sistema de armazenamento escalável e tolerante a faltas, baseado em Object-Based Storage; b) a construção de um sistema de replicação com as características supra-mencionadas que suporte, em cada localização iCBD, o armazenamento consistente de imagens de VMs e a criação dos respetivos clones, ou instâncias. Como este trabalho está incluído num projeto P2020 financiado, e nele inserido na linha de investigação de sistemas de armazenamento de objetos para suportar VMs, o nosso objetivo é o de conseguir realizar um serviço que suporte a replicação de objetos - imagens e suas instâncias - na iCBD.

Palavras-chave: Virtualização, Virtual Desktop Infrastructure, VDI, Client-Based VDI, Virtualização de Desktops, Sistema de armazenamento de Objetos, OBS

Abstract

Virtualization technologies have increasingly been used by organizations for the benefits they can bring; among these we highlight, as an example, the reduction on the infrastructure size (and its costs) obtained as a result of consolidation: of servers, storage and network. In this vein, iCBD (Infrastructure for Client-Based Desktops) is an implementation of a Virtual Desktop Infrastructure (VDI) that provides virtual desktops to its users. Differently from other VDIs, iCBD utilises the user's devices, such as a PC or laptop, to execute virtual machines (VMs), thus making the best use of available resources.

iCBD aims to be scalable, capable of efficiently support both small and large companies, ranging in size from a single site to multiple, geographically distributed, locations. Furthermore, it should also be highly available, i.e., able to provide its services even in the presence of faults that may hinder inter-site communications. This, on the other hand, requires a service that provides replication, to deal with disconnected operation.

The motivation for the present work is, in short, twofold: a) providing a scalable storage service to iCBD, based on Object-Based Storage, and b) the development of a replication (with the abovementioned characteristics) system able to support, in each iCBD site, a consistent store for VM images, and the creation of their respective clones, or instances. As this work is part of a P2020 project grant and also a task in a research line on the use of Object Storage to support VMs, our objective is to implement a service that supports replication and caching of objects - VM images and its instances - in iCBD.

Keywords: Virtualization, Virtual Desktop Infrastructure (VDI), iCBD, Client-Based VDI, Desktop Virtualization, Object Storage, OBS

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contexto

As técnicas de virtualização adicionam um nível de abstracção que permite a separação e partilha dos recursos disponíveis numa única máquina física para a criação de vários ambientes virtuais.

Cada vez mais as tecnologias de virtualização tendem a ser usadas nas organizações, pelos benefícios que podem trazer; destes destacamos, a título de exemplo, a redução conseguida na exaustiva e muitas vezes dispendiosa manutenção das infraestruturas, redução essa conseguida pela via da consolidação: de servidores, de armazenamento e de rede.

Mas não só ao nível das infraestruturas os benefícios da virtualização se podem fazer sentir: ela tem cada vez mais um importante papel junto aos dispositivos dos utilizadores. Como é sabido, um ambiente de trabalho (*Desktop*) oferece ao utilizador uma interface gráfica que permite uma interação mais fácil e intuitiva. Este modelo, ao nível do sistema operativo, pode providenciar uma apresentação de um ambiente onde o utilizador consegue manipular um sistema de ficheiros (podendo criar, editar, e apagar ficheiros) assim como executar diversas aplicações utilizando diferentes janelas para exibir informação.

A combinação do conceito de *Desktop* com as tecnologias de virtualização permite oferecer vantagens adicionais na criação de ambientes de trabalho possibilitando, por exemplo, a sua execução numa extensa diversidade de dispositivos. Nesta linha, o projeto iCBD visa a implementação de uma *Virtual Desktop Infrastructure* (VDI), i.e., fornece ambientes de trabalho virtualizados aos seus utilizadores; a iCBD tira partido dos recursos do posto de trabalho (o PC) do utilizador para executar as máquinas virtuais (VMs), permitindo o melhor aproveitamento dos recursos disponíveis, abordagem que contrasta com uma outra, mais comum e adotada pelos principais atores nesta área - Citrix[1],

Microsoft[2] e VMware[3] - na qual um ou mais servidores executam as máquinas virtuais "dos utilizadores", cujo *Desktop* é depois acedido através de um protocolo de ecrã remoto (*remote desktop*) como é mostrado na Figura 1.1. Estes servidores têm de ser poderosos em recursos, tornando-os dispendiosos tanto na compra como na sua manutenção além disto a escalabilidade é bastante reduzida quando existe um grande aumento no número de clientes.

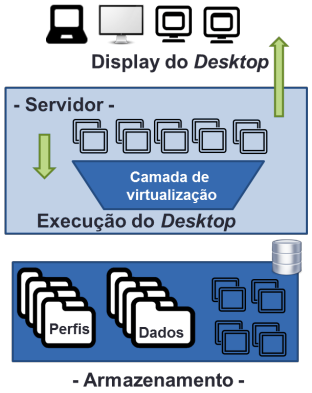


Figura 1.1: *Virtual Desktop Infraestructure* [4]

A utilização destas tecnologias oferece mais benefícios para além de introduzir uma grande compatibilidade com diversos dispositivos. Estas permitem simplificar bastante o trabalho de um administrador, que numa situação normal onde existem múltiplos postos de trabalho, cada um tem que sofrer um conjunto de procedimentos (instalação de SOs e várias peças de software assim como futuras actualizações) para que este esteja totalmente pronto a utilizar, este processo pode tornar-se bastante demoroso. As tecnologias de virtualização (como as VDIs) permitem amortizar significativamente o trabalho do processo de manutenção dos postos de trabalho. Muito suscitamente, as máquinas virtuais usadas pelos clientes, são geradas a partir de um *template* que contém toda a informação de uma máquina virtual, desde das configurações do hardware da máquina até ao disco que contém toda a informação do software nela instalado. Assim, a administração dos postos de trabalho passa pela gestão de um pequeno conjunto de *templates* disponibilizados na infraestrutura, simplificando significativamente todo o processo.

1.2 Motivação

A virtualização é a tecnologia que permite criar vários ambientes simulados ou recursos dedicados a partir de um único sistema de hardware físico, as *Clouds* usam esta tecnologia para criar um ambiente TI capaz agrupar e partilhar recursos computacionais através da Internet.

Os grandes fornecedores de serviços em *Cloud* como a AWS[5], Google Cloud[6] ou IBM Cloud[7] dispõem de sistemas de armazenamento (proprietários) robustos, dotados

de vários mecanismos que permitem suportar eficientemente um ambiente de virtualização, onde múltiplas máquinas virtuais estão constantemente a ser criadas ou destruídas conforme a necessidade dos clientes. Um dos mecanismos mais comuns é a utilização de *thin-clones* baseados em *copy-on-write*, que permite criar instantaneamente várias cópias um grande volume de dados sem gerar duplicação de dados.

Além disto, estas empresas têm diversos *sites* dispersos geograficamente para garantir a integridade e disponibilidade dos dados assim como oferecer uma melhor qualidade de serviço ao colocar *sites* junto das áreas com mais utilizadores.

O Ceph[8] é um sistema de armazenamento de objetos baseado em software aberto que tem ao dispor várias ferramentas que permitem suportar uma infraestrutura de virtualização como a iCBD com provas em estudos já realizados no âmbito deste trabalho [9]. Este tem como base um *storage cluster* (agregado de nós) altamente escalável dotado de mecanismos de deteção e recuperação de faltas. Além disto este possuí outras ferramentas como *snaphots* e *thin-clones* que são essenciais para suportar um ambiente de virtualização.

A integração do Ceph na iCBD pretende torná-la uma plataforma altamente escalável que se adapte tanto a pequenas como grandes organizações compostas por vários *sites* dispersos geograficamente. No entanto, um *cluster* Ceph não consegue estar geograficamente separado, dado que os nós estão constantemente a trocar dados (para realizar tarefas de recuperação por exemplo) é importante que os nós estejam relativamente próximos de forma a reduzir ao máximo a latência das comunicações. Por esta razão, vai-se tirar partido da replicação para criar um sistema que permita manter consistente vários *clusters* Ceph que vão servir os diversos *sites* iCBD.

1.3 Projecto iCBD

A *Infrastructure for Client-based Descktop Virtualization* (iCBD) [10] é um projeto de investigação e desenvolvimento, realizado por uma equipa mista da SolidNetworks – Business Consulting, Lda (do grupo Reditus, S.A.) e do Centro de Investigação NOVA LINCS da FCT/NOVA, financiado pelo programa P2020 (projeto Nº 11467). Está já operacional, dispondo de uma arquitetura que oferece o armazenamento e distribuição de imagens de VMs que são instanciadas e executadas nos dispositivos dos utilizadores, retirando a sobrecarga da computação dos servidores, reduzindo o custo da infraestrutura, e facilitando a manutenção de todo o sistema.

Como se pode facilmente inferir ao observar a Figura 1.2, até chegarmos à utilização de um *desktop* virtual, sucedem-se - muito resumidamente - as seguintes fases: i) início do arranque do PC, solicitação de uma imagem a executar (no exemplo da figura, Windows 10); ii) carregamento de um sistema Linux com um hipervisor embutido; iii) criação, num servidor, de uma instância da imagem (no exemplo, Windows 10) escolhida e partilha na rede dessa mesma instância; iv) execução no posto de trabalho da instância.

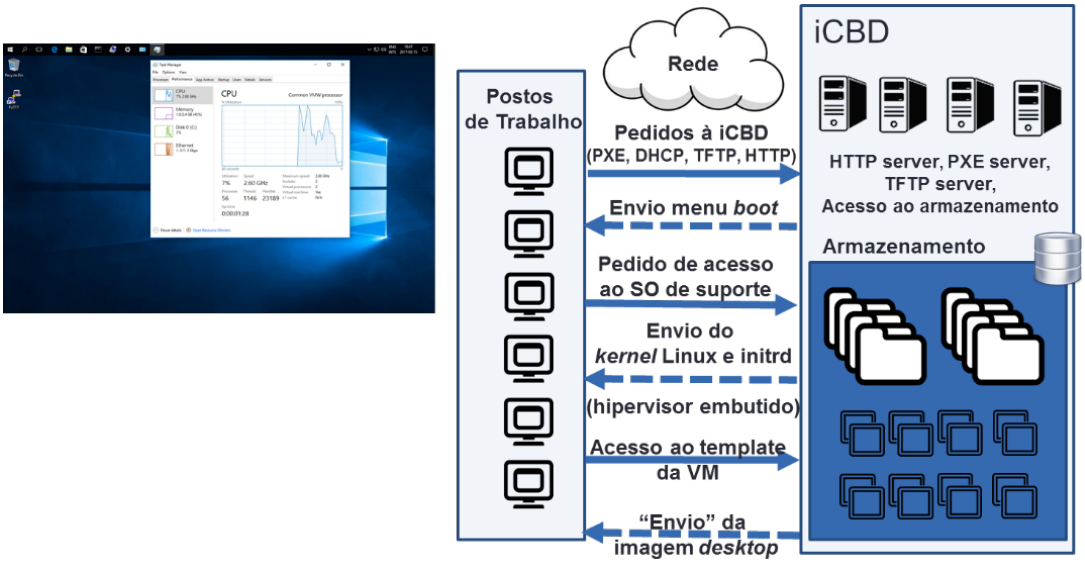


Figura 1.2: Processo de execução da iCBD [4]

Assim, na iCBD os aspetos-chave são os que se relacionam com o armazenamento: das imagens [1] (*templates*, para sermos mais precisos), e das instâncias, criadas em tempo de execução para suportar as distintas VMs que "nascem" de um mesmo *template* por clonagem.

No projeto iCBD, no âmbito do armazenamento e distribuição de imagens e de instâncias, desenrolam-se duas linhas de investigação complementares: uma, na qual o armazenamento é realizado sobre sistemas de ficheiros; outra, na qual o ele é realizado sobre sistemas de armazenamento de objetos. A primeira linha de investigação já produziu, como resultados, um sistema iCBD completamente funcional [10] e duas teses de mestrado [4] e [11] assim como um artigo[12]. A segunda linha de investigação suportou a realização de uma tese de mestrado [9] mas os resultados, incompletos, ainda não se encontram transpostos para o protótipo.

1.4 Objetivos

A iCBD pretende ser uma infraestrutura escalável, capaz de suportar eficientemente tanto pequenas organizações, constituídas por uma única localização (*site*), como grandes organizações, geograficamente dispersas por múltiplas localizações interligadas por uma infraestrutura de comunicações com uma largura de banda da ordem das muitas centenas de Mbps ou, de preferência, Gbps. Uma tal infraestrutura necessita, para oferecer aos utilizadores um desempenho adequado, de incluir níveis de *caching*; paralelamente, a iCBD tem de exibir alta disponibilidade, i.e., ser capaz de continuar a oferecer os seus serviços apesar de eventuais indisponibilidades ao nível das comunicações inter-*sites*. Tal

[1] Este e outros conceitos relacionados com as tecnologias de virtualização serão abordados com maior detalhe no próximo capítulo

exige que, cada *site* iCBD possa integrar um serviço de replicação capaz de o manter a funcionar se se encontrar desconectado da restante infraestrutura, mas capaz também de promover a reconciliação quando o *site* for de novo re-conectado.

Assim, o objetivo principal do presente trabalho é a construção de um sistema de replicação com as características supra-mencionadas, que suporte, em cada *site* iCBD, o armazenamento consistente de imagens e a criação dos respectivos clones ou instâncias em *clusters* Ceph.

1.4.1 Contribuições Esperadas

As principais contribuições que se espera são as seguintes:

- Investigação e desenvolvimento de um sistema distribuído de replicação de objetos para o sistema de armazenamento de objetos Ceph.
- Integração das soluções no protótipo iCBD.
- Análise da integração do Ceph na iCBD através de testes de desempenho.
- Realização de testes de funcionalidade e desempenho do sistema de replicação e desenvolvido.

1.5 Estrutura do Documento

O resto do documento está estruturado da seguinte maneira:

- **Capitulo 2**: Neste capítulo estão descritas as várias tecnologias virtualização que têm um papel fundamental no trabalho desenvolvido, desde os hipervisores que suportam a execução de VMs, até ao armazenamento das próprias VMs.
- **Capitulo 3**: Dado que a infraestrutura da iCBD é algo complexo, neste capítulo é explicado com mais detalhe o funcionamento da iCBD. Este passa pela apresentação da arquitetura da iCBD, assim como a descrição do processo de arranque dos postos de trabalho.
- **Capitulo 4**: Este capítulo documenta a implementação da solução desenvolvida em conjunto com os seus requisitos.
- **Capitulo 5**: Este capitulo contém a descrição de todos os testes realizados para analisar a funcionalidade e desempenho do sistema de replicação assim como a nova integração do suporte para sistemas de armazenamento de objectos na iCBD.
- **Capitulo 6**: Por fim, neste capítulo estará a conclusão retirada de todo o trabalho realizado assim como trabalhos a realizar futuramente originados a partir deste.

Capítulo 2

Tecnologias de Virtualização

A introdução de tecnologias de virtualização possibilitou novos casos de uso para a implantação (*deployment*) de serviços de computação, vantajosos em relação ao modelo tradicional de um computador no qual os recursos físicos são controlados por um sistema de operação (SO) instalado diretamente sobre o hardware, no qual o SO distribui e fornece às aplicações os recursos necessários para a sua execução. Com a utilização da virtualização é possível criar e executar várias máquinas virtuais numa única máquina física, sendo que cada VM se comporta como um computador independente, com o seu próprio hardware (virtualizado), no qual se podem instalar um SO e aplicações, proporcionando ao utilizador uma experiência indistinguível da oferecida por um computador real.

A importância destas tecnologias é evidenciada quando analisamos os seus benefícios. As primeiras abordagens começaram pela virtualização de servidores, tendo como objetivo uma melhor utilização dos servidores físicos, ao permitir a rápida criação de novos servidores virtuais (pré-configurados) quando necessário. A utilização desta tecnologia num centro de dados, por exemplo, permite um muito melhor aproveitamento dos recursos disponíveis, o que se pode traduzir numa redução do número de infraestruturas necessárias, e consequentemente na diminuição dos custos de manutenção das mesmas. Visto existir uma maior concentração de muitos servidores (virtuais) em menor número de máquinas físicas, há ainda uma diminuição no consumo geral de energia da infraestrutura, assim como no espaço e arrefecimento necessários para suportar a mesma.

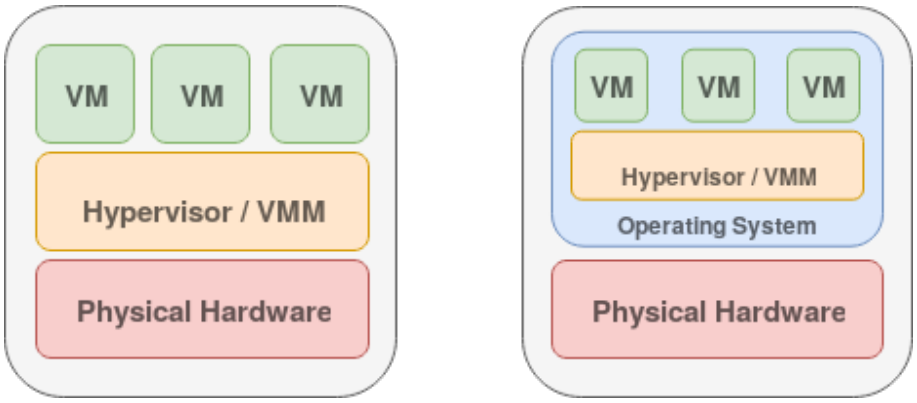
2.1 Tecnologia dos Hipervisores e Máquinas Virtuais

2.1.1 Hipervisores e VMs

Uma Máquina Virtual (VM) pode definir-se como uma abstração de um computador físico, com o seu próprio hardware (CPU, memória RAM, dispositivos de armazenamento e placa de rede, entre outros componentes). Na medida em que pode ser, para o software que nela é executado, indistinguível de uma máquina física, uma VM pode correr um sistema de operação, designado neste contexto como *guest operating system*, sobre o qual se executam as aplicações.

Para alguns autores [13] a referida abstração é implementada por um módulo de software designado *Virtual Machine Monitor* (VMM), que também controla a sua execução, sendo capaz de realizar operações tais como *Power-On*, *Power-Off*, *Reset*, suspender ou retomar a execução, etc.. A gestão dos recursos da máquina física e sua distribuição pelas VMs em execução fica a cargo de um "sistema de operação especializado", designado hipervisor.

O hipervisor é uma peça de software que assenta entre o hardware, ou *host*, e as VMs, ou *guests*, fornecendo-lhes todos os mecanismos e recursos cruciais para a sua execução e gerindo o hardware real. Na figura 2.1 são apresentados os dois tipos de hipervisores.



(a) Tipo I / *Bare-Metal* (b) Tipo II / *Hosted*

Figura 2.1: Tipos de Hipervisores

Os hipervisores de tipo 1, também designados nativos ou *bare-metal*, são executados diretamente no hardware, tendo total controlo dos recursos. Nesta categoria podemos destacar soluções como o VMWare ESXi [14], Linux KVM [15], Microsoft Hyper-V[16] e Citrix XenServer [17]. Estes hipervisores são os mais adequados para suportar aplicações empresariais, ambientes críticos, garantir segurança e isolamento entre VMs, e situações que requerem desempenho e fiabilidade elevados, entre outras. Cada um dos fornecedores acima mencionados tem disponível a lista de servidores certificados para suportar os seus hipervisores.

Por fim os hipervisores de tipo 2, ou hospedados, são aqueles que correm como processos num sistema de operação hospedeiro que, por esse motivo, serve de intermediário

e gere os recursos fornecidos ao hipervisor. Existem vários exemplos deste tipo de hipervisores, dos quais destacamos o Oracle VirtualBox [18] e o VMWare Workstation [19]. Estes hipervisores são geralmente instalados em PCs ou portáteis para suportar tarefas menos críticas tais como provas de conceito, experimentação de novas versões de produtos, desenvolvimento e teste de pequenas aplicações, etc.

2.1.2 Virtualização de Hardware

O hardware de um computador é gerido por um sistema de operação que assume o controlo total de todos os componentes, como CPU, memória RAM, dispositivos de armazenamento e de I/O. Para a correta execução de todos os processos o SO precisa de ter acesso completo a todo o hardware disponível, sendo-lhe reservada a possibilidade de executar algumas operações sensiveis no CPU, como o acesso aos periféricos, o estabelecimento de mapas de memória, o atendimento de interrupções, etc.; tais instruções são designadas privilegiadas, e só podem ser executadas quando o registo de estado do CPU indica que este está em modo supervisor, sendo a execução abortada se o registo indicar que o CPU está em modo utilizador.

A fim de providenciar um ambiente para a execução de várias VMs é necessário criar as abstrações que realizam as máquinas virtuais; uma forma simples de apresentar estas abstrações é imaginar que cada uma representa um recurso físico que é partilhado entre elas - por exemplo, um CPU virtual é idêntico ao CPU real, mas só está disponível para a VM uma fração do tempo; ou, a RAM de uma VM corresponde a uma fração da RAM da máquina física. Assim, quando um hipervisor quiser iniciar (ou retomar) a execução de uma VM ele "entrega-lhe" os recursos para que esta possa correr.

A descrição anterior permite evidenciar um problema potencial: para um hipervisor conseguir controlar adequadamente a execução das VMs, tem de usar instruções privilegiadas. Mas um sistema de operação que corre numa VM precisa também de usar essas instruções, para distinguir a sua (do SO) execução da das aplicações que suporta. Ora um único estado, binário, supervisor/utilizador não permite acomodar os dois requisitos. Mas isto parece fácil de resolver, se considerarmos que cada CPU virtual tem um conjunto de registos (virtuais) e tal conjunto inclui um registo de estado: então, quando uma VM está em execução e é executada uma instrução privilegiada, a execução da VM é suspensa e o controle passa ao hipervisor[1]; este, observa o registo de estado do CPU virtual da VM: se este indica modo supervisor, a execução do *guest OS* prossegue; caso contrário, é registada uma violação de privilégio (eventualmente por parte de uma aplicação) a ser tratada pelo *guest OS* quando a VM for de novo executada.

Infelizmente a realidade é bastante mais complexa: os processadores x86 e x86-64 da Intel e AMD, ao contrário dos processadores da IBM, não são *virtualization-friendly*: têm algumas instruções, das quais as mais referidas na literatura [13] são a PUSHF e POPF, que têm comportamentos distintos conforme estejam a ser executadas em modo

[1] Consideramos, a partir deste momento, que é desnecessária a destrinça entre hipervisor e VMM

utilizador ou supervisor; tais instruções, designadas sensíveis [20] são um sério entrave à virtualização eficiente das arquiteturas x86 e x86-64.

O problema pode ser resolvido usando três abordagens distintas [21]: virtualização completa (*full virtualization*), paravirtualização, e virtualização assistida por hardware.

A técnica de virtualização completa realiza a tradução binária das instruções x86 sensíveis noutras funcionalmente equivalentes, de forma que as primeiras nunca chegam a ser executadas no CPU real; naturalmente, as instruções não-sensíveis são "passadas" sem alteração ao CPU real.

A paravirtualização é uma técnica que recorre à alteração do núcleo do *guest OS*, expurgando-o das instruções sensíveis e substituindo-as por chamadas a funções do hipervisor; naturalmente, esta técnica permite obter melhor um melhor desempenho mas a necessidade de modificar o SO obriga a que o código fonte do mesmo seja disponibilizado, limitando a aplicabilidade desta solução.

Por fim a virtualização assistida por hardware é o resultado da implementação, ao nível do CPU, de mecanismos que permitem ultrapassar, de forma eficiente, os constrangimentos anteriores. Ao nível do CPU a virtualização assistida por hardware foi introduzida em 2006 com a tecnologia VT-x (Intel) ou AMD-V (AMD), e abraçou de seguida a memória com a tecnologia EPT (Intel) ou RVI (AMD), e continua a evoluir. Modernamente, os hipervisores de tipo 1 como o KVM ou o ESXi começam a requerer CPUs com suporte de virtualização assistida por hardware para funcionarem.

A virtualização de dispositivos de I/O, permitindo a interação com as VMs, é um aspeto essencial da virtualização. A abstração destes dispositivos, como NICs (placas de rede - *Network Interface Cards*) e outros periféricos, pode ser obtida através do mapeamento efetuado pelo hipervisor entre os componentes físicos e virtuais. Para além disso a utilização de software para simulação de alguns dispositivos como NICs e *switches* permitem emular as suas funcionalidades para comunicações inter-VMs sem sobrecarregar os componentes fisicos.

VMs e imagens

Sumariamente, uma VM é definida pela especificação dos recursos que a constituem (hardware, como o número de processadores, quantidade de RAM, tipos, características e número de periféricos - placas de rede, discos, etc. - mas também *firmware*, como o código do BIOS, parâmetros de configuração do mesmo, etc.). Alguns dos recursos são representados estaticamente por simples valores, como o número de processadores, a quantidade de RAM, ou o endereço MAC de uma placa de rede; outros, contudo, são de muito maior dimensão e têm de ser armazenados de forma persistente, seja em ficheiros ou volumes (discos). Um exemplo deste último é o SO que vai ser executado quando a VM arrancar[2]. A especificação da VM indica também a localização e nome dos ficheiros ou volumes que representam o(s) disco(s) da VM.

[2]Naturalmente a VM pode arrancar por outro dispositivo, e.g., placa de rede...

O termo imagem é usado muitas vezes com propósitos distintos; no contexto deste trabalho, imagem - ou mais precisamente imagem de um disco - é um suporte persistente de informação (disco, ficheiro, ou outro) que tem (ou pode conter) um conteúdo que representa uma cópia byte-a-byte de um volume (uma parte ou a totalidade de um disco) conteúdo esse que é, geralmente (e é nesse contexto que se usa a forma abreviada referindo unicamente a palavra "imagem"), um SO *bootable*.

Assim, um hipervisor pode (i) gerir/montar um sistema de ficheiros no qual um ficheiro pode ser exibido a uma VM como se fosse um disco físico, (ii) permitir que um volume (que pode ser um disco físico, um disco lógico, ou uma outra entidade capaz de armazenar os bytes da imagem - veremos mais tarde e mais detalhadamente este outro aspeto) seja diretamente acedido por uma VM. Assim, o *guest* OS que a VM vai correr poderá ser arrancado, correspondentemente, de um ficheiro, (i), ou de um outro tipo de armazenamento (ii).

2.1.3 Ciclo de vida das VMs

Uma das grandes vantagens dos ambientes virtualizados é a simplicidade com que se lida com o dinamismo: é muito fácil criar e destruir VMs, alterar as suas configurações, colocá-las em suspenso e reactivá-las.

Templates e Instâncias

No caso em que o disco-imagem da VM é representado por um ficheiro, criar uma imagem para uma outra VM é muito simples: "basta" fazer uma cópia do ficheiro, e indicar na especificação da nova VM que o caminho para o seu disco é o ficheiro-cópia entretanto criado. Esta operação designa-se por clonagem (da imagem), e os hipervisores fazem-na como parte integrante da clonagem de uma VM: cria-se uma nova VM, com especificações idênticas à original, e clona-se também o ficheiro-imagem. Naturalmente, alguns parâmetros da nova VM são alterados, como, e.g., os endereços MAC das placas de rede. Note-se, contudo, que o utilizador ainda tem, muito provavelmente, algumas tarefas a realizar, tais como mudar o *hostname* e endereço IP na imagem da nova VM.

Quando uma VM está em execução, o seu disco de sistema está constantemente a ser alterado, pois o SO escreve neste as alterações dinâmicas do estado dos processos, os *logs*, cria e apaga ficheiros temporários, etc.; assim, é usual ter-se uma imagem "limpa" a partir da qual todas as outras são clonadas. Uma tal imagem designa-se geralmente por *golden image*, sendo que a VMware usa o termo *template* para designar o par "especificação da VM"/*golden image* enquanto outros, como o OpenStack ou a Amazon AWS preferem manter as duas entidades independentes uma da outra, designando-as por *instance type* e *machine image*.

Quando se adopta esta filosofia de imagem-base, ou *template*, usa-se o termo instância para denotar uma VM que foi clonada a partir dessa imagem original - assim, teremos o conjunto de instâncias clonadas a partir de um mesmo *template*. A existência destas instâncias é também uma oportunidade para poupanças: uma vez que têm origem num mesmo

template, os seus discos podem não diferir, mesmo apesar das alterações em *runtime*, muito da imagem original. Isto é uma oportunidade para usar técnicas de de-duplicação de dados.

Full- e Linked-clones

A "cópia integral" de uma imagem para a criação de um clone designa-se por *full-clone*; contudo, os hipervisores são capazes de, em vez de copiar a imagem, criar a cópia usando um processo de *copy-on-write* da imagem original: as alterações realizadas na instância clonada são escritos na "imagem-cópia", mas os blocos[3] não modificados são obtidos da imagem original. Clones obtidos por este processo designam-se por *linked-clones*.

Snapshots

Uma outra funcionalidade relevante e presente nas tecnologias de virtualização é a possibilidade de tirar um *snapshot* a uma máquina virtual. Um *snapshot* é fundamentalmente uma cópia do estado da imagem no momento em que esta foi tirada, e isto é possível mesmo que a VM esteja em execução, pois neste caso é também guardando o estado da máquina virtual, por cópia da RAM e do estado do CPU.

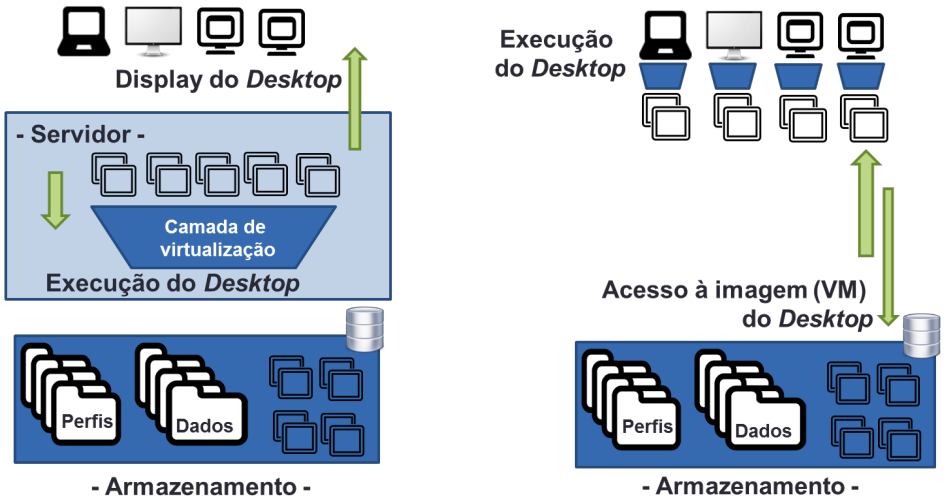
Os *snapshots* podem ser utilizados como cópias de salvaguarda (*backups*) a fim de prevenir futuras falhas que possam ocorrer, recorrendo-se então a *rollback* para se regressar a um estado anterior. Note-se que as falhas acima mencionadas podem incluir as da própria infraestrutura de virtualização: se o *template* e um ou mais *snapshots* da instância estiverem disponíveis, esta poderá ser recriada noutra infraestrutura - uma situação usualmente designada por *disaster recovery*.

2.2 *Virtual Desktop Infrastructures*

Com o desenvolvimento da virtualização, o número de aplicações desta tecnologia foi também aumentando, sendo de destacar a sua utilização com o objetivo de virtualizar o ambiente de trabalho (*Desktop Virtualization*). As infraestruturas criadas para este fim designam-se VDIs, *Virtual Desktop Infrastructures*.

As VDIs são uma proposta para reduzir os custos operacionais dos parques de computadores pessoais (PCs) nas organizações; tais custos decorrem fundamentalmente dos processos, ainda que automatizados, de instalação de software nos PCs, mas são sobretudo devidos à manutenção, às constantes atualizações e à resolução dos problemas que por vezes estas acarretam (compatibilidade entre versões, por exemplo), etc.. Nas VDIs os utilizadores não dispõem de máquinas físicas, mas sim virtuais, que, naturalmente, serão clones de *templates* pré-definidos pelos administradores da VDI. Assim, uns utilizadores terão VMs com Windows 10 e Office, outros utilizarão Windows 7 com aplicações empresariais que ainda não estão disponíveis em Windows 10, outros serão utilizadores de Ubuntu 16.04, outros de ... a lista pode ser longa.

[3] O termo blocos é aqui usado de forma livre, e não tem de corresponder a blocos do disco



(a) *Server-Based VDI* (b) *Client-Based VDI(iCBD)*

Figura 2.2: Tipos de VDIs [4]

A figura 2.2 distigue os dois tipos de VDIs: *Server-Based*(2.2a) e *Client-Based*(2.2b). No primeiro caso, a instanciação e execução das VMs ocorre num servidor, sendo a VM acedida no dispositivo do utilizador (PC, *laptop*, *tablet*) através de um protocolo capaz de exibir o ambiente de trabalho produzido pela VM (RDP, *Remote Desktop Protocol*). Neste tipo de VDI o poder computacional da infraestrutura tem de ser elevado, permitindo a instanciação de muitas máquinas virtuais em períodos de maior afluência.

No caso das *Client-based* VDIs, o utilizador usa o seu próprio dispositivo (PC ou *laptop*) para executar a VM. Esta técnica requer a presença de um hipervisor no PC ou *laptop* do utilizador, acesso ao *template* que se pretende instanciar, e espaço de armazenamento para albergar a instância clonada. O benefício, que decorre do facto de ser o dispositivo do utilizador a instanciar e correr a VM, é o de retirar grande parte da sobrecarga de processamento da infraestrutura, levando a uma melhoria significativa na performance e na redução dos custos dos equipamentos necessários.

2.3 Armazenamento de VMs

A forma de armazenamento utilizada pelos hipervisores depende muito do caso de uso. Em instalações com poucos nós (i.e., poucos servidores físicos) e sem necessidade de alta disponibilidade, o armazenamento em sistemas de ficheiros locais aos próprios nós (i.e., nos discos internos desses mesmos nós) é suficiente. Mas, se pelo contrário há necessidade de um sistema que seja tolerante a faltas e/ou altamente escalável e necessite de lidar com grandes quantidades de imagens, provavelmente vai necessário dispôr de um sistema externo para o armazenamento, baseado numa SAN (*Storage Area Network*) com *disk arrays* ou *Storage Clusters*, ou numa NAS (*Network Attached Storage*).

Seja qual for a tecnologia de armazenamento adotada, o hipervisor terá de, sobre a

mesma, gerir os *templates* e instâncias, pelo que precisará de as poder endereçar individualmente; para isso, dispõe de três alternativas: sistemas de ficheiros, sistemas de armazenamento de objetos, ou volumes numa SAN.

2.3.1 Sistemas de Ficheiros

Um Sistema de Ficheiros (SF), permite a criação, alteração e eliminação de ficheiros, assim como a sua organização em diretorias. Este sistema impõe uma hierarquia entre diretorias e ficheiros, permitindo a estruturação do espaço de armazenamento do utilizador, proporcionando uma ótima organização da informação guardada no sistema.

Um sistema de ficheiros pode classificar-se como sendo local a um único nó, sendo portanto gerido por módulos de software que correm nesse nó (usualmente no interior do núcleo do SO) ou distribuído, caso em que é usualmente acedido por múltiplos nós simultâneamente. Destes, ainda há que distinguir entre os SF assimétricos, dos quais os mais comuns são do tipo cliente-servidor, como o NFS (*Network File System*) [22], e os simétricos, também designados *cluster file systems*, dos quais são exemplos o Red-Hat GFS (*Global File System*) ou o VMFS (*Virtual Machine File System*) [23]. Os SF distribuídos são ainda designados por partilhados (*shared*) pois, sendo montados por múltiplos nós, permitem naturalmente a partilha de informação entre estes.

Os SF partilhados são especialmente interessantes no caso dos hipervisores pois permitem que cada nó tenha uma visão global dos repositórios de VMs, e isso é muito importante por duas ordens de razões: a) uma VM pode ser muito fácilmente executada num ou noutro nó, sem haver necessidade de copiar imagens, já que o SF é o mesmo, em todos os nós; b) em caso de falha de um nó, (a) aplica-se; finalmente, c) uma VM em execução pode ser deslocada de um nó para outro sem interrupção de serviço "bastando" para isso movimentar as suas páginas de memória, pois não há necessidade de copiar a imagem.

Os hipervisores[4] diferem também nos SF que suportam: em geral, todos suportam o NFS como SF partilhado, mas em termos de SF locais suportam os que são tradicionais no SO; assim, o KVM e XenServer suportam vários SF Linux (Ext3, Ext4, XFS, BTRFS), o Hyper-V suporta o NTFS, e o ESXi suporta um SF próprio, partilhado, o VMFS.

Uma desvantagem dos SF genéricos, como o Ext3, Ext4, XFS, e NTFS, é que foram concebidos para suportar, com boa eficiência, requisitos muito díspares: tanto elevado, como reduzidos números de ficheiros, de dimensões que vão dos poucos bytes às dezenas de GB; tanto diretorias com poucos como por vezes (caso do XFS e do BTRFS) muitíssimos ficheiros, e ainda de poucos a muitos níveis na árvore do SF. Esse mesmo grau de generalidade acaba por se traduzir num SF pouco eficiente no caso do suporte de VMs, que são estruturas de informação com um reduzidíssimo número de ficheiros (podem ser apenas dois - a definição da VM e a sua imagem) sendo que as imagens são muito grandes - dezenas de GB.

[4] Daqui em diante referimos-nos-emos quase sempre ao Tipo 1, pelo que omitiremos essa informação nesses casos

2.3.2 Sistemas de Armazenamento de Objectos

Para ultrapassar as desvantagens dos SF, os hipervisores têm vindo a utilizar Sistema de Armazenamento em Objetos, caracterizados por um espaço de nomes plano em que cada objeto é referenciado por um identificador único, do tipo dos conhecidos UUIDs (*Universal Unique Identifier*), identificadores de 128 bits normalmente apresentados como *strings* hexadecimais.

Este modelo apresenta o armazenamento de dados em forma de objetos, entidades contendo dados não estruturados que são posteriormente agrupados em espaços de armazenamento denominados de *pools* ou *buckets*. As *pools* são bastante flexíveis em termos de espaço de armazenamento podendo ser facilmente expandidas ou contraídas consoante as necessidades.

As implementações, das quais daremos conta de algumas na próxima secção, baseiam-se em múltiplos nós (servidores) de armazenamento, contendo tipicamente dezenas de discos cada um; sobre estes nós são distribuídas as *pools*, sendo que um objeto é segmentado em porções distribuídas por múltiplos nós, podendo estas porções ser, ou não, replicadas como forma de garantir tolerância a faltas e elevado desempenho por via do paralelismo nos acessos.

Os objetos são compostos pelo identificador, meta-dados, e dados (do utilizador) propriamente ditos. Os meta-dados têm um papel relevante fornecendo uma descrição objeto guardado, sendo esta de grande flexibilidade, permitindo uma melhor gestão de todo o sistema. A utilização de objetos para armazenamento de dados oferece uma grande versatilidade em termos de dimensões dos mesmos, otimizando a criação de grandes volumes de dados assim como alteração da sua dimensão.

Por estes motivos é possível admitir que uma *Object Storage* é uma opção muito boa para suportar o armazenamento de VMs, tendo em conta as exigências do projeto iCBD. Existem várias soluções desenvolvidas as quais podemos salientar o OpenStack Swift [24], Ceph [25] e ScaleIO [26], sendo os dois primeiros projetos *open source* e o último desenvolvido pela Dell/EMC e código fechado, mas de distribuição gratuita.

2.3.2.1 Ceph

O Ceph é um sistema de armazenamento de objetos baseado em software aberto que corre em servidores comuns (por oposição a outros que usam hardware especializado o que, na língua inglesa, se vem designando por *software-defined storage*. A entidade fundamental é o *storage cluster*, agregado de nós, altamente escalável, com mecanismos de auto-gestão e recuperação de faltas.

O facto de ser baseado em software evita, como já se disse, ter de se utilizar hardware especializado, geralmente muito caro; em seu lugar, a infraestrutura do sistema pode ser composta de hardware comum, disponível no mercado, e por esta razão tanto o investimento inicial para suportar um sistema destes como o custo total de manutenção do sistema podem ser reduzidos. Isto também se aplica ao ato de aumentar a capacidade do sistema, empregando o principio do *scale-out*: simples adição de mais recursos - nós - à infraestrutura existente. Outros sistemas existem que crescem aumentando a capacidade dos recursos já existentes (o que se designa por *scale-up*), o que é uma solução menos flexível pois com alguma facilidade se pode atingir um limite em que a capacidade não pode mais ser aumentada.

É relevante destacar que o Ceph possuí mecanismos que permitem que o sistema seja gerido automaticamente, detetando faltas de nós e removendo do sistema os nós danificados, e adicionando novos nós aquando da instanciação dos mesmos.

Este sistema *object storage*, designado RADOS (*Reliable*, *Autonomous*, *Distributed Object Store*), possui três interfaces diferentes para que os seus clientes interajam com os dados armazenados: RADOS *Object Gateway*, que permite o acesso aos dados a nível de objetos; o RADOS*Block Device*, ou RBD, permite expôr os objetos como volumes de armazenamento; e o Ceph *File System* que implementa um sistema de ficheiros e diretorias. Tendo em conta o contexto deste projeto, apenas o RBD vai ser abordado; no entanto, a variedade de interfaces disponíveis proporciona uma grande diversidade de casos de uso para este sistema de armazenamento.

Arquitetura

A arquitetura do Ceph, representada na figura 2.3, é composta por diferentes peças de software que proporcionam diferentes funcionalidades ao sistema. A base de toda esta arquitetura é o RADOS que organiza o sistema de armazenamento sob a forma de um *cluster*.

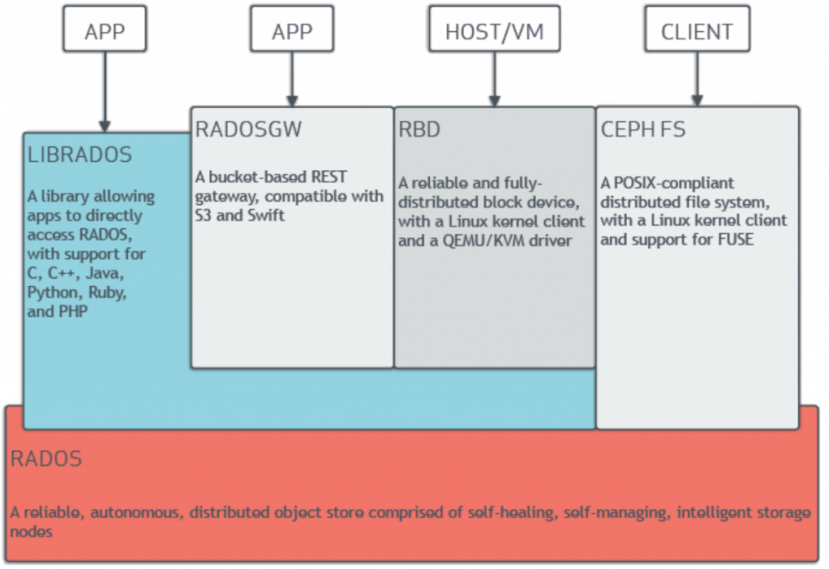


Figura 2.3: Arquitetura do sistema Ceph[27]

Com o intuído de entender o funcionamento do RADOS é essencial identificar os dois tipos de membros do *cluster* imprescindiveis ao seu funcionamento: OSDs (*Object Storage Daemons*) e *Monitors*.

Monitor Um monitor atua como um gestor do estado do conjunto de membros do *cluster*, usando o conhecido protocolo de consenso denominado Paxos [28]. Dado que um grupo de monitores tem que chegar a um consenso, é desejável dispôr de um número pequeno e ímpar destes, a fim de tornar o processo de consenso mais coerente e rápido. É usual, em ambientes Ceph de produção com alguma dimensão, que existam nós que apenas correm os serviços monitores.

Object Storage Daemon Um OSD é um componente de software que é executado num nó de armazenamento, e que usualmente tem a seu cargo um único "disco", possibilitando a sua utilização pelo *cluster* RADOS; como exemplo, um nó de armazenamento com 10 discos de dados (e talvez mais 2 de sistema) poderia ter 10 OSDs em execução, num mapeamento 1:1. Basicamente um OSD consegue mapear um local de armazenamento para o *cluster*. Para obter uma distribuição dos dados pelo *cluster* os OSDs utilizam o algoritmo CRUSH [29] que calcula (utilizando um mapa e um conjunto de regras fornecido pelos monitores) os locais onde devem ser guardados os blocos de dados. Este algoritmo consegue também dar ao cliente o potencial de calcular a localização dos seus dados tornando dispensável um sistema de *look-up*.

Os OSDs são essencialmente responsáveis pela transferência de dados de/para os clientes, mas também colaboram entre si com o propósito de cumprir tarefas de recuperação de falhas (como o re-balanceamento dos dados).

As versões mais recentes do Ceph requerem um outro membro, denominado *Manager*, que corre em conjunto com um *monitor* afim de proporcionar monitorizações adicionais e interfaces para outros sistemas de gestão e administração do *cluster*.

Pode existir também um outro tipo de membro, designado por *Meta-Data Server*, o qual permite oferecer um sistema de ficheiros (*Ceph File System*). Como esta funcionalidade não é necessária no nosso trabalho, não vamos abordar o funcionamento deste componente.

Para aceder ao *cluster* RADOS foi desenvolvida uma API que foi implementada tanto sob a forma de uma biblioteca de nível utilizador, a libRADOS, como sob a forma de um *kernel module* rbd.ko que os clientes podem usar para expôr objetos RADOS como *disk devices*.

O RADOS *Block Device* (RBD) é a solução do Ceph para o armazenamento de dados em blocos. O RBD possibilita a criação e alteração de Ceph *block devices* cujo conteúdo se encontra distribuído por todo o *cluster*; dispõe ainda de funcionalidades de *thin-provisioning*[5] e *snapshotting*. Estes volumes de dados podem ser ligados a hipervisores para que estes possam utilizá-los com a finalidade de oferecer discos a máquinas virtuais. Como o sistema de armazenamento está totalmente dissociado da máquina física em que corre o hipervisor, o RBD facilita a rápida e fácil migração de VMs, tendo apenas que instanciar o volume de dados no hipervisor da máquina física pretendida. Esta dissociação e distribuição dos dispositivos de armazenamento promove um sistema totalmente confiável e robusto para o armazenamento de imagens de VMs que, aliado aos mecanismos de *thin-provisioning* e *snapshotting*, são essenciais para suportar uma infraestrutura de virtualização flexível.

CRUSH

Os OSDs e os clientes Ceph usam o algoritmo CRUSH para calcular eficientemente a localização dos dados. O CRUSH proporciona uma ótima gestão de dados permitindo distribuir os objectos de uma forma uniforme pelos vários nós do *cluster*, além disso este algoritmo garante a replicação dos dados aumentando a confiabilidade dos mesmos.

Os clientes recorrem ao *cluster map*[6] (obtido a partir de um monitor) para saber a topologia do *cluster* e conseguir calcular os locais onde estão armazenados os dados.

Existem três parâmetros que influenciam a maneira como o Ceph realiza a distribuição dos dados pelo *cluster*:

[5] O termo *thin provisioning* é usado para indicar que é possível definir um volume com uma dada dimensão, e.g., 100GB, mas não pré-alocar os blocos, fazendo-o apenas à medida que forem necessários

[6] O *Cluster Map* é composto por outros cinco mapas (*Monitor*, OSD, *Placement Groups*, CRUSH e MDS) que fornecem a informação sobre a topologia do *cluster*

- **Número de Réplicas:** este parâmetro determina o número de cópias de cada objecto no *cluster*. São necessárias pelo menos duas réplicas para garantir uma segurança mínima dos dados, para ter uma boa disponibilidade nos dados são recomendadas pelo menos três réplicas.
- **Número de Placement Groups:** as *pools* do Ceph são divididas em vários *Placement Groups* (PGs). Quando um objecto vai ser guardado no *cluster*, este é mapeado dentro de um PG, um PG é para vários OSDs. Os objectos de um PG são replicados por todos os OSD pertencentes ao PG. Este valor é definido aquando da criação de uma *pool* e deve ser ajustado dependendo do tamanho do *cluster*.
- **Regras CRUSH:** estas regras definem politicas de distribuição dos dados no *cluster* permitindo aos administradores especificar o exatamente como é que os dados são guardados no *cluster*.

Para calcular o local de um objecto, um cliente apenas necessita de fornecer o nome e a *pool* do objecto. A figura 2.4 esquematiza o processo que se divide em duas partes:

1. Calculo do PG a que o objeto pertence
   a) Em primeiro lugar é realizado o hash do objecto.
   b) Calcula-se o modulo do hash pelo número de PGs para saber o id do PG.
   c) O id da *pool* é verificado tendo em conta o nome.
   d) O id da *pool* é juntado ao id do PG para formar o PG do objecto
2. Executa-se o algoritmo CRUSH fornecendo o PG, o estado do *cluster* e o conjunto de regras. Isto devolve os OSDs que contêm o objecto.

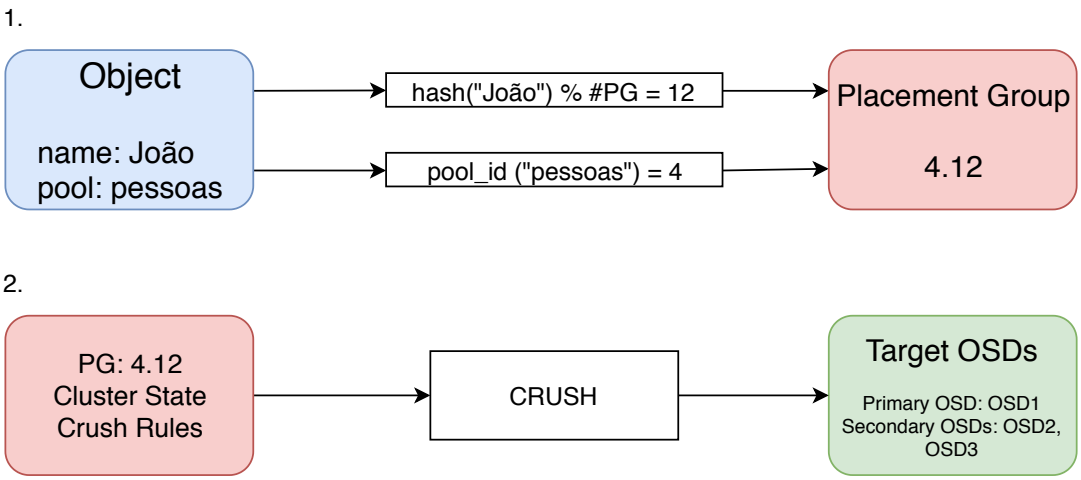


Figura 2.4: Processo do calculo do local dos dados num *cluster* Ceph

Os clientes realizam este processo para poderem aceder diretamente ao local onde os dados estão guardados. Quando é realizada uma leitura, o cliente pode ler de qualquer

OSD que contenha uma cópia do objecto (i.e. qualquer OSD que pertença ao PG do objecto), permitindo ao *cluster* distribuir a carga por diferentes nós.

As escritas são realizadas apenas no OSD primário do PG. Os OSDs primários lidam com todas as escritas e replicam as alterações pelos OSDs secudários pertencentes ao mesmo PG. O cliente recebe a resposta da escrita depois de ser realizada a replicação por todos os OSDs.

Snapshots

O RADOS incorpora um mecanismo de *snapshotting* dividido em dois tipos:

- **Pool Snapshots:** este tipo de *snapshots* são aplicados a todos os objetos numa *pool*.
- **Self Managed Snapshots:** neste caso é fornecido um contexto de *snapshots* (*SnapContext*[7]) a cada escrita.

Para cada objecto num PG, em termos de estruturas guardadas em disco, este tem associado um *head object*[8] que corresponde à versão *writable* do objeto.

Para além dos *head-objects*, cada objecto pode ter vários clones associados. Sempre que existe uma escrita num objecto com novo snapshot, antes da alteração dos dados, é criado um clone para salvaguardar o estado do objeto referente ao novo snapshot.

Esta funcionalidade é estendida para o RBD permitindo criar um *snapshot* de uma imagem RBD que corresponde a um cópia read-only da imagem. Para além da utilidade na criação de *backups*, os *snapshots* permitem a criação de *linked-clones*.

O RBD inclui também a possibilidade de usar *incremental backups* que permite a criação de *snapshots* contendo apenas o conjunto de dados que foi alterado em relação ao ultimo *snapshot*. Esta funcionalidade beneficia especialmente a criação de *backups* quando se implementa uma estratégia de *disaster recovery* reduzindo significativamente a ocupação do espaço de armazenamento assim como a largura de banda necessária para manter os *backups* atualizados.

[7] O *SnapContext* é o conjunto de snapshots definido para um objecto e o número de sequência do *snapshot* mais recente

[8] Este objeto contém também os conjuntos de snapshots definidos e os clones existentes, entre outras informações para conseguir saber o espaço que o objecto ocupa

Thin-provisioning

RBD *Layering* é o nome dado ao mecanismo de *thin-provisioning* disponível no RBD, este permite criar clones *copy-on-write* (COW) a partir de um snapshot. Um clone COW, após a sua criação, é composto apenas por uma referência para o snapshot que o originou e não ocupa espaço de armazenamento extra, o clone só começa a ter peso no espaço de armazenamento depois de existirem escritas realizadas no clone. Este mecanismo permite suportar eficientemente estratégias de virtualização que usem de *golden-images* permitindo a clonagem de várias VMs instantaneamente e sem ocupar espaço adicional com duplicação de dados.

O processo de criação de um clone no RBD pode ser apresentado na figura 2.5. Em primeiro lugar cria-se um snapshot da imagem que se pretende clonar, de seguida protege-se o snapshot para bloquear qualquer tentativa de remoção, finalmente realiza-se o clone do snapshot pretendido.

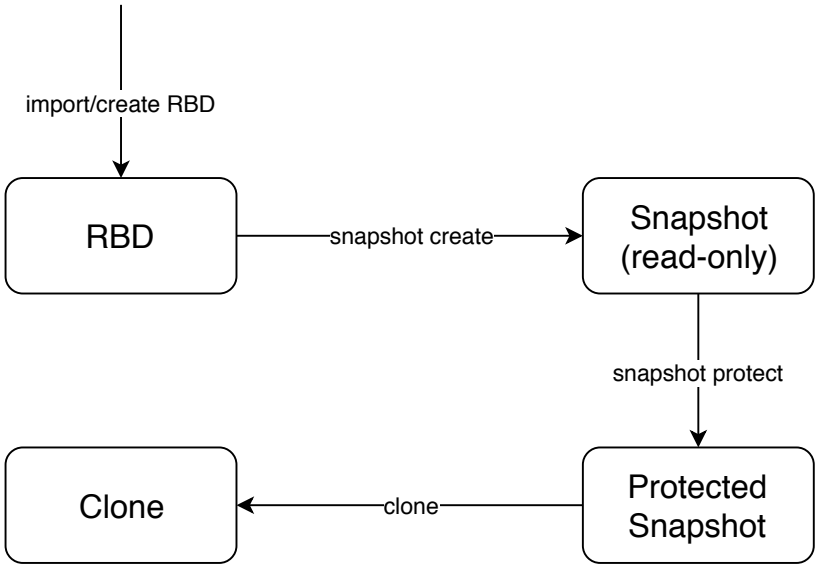


Figura 2.5: Processo de clonagem de um RBD

2.3.3 *Mirroring*

O *mirroring* é o mecanismo que permite a replicação de imagens RBD entre dois *clusters*. Neste momento a implementação do Ceph apenas permite realizar *mirroring* entre dois *cluster* realizando: *one-way replication* (os dados são replicados de um *cluster* primário para um secundário) ou *two-way replication* (ambos os *cluster* são primários e secundários ao mesmo tempo e replicam os dados entre os dois). O *mirroring* pode ser configurado para replicar apenas uma imagem ou todas as imagens contidas numa *pool*.

As imagens guardadas no *cluster* primário (designadas de imagens primárias) são replicadas no *cluster* secudário (imagens secundárias) com o auxilio de um *daemon* que se responsabiliza por uma instância de replicação *one-way*. Se o objectivo for realizar

two-way replication é necessário utilizar um *daemon* adicional para realizar a replicação no outro sentido.

No capítulo 4 esta mecanismo vai ser abordado como possível solução para realizar a replicação entre vários *clusters*

2.4 Replicação

Um sistema distribuído designa-se como um conjunto de diversos computadores (normalmente denominados nós) que estão conectados através de uma rede e coordenam-se através da troca de mensagens. Os sistemas distribuídos apresentam várias vantagens, em primeiro lugar, possibilitam a partilha de recursos, e.g. uma empresa que tenha comércio online necessita que os clientes tenham acesso ás informações produtos disponíveis como o preço e o stock, através da partilha de recurso é possível que os vários utilizadores tenham acesso á mesma informação e a possam alterar de forma coerente.

Em segundo lugar os sistemas distribuídos concedem um nível de desempenho superior aos sistemas centralizados, o aumento do número de utilizadores pode facilmente sobrecarregar um sistema centralizado com uma quantidade excessiva de pedidos, no entanto, num sistema distribuído estes pedidos podem ser distribuídos por diferentes nós permitindo equilibrar a carga de processamento por vários pontos. Além disto, estes sistemas oferecem uma grande escalabilidade tornando possível aumentar a capacidade do sistema através da adição de um novo nó (*scale-out*) que apresenta uma solução mais eficaz e económica para lidar com a sobrecarga dos sistemas.

Independentemente dos benefícios, a construção de sistemas distribuídos acarreta outros obstáculos que necessitam têm de ser tomados em conta quando se desenha um sistema distribuído. A tolerância a faltas é um fator importante no que toca a estes sistemas sendo que a falta de uma (ou várias) máquinas não pode impossibilitar o correto funcionamento do sistema, assim torna-se imprescindível o uso de mecanismos que permitam lidar com estes problemas.

Um sistema de replicação pode dividir-se em dois tipos:

Replicação Ativa Na replicação ativa os pedidos dos clientes são ordenados através de um protocolo e executados em todas as máquinas que constituem o sistema, é importante referir que para obter consistência nos dados neste tipo de replicação as operações realizadas nos servidores têm que ser deterministas, i.e. a operação produz o mesmo resultado para os mesmos inputs. A replicação ativa, apesar de mais complexa, representa uma solução mais adequada para quando existe uma necessidade obter respostas rápidas por parte do sistema, mesmo na presença de grandes cargas ou faltas.

Replicação Passiva A replicação passiva impõe que as operações são realizadas apenas num único servidor (designado de primário) e posteriormente propagadas para

todos os outros. Neste tipo de replicação é possível criar sistemas não deterministas, no entanto os tempos de resposta podem ser bem maiores que os do tipo anterior, especialmente na presença de uma falha o que leva a re-execução da operação. No caso de existir uma falta no servidor primário, um dos servidores secundários assume o papel de primário.

Devido ás múltiplas cópias do mesmo conjunto de dados é possível assim obter uma redundância que permite eliminar consequências como a perda de dados, com a adição de um mecanismo de deteção e recuperação de faltas é possível tornar o sistema tolerante a faltas.

A replicação tem a capacidade de aumentar o desempenho do sistema, o facto de existirem várias cópias dos mesmos dados em diferentes locais permite distribuir a carga dos acessos por diferentes nós evitando assim sobrecargas num único nó.

Uma grande desvantagem da replicação reside no facto da duplicação. A duplicação implica as alterações efetuadas num conjunto de dados sejam re-executadas em todos os nós do sistema, dependendo do número de réplicas, as alterações podem ser mais extensas e ocupar capacidade de processamento. Além disto o facto de existir duplicação provoca sobrecarga do espaço de armazenamento dependendo da quantidade de dados a serem replicados e do número de replicas criadas.

Capítulo 3

iCBD

A iCBD (*Infrastructure for Client-Based Desktop*) é um projeto de investigação e desenvolvimento, realizado por uma equipa mista da SolidNetworks (do grupo Reditus, S.A.) e do Centro de Investigação NOVA LINCS da FCT/NOVA, este visa a implementação de uma *Client-Based* VDI que tira partido dos recursos disponíveis nos dispositivos dos utilizadores para executar ambientes de trabalho virtualizados de uma forma não intrusiva, assim, o peso de processamento é retirado da infraestrutura e distribuído pelos diversos postos de trabalho que executam as instâncias localmente, no entanto, o armazenamento da instância é dissociado da máquina do utilizador e encontra-se no sistema de armazenamento da infraestrutura.

A solução dada pela iCBD contrasta com as atuais implementações de VDIs (*server-based*) disponibilizadas pelas grandes empresas como VMWare e Citrix que requerem infraestruturas com uma grande capacidade de processamento para hospedar todas as VMs dos utilizadores. A iCBD permite reduzir significativamente a carga da infraestrutura resultando numa solução muito mais acessível sem comprometer o desempenho do serviço, esta contém o seu próprio sistema de administração dotado de uma grande simplicidade pois só existe uma única entidade a administrar: o *template*. Dada a complexidade da iCBD a secção 3.1 é apresentada a arquitetura da solução e posteriormente na secção 3.2 será explicado o funcionamento da infraestrutura através da descrição do processo de *boot* de um posto de trabalho. Na ultima secção é explicado o processo de administração de *templates*.

3.1 Arquitetura

Daqui em diante, vamos adotar o conceito de iMI (iCBD Machine Image) retirado de [11] para denotar o conjunto de ficheiros necessários para executar uma instância na iCBD,

este pode ser dividido em três grupos:

- ***Template***: engloba todos os ficheiros referentes a um *template* de uma VM como os ficheiros de configuração, discos (que contêm um SO), etc.
- **Pacote de *boot***: este conjunto de ficheiros é utilizado para realizar o *boot* das instâncias, este conjunto inclui: uma ferramente *BusyBox*, um ficheiro init e dois *Run Control Scripts* (rc0 e rc1) para iniciar os serviços de rede, montar todos os SF e levar o sistema para modo *single-user*.
- **Configurações do Serviço**: alguns dos serviços usados na iCBD necessitam de ficheiros de configuração específicos, como por exemplo, o iSCSI necessita de configurar *targets* para possibilitar o acesso aos *block-devices* que contêm a imagem *template*, por outro lado, o Ceph necessita de alguns ficheiros de configuração que permitam a conexão com o cluster e o acesso aos dados.

Posto isto, a iCBD contém duas peças na sua arquitetura que são essenciais e permitem suportar a execução de iMIs nos postos de trabalho dos utilizadores: suporte para *boot* remoto e o sistema de armazenamento de iMIs.

Suporte de *Boots* Remotos

Relativamente ao *boot* remoto, este é suportado por tecnologias que já têm provas dadas há muitos anos, das quais podemos destacar:

- ***Preboot Execution Enviroment***: O PXE permite a criação de um ambiente de *boot* para inicialização dos postos de trabalho através de uma placa de rede.
- ***Dynamic Host Configuration Protocol***: O DHCP é o primeiro serviço utilizado pelo PXE, este é responsável pela atribuição de IPs assim como listar a localização na rede dos ficheiros de *boot*.
- ***Trivial File Transfer Protocol***: O TFTP é um protocolo para a transferência de ficheiros, este é utilizado pelo PXE para transferir o ficheiros iniciais para realizar o *boot* de uma máquina.
- ***Hypertext Transfer Protocol***: O HTTP é utilizado para transferir alguns *scripts* relativos ao iCBD durante o processo de *boot*, além disto o HTTP também tem a possibilidade de suportar a transferência dos ficheiros iniciais para o *boot* tal como o TFTP.

Os protocolos descritos anteriormente são cruciais para o processo de *boot* das máquinas dos utilizadores e são ponto de partida na execução das instâncias iCBD.

As primeiras fases do *boot* de um posto de trabalho, é apresentado ao utilizador um menu (através do ambiente de execução PXE) que contem todas as iMIs registadas no

sistema, esta fase, é a única que requer a interação do utilizador com a máquina durante todo o processo de *boot*.

A iCBD possibilita realizar o *boot* remoto de iMIs de duas formas possibilitando a execução nativa ou virtualizada de instâncias. A primeira opção (apenas suportada em SOs Linux) permite a execução da instância diretamente no hardware da máquina. A segunda opção utiliza um sistema Linux com um hipervisor embutido para criar uma VM que vai suportar a instância.

Sistema de Armazenamento

O sistema de armazenamento da iCBD guarda toda a informação relacionada com o serviço, este divide-se em dois caminhos que usam tecnologias distintas, um deles baseia-se na utilização do sistema de ficheiros btrfs e originou dois estudos: utilização de *linked-clones* baseados em funcionalidades de *snapshots* do btrfs[4] e a criação de um sistema de replicação e caching para suportar as VMs armazenadas no btrfs[11]. Por outro lado, a iCBD suporta a utilização de sistemas de armazenamento de objetos (OBS), esta abordagem usa como solução o Ceph que proporcionou uma nova linha de investigação, na qual este trabalho se insere, e já fez surgir um estudo relativo à utilização de OBS para criação de linked clones utilizados pelas VMs [9] que serviu como base fundamental para este trabalho.

O btrfs é a opção que serviu de base para a implementação da iCBD. Esta foi escolhida pelas suas funcionalidades como: a criação de sub-volumes para guardar as informações do serviço; *snapshots* para criar um controlo de versões dos *templates* e gerar *backups*; *seeding* que permite montar múltiplas vezes (em modo *read-only*) o mesmo sistema de ficheiros btrfs, isto possibilita a utilização do mesmo *template* por vários clientes [11].

Os sistemas de armazenamento de objetos fizeram surgir um novo caminho na implementação da iCBD permitindo dotar o sistema de armazenamento da infraestrutura com uma grande escalabilidade e confiabilidade. O Ceph apresentou-se como a solução mais viável porque, para além de ter como base um software aberto, apresenta um leque de funcionalidades que permitem suportar a iCBD eficientemente. Tal como explicado em 2.3.2.1 o sistema RBD facilita a criação de vários *block-devices*, que podem ser mapeados (em máquinas Linux) ou ligados a VMs, estes, em combinação com os mecanismos de *snapshots* e *layering* possibilita a utilização de um RBD para suportar o armazenamento de iMIs. O mecanismo de *snapshots* facilita o controlo de versões e a utilização do *layering* viabiliza o suporte de fornecimento de imagens para as instâncias permitindo criar vários *thin-clones* a partir de um *template* para suportar múltiplos clientes.

Ambos os sistemas de armazenamento utilizam protocolos que permitem interagir com os sistemas, no caso do btrfs este é suportado pelo NFS e iSCSI. No caso do Ceph como referido anteriormente este usa o RBD para interagir com o sistema através de *block-devices*.

Um aspeto importante que se enquadra no tópico do armazenamento é suporte para o espaço de escrita da instância do utilizador, dado que as imagens *template* são do tipo

read-only (ro), é necessário criar um volume de armazenamento *read-write* (rw) para que o utilizador consiga interagir com o sistema, assim a iCBD contém um conjunto de *scripts* que realizam a criação de volumes no sistema de armazenamento (btrfs ou Ceph) durante o processo de *boot* de um posto de trabalho. No caso do Ceph é ainda possível tirar partido do mecanismo de *layering* presente no RBD para criar *thin-clones* (do tipo *read-write*) que substitui a utilização de dois volumes (ro + rw) por um único volume (rw).

Sistema de Administração

O sistema de administração desenvolvido para a iCBD destina-se a controlar o ciclo de vida das iMIs. Este tem como base um conjunto de *scripts* que permite a um administrador adicionar, remover ou alterar iMIs.

Independentemente do tipo de execução da iMI (nativa ou virtual), o processo de administração é sempre realizado da mesma maneira, que de uma forma geral (explicado com mais detalhe em 3.3), passa pela alteração do *template* do iMI e termina na geração do pacote de *boot* para o iMI.

3.2 Arranque dos Postos de Trabalho

Quando um utilizador liga uma máquina (configurada para usar a iCBD), esta envia um pedido *broadcast* para a rede com a indicação que pretende realizar um *boot* PXE. O servidor DHCP interceta o pedido e responde com as informações de rede necessárias (como IPs, *gateways*, DNS) assim como a localização do servidor TFTP. Com isto, é possível transferir um *bootloader* (pxelinux) que permite exibir no ecrã uma interface gráfica que apresenta as várias iMIs disponíveis no sistema.

Após a seleção do utilizador, o *bootloader* inicia a transferência (TFTP ou HTTP), de um núcleo (*kernel*) Linux em conjunto com o ficheiro initramfs[1] (*initial RAM file system*) que são posteriormente carregados em memória. De seguida, o *kernel* é executado, que depois de várias verificações ao hardware, inicia o primeiro processo do sistema (init).

O processo init realiza um série de configurações no *hardware*, em particular, a configuração das placas de rede, para posteriormente permitir a transferência de outros ficheiros fundamentais para o *boot*: dois *scripts* de controlo de execução modificados (rc0 e rc1) assim como ficheiros necessários para realizar as ligações com os sistemas de armazenamento (e.g. chaves de autenticação para clientes Ceph). De seguida, é executado o *script* rc0 que prepara o sistema para arrancar a imagem, mais especificamente, este *script* permite montar os volumes de armazenamento que contêm os dados da imagem assim como os espaços *read-write* para suportar a instância. Este *script* termina com o *switchroot* que é o último passo na inicialização do sistema.

No caso do sistema iniciado ser o suporte para executar uma instância virtual, o *script* rc1 é executado. Este é responsável por preparar o ambiente para correr a VM que vai

[1] O initramfs (ou initrd noutros sistemas) é o ficheiro que contém o SF inicial do sistema. O *bootloader* carrega o initramfs em RAM que é montado pelo *kernel* para realizar o resto do processo de *boot*.

executar a imagem final, que começa com a execução do hipervisor seguida da criação e configuração da VM e termina com a inicialização da VM que vai realizar um *boot* normal.

3.3 Administração de iMIs

Dado que o sistema de administração controla o ciclo de vida dos iMIs, em primeiro lugar temos de definir as diferentes fases que constituem este ciclo.

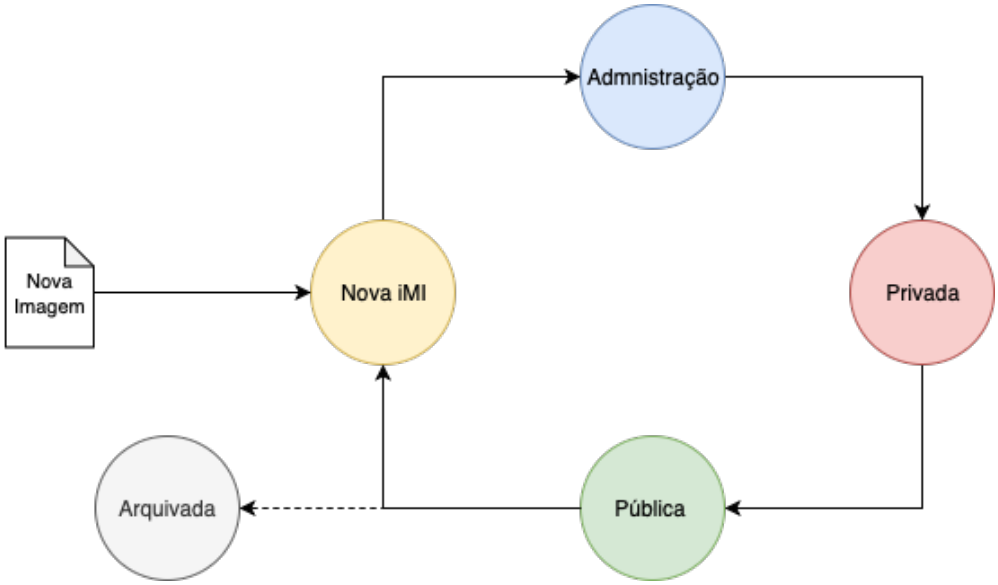


Figura 3.1: Ciclo de vida das iMIs

Tal como ilustrado na figura 3.1 a primeira fase de uma iMI passa pela adição da nova imagem à iCBD. Dado que a nova iMI ainda não contém um pacote de *boot* associado, é necessário que esta passe pelo primeiro processo de administração que permite gerar o pacote de *boot* para a nova imagem.

A fase de administração começa com a execução da imagem para que esta possa alterada (no caso de *updates*) e termina com a criação do pacote de *boot*. Durante esta fase o administrador pode realizar actualizações ou adicionar novo software à imagem conforme o necessário.

Terminada a fase de administração, a iMI entra num estado privado, invisível para todos os utilizadores. Neste ponto, devem ser realizados os testes necessários antes de disponibilizar a iMI para todos os utilizadores. Após a iMI ter sido testada, esta pode passar para o estado público que torna a iMI visível no menu de *boot* dos postos de trabalho.

Por fim, quando é publicada uma nova versão de uma iMI, a versão antiga (se existir) é marcada para remoção antes de ser arquivada. Dado que os utilizadores podem ter sessões iniciadas com a iMI em causa, é necessário esperar que todas as sessões sejam terminadas para retirar a iMI do sistema.

As versões das iMIs são representadas no sistema através de um índice numérico, sendo que o maior índice corresponde à versão mais recente. Sempre que é iniciado o processo de administração é criado um *linked-clone* (temporário) a partir da versão mais recente que serve como uma imagem temporária permitindo realizar alterações sem comprometer as sessões dos clientes associadas a essa iMI.

Posto isto, o processo de administração pode ser inicializado de duas formas: nos postos de trabalho, ou através do terminal no servidor de imagens. De qualquer das formas, este processo tem como base um *script* (adm) que quando executado sobre uma iMI, inicia uma VM para suportar o processo de administração, esta usa um sistema Linux para poder tirar partido das várias funcionalidades de snapshoting (Ceph e btrfs) permitindo interagir directamente com o sistema de armazenamento e criar os *linked-clones* assim que a VM é iniciada. Posteriormente é executado um hipervisor (*VMware Player* ou KVM) que inicia um outra VM configurada para executar o clone recém-criado, com isto, o administrador consegue realizar as alterações necessárias na imagem.

Depois do administrador ter terminado as alterações no *template* e desligado o sistema, este pode escolher descartar ou instalar as alterações. A instalação refere-se ao processo de publicação de uma iMI que começa com a criação do pacote de *boot* para a nova versão do *template* da iMI (através do *script* mki), e termina com a criação dos volumes nas directorias correctas, com isto o processo de administração termina e a nova iMI passa a estar disponível para todos os clientes iCBD.

Capítulo 4

Implementação

Neste capitulo é apresentada a implementação dos serviços de replicação que suportam o armazenamento consistente em múltiplos *clusters* de *golden-images* (ou *templates*) assim como a criação dos respetivos clones. No contexto do trabalho, o sistema de replicação interage com o sistema de armazenamento de objetos (nomeadamente o *Ceph*) permitindo a adição e alteração de imagens RBD (RADOS Block Devices) que são posteriormente replicadas noutros *clusters* associados a diferentes nós do sistema de replicação.

Como explicado anteriormente, as imagens *template* usadas pelos clientes iCBD são do tipo *read-only*; a partir destas são geradas iMIs, imagens *bootable*, que quando instanciadas num cliente são clonadas e transformadas em *read-write*. Desta forma, o foco principal desta implementação passa pela gestão de um conjunto de *templates* garantindo disponibilidade e confiabilidade através da replicação tornando a iCBD uma VDI escalável e tolerante a faltas.

4.1 Arquitetura

O sistema de replicação tem como base fundamental a utilização da API RBD (RADOS Block Devices) para interagir com as várias imagens *template* armazenadas nos *clusters* Ceph.

Para realizar a replicação total de uma imagem, o módulo de replicação tira partido das funções *export* e *import* dessa API. Quando um administrador (de imagens) de um *site* iCBD com suporte à replicação pretende actualizar uma imagem, fá-lo executando os comandos administração de imagens no nó de administração (ou primário). Terminada a actualização da imagem, a replicação desta é então realizada com recurso a *snapshots* incrementais obtidos usando as funções Ceph *export-diff* e *import-diff* que permitem gerar apenas um delta que corresponde ao conjunto de dados alterados aquando da actualização da imagem, reduzindo drasticamente a quantidade de dados transmitidos do primário para as réplicas. Para que isto seja possível, um nó de replicação tem de conter também o software Ceph para realizar operações no *cluster* em causa.

É importante recordar aqui que num *cluster* Ceph há diversos tipos de nós (no sentido mais abstrato do termo), e.g., monitor, OSDs, etc., sendo que num nó podem coexistir alguns desses tipos - embora tal não seja aconselhado. Para manter essa separação de papéis, os nós de que falaremos daqui em diante são apenas de um único tipo (e.g., de replicação, monitor, OSD, etc.). Também, recordemos, para que um sistema (computador) utilize o armazenamento de objetos que lhe é oferecido pelo Ceph, tem de ter instalado software (uma biblioteca ou um módulo *kernel*) que implemente o protocolo RADOS (além de ficheiros de configuração, chaves de autenticação, etc.) para comunicar com o *cluster* Ceph. Um tal sistema designa-se por cliente Ceph e, se bem que um nó servidor do *cluster* Ceph possa também ter o módulo cliente instalado, tal não é de todo recomendado.

Assim, os nós do sistema de replicação são "simples" clientes Ceph, e não nós do *cluster*. A interação de um administrador com o sistema de replicação é realizada num nó do sistema de replicação através de um conjunto de *scripts* que fornecem os diversos comandos. A figura 4.1 apresenta os dois componentes essenciais de um nó do sistema de replicação:

- **Cliente Ceph**: módulo *kernel* utilizado para interagir com um (ou vários) *clusters* Ceph. O cliente usa a API RADOS para gerir *block devices* assim como criar *snapshots* ou clones nos *clusters* Ceph.

- **Módulo de Replicação**: este foi o componente implementado nesta dissertação, e coordena os processos de replicação utilizando serviços Ceph disponibilizados pelo módulo *ceph-client*. Além disso, este módulo é responsável pelas tarefas de deteção e recuperação de faltas.

Assim, permite-se a cada instância iCBD num *site* possa ter um *cluster* Ceph dedicado para o armazenamento persistente das imagens, e interagir (por via da replicação)

diretamente com *clusters* de outras instâncias. O estado do sistema é traduzido em dois conjuntos, correspondentes aos *templates* disponíveis e aos nós que constituem o sistema. O primeiro conjunto é alterado como resultado de tarefas de administração da iCBD, por exemplo, sempre que é adicionado, removido ou realizada uma atualização de um *template*. O segundo corresponde ao grupo de membros do sistema, e é alterado sempre que é adicionado ou removido um nó (instância) de replicação.

O sistema de replicação é composto por dois tipos de servidores: de administração (ou primário) e réplicas. Estes servidores estão representados na figura 4.1, que exibe algumas das interações realizadas pelas diferentes instâncias.

Servidor Primário Esta instância é responsável pela gestão de todo o sistema. Este servidor regista a adição ou remoção de nós do sistema assim como o processo de *bootstrapping*[1] de novos nós. A administração dos *templates* também é efetuada pela instância primária, sendo que esta é a única que contém todos os *templates* disponíveis no sistema. Por fim, as tarefas de replicação são executadas por este servidor.

Servidor Réplica Esta instância armazena os *templates* que vão ser utilizados na criação de clones que suportam as VM instanciadas nos clientes do *site* iCBD ligado à réplica. Desta forma, o servidor-réplica é capaz de transferir qualquer *template* presente no servidor primário e ir recebendo as atualizações posteriores.

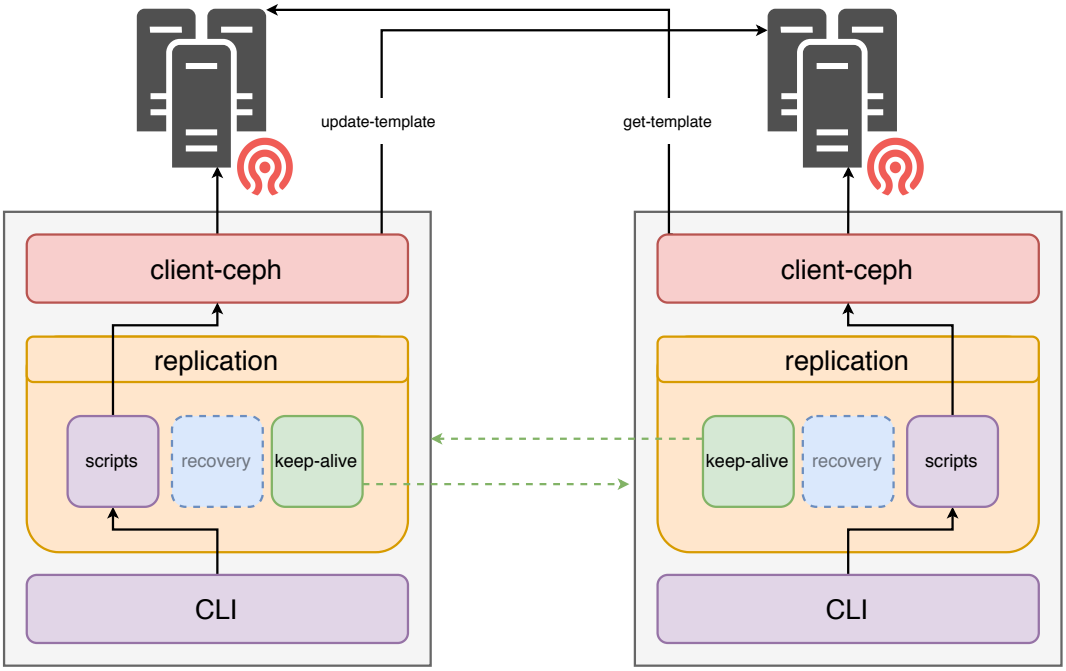


Figura 4.1: Arquitetura do sistema de replicação

[1] O *bootstrapping* de um nó é o processo de inicialização de um nó, e passa pela de criação dos ficheiros de configuração e inicialização do keep-alive no novo servidor.

A figura 4.1 mostra, também, que módulo de replicação contém ainda dois componentes, que realizam os processos de deteção e recuperação de faltas:

- **Keep-alive:** este processo é responsável por verificar o estado (*up* ou *down*) de um determinado servidor; esta verificação é feita a partir dos resultados obtidos de duas fontes: via protocolo ICMP (para testar a conexão IP com o servidor-alvo) e via *ceph-client* (para verificar o estado do *cluster*). O servidor primário vai lançar um processo *keep-alive* para cada servidor-réplica e cada réplica dialoga com o *keep-alive* do primário

- **Recovery:** aquando da detecção de uma falta, o processo de *keep-alive* é terminado e é criado um processo de recuperação para o servidor em falta, processo esse que é semelhante ao dos processos que fazem a verificação do estado do servidor. Após a recuperação do servidor, o processo de *recovery* é terminado e é restaurado o processo *keep-alive* para detectar futuras faltas.

Estes processos gerem o conjunto de membros do sistema, composto por duas listas, uma para guardar os servidores que estão a funcionar corretamente e a outra para guardar os servidores que estão em falta devido a uma falta na rede ou no próprio servidor.

A implementação do módulo de replicação é baseada um conjunto de *bash scripts* que são executados quando existe uma alteração relevante que decorre de uma tarefa de administração, nomeadamente uma adição ou remoção de um nó ou de um *template*. Os *scripts* usam as funções da API do Ceph para interagir com os vários *clusters* Ceph e usam SSH para copiar ou alterar ficheiros, e executar comandos noutros nós.

Recorde-se agora os princípios nos quais se baseia o funcionamento da iCBD:

- **Cliente:** É um dispositivo (e.g., PC, *laptop*) do utilizador, possivelmente sem disco (*diskless*) que realiza o *boot* do sistema de operação (SO), e opcionalmente outro software adicional, por rede, usando uma imagem.

- ***template*:** Imagem (ficheiro) read-only contendo um SO *bootable* e aplicações, a partir da qual se criam *snapshots*.

- **iMI:** Conjunto de ficheiros necessários para realizar um *boot* iCBD, que inclui o *template*, o pacote de boot e ficheiros de configuração adicionais.

- **Snapshot:** Ficheiro read-only contendo as diferenças (versão, actualização, *patch*) entre o estado original de uma iMI e a versão actual, a partir da qual se criam *clones*. Do ponto de vista da utilização prática, indistinguível de uma iMI.

- **Clone:** Ficheiro read-write criado a partir de um *snapshot* e servido a um cliente particular, que o monta e no qual escreve, comportando-se como sendo o disco local desse cliente.

As versões dos *templates* são geridas através do mecanismo **nativo** de *snapshots* presente no RADOS. Quando se realiza uma atualização de um *template* é possível replicar as alterações com recurso a um *snapshot* incremental, transferindo apenas um delta equivalente ao conjunto de dados alterado em relação ao último *snapshot* guardado.

Sendo este sistema baseado numa estratégia de replicação passiva, todas as operações que alteram o estado do sistema são realizadas no servidor primário e posteriormente replicadas pelos restantes nós do sistema. Desta forma, as alterações no conjunto de membros (adição ou remoção de um nó) ou no conjunto de *templates* (adição, remoção ou atualização) têm de ser obrigatoriamente realizadas no servidor primário. Consoante a alteração realizada esta pode, ou não, ser posteriormente replicada pelos restantes nós. Exemplos são: sempre que existe uma alteração do conjunto de membros, esta é aplicada em todos os nós do sistema; contudo, no caso de alteração de um dado *template*, esta alteração apenas é replicada pelos nós que já continham esse mesmo *template* que foi entretanto atualizado.

4.2 Replicação

O exemplo mais simples para uma instalação iCBD com múltiplos *sites* é o apresentado na figura 4.1; aí, cada *site* tem o seu armazenamento suportado num *cluster* Ceph com um certo número de nós (função da capacidade de armazenamento, disponibilidade, performance desejada, etc.).

Em cada *site* haverá que dispôr (para manter a simplicidade) de um nó de replicação, sendo um dos nós configurado como primário e o outro como réplica. Para promover a instalação do software é necessário identificar o nó (que por razões de segregação de papéis poderá ser uma VM ou até um container) e nele instalar, de momento manualmente usando um *script*, os pacotes necessários.

O passo seguinte é configurar os nós, primário e secundário, o que é feito através de alguns ficheiros de configuração (ficheiros de texto): servidor, lista de réplicas, e lista de *templates*, a figura 4.2 apresenta um exemplo de uma possível configuração de dois servidores:

- **Configuração do servidor:** em cada servidor de replicação, este ficheiro contém a) o endereço de IP e o nome do próprio servidor (i.e., do próprio *host* ou container onde está a correr o módulo de replicação), b) o endereço de IP e o nome do servidor primário, e c) o nome do *cluster* Ceph que este servidor, (a), "gere"

- **Lista de servidores:** esta zona contém a lista de servidores-réplica, organizada em dois ficheiros: a que contém a lista dos que estão a funcionar corretamente, e a que contém a lista dos que estão em falta devido a um falta na comunicação ou no próprio nó. Como o servidor primário está definido no ponto anterior, aqui figuram apenas os servidores réplica pertencentes ao sistema de replicação. cada um identificado por um par IP/nome do *cluster* Ceph ao qual estão associados.

```
[server.cfg]
host_ip=10.0.10.110
host_cluster_name=ceph-lisboa
primary_ip=10.0.10.110
host_cluster_name=ceph-lisboa

[server-list.txt]
10.0.20.10;ceph-porto

[down-servers.txt]


[templates-list.cfg]
suse_15.flat.seed;10
w10x1_si.img.seed-zlib;6
```

```
[server.cfg]
host_ip=10.0.20.10
host_cluster_name=ceph-porto
primary_ip=10.0.10.110
host_cluster_name=ceph-lisboa

[server-list.txt]
10.0.20.10;ceph-porto

[down-servers.txt]


[templates-list.cfg]
suse_15.flat.seed;10
```

Figura 4.2: Exemplos de ficheiros de configuração

- **Lista de *templates*:** este ficheiro contém a lista dos *templates* que estão guardados em cada instância de replicação. O servidor primário contém todos os *templates* disponíveis no sistema e cada servidor réplica lista apenas os que estão presentes nesse servidor (e, portanto, nesse *site*). Cada entrada desta lista é representada por um par *template*/versão sendo que existe uma entrada para cada *template* e esta identifica a versão mais recente presente na instância.

Os *scripts* desenvolvidos fornecem um conjunto de comandos que são executados na CLI (*Command Line Interface*) do servidor primário, e permitem a) gerir todo o sistema de replicação e b) realizar as alterações em imagens e desencadear, depois, a sua replicação pelos restantes nós.

A actualização de um *template* tira partido de *snapshots* incrementais para enviar apenas as partes da imagem que foram alteradas com a atualização. Portanto, depois de se realizar uma actualização de um template, o servidor primário cria um novo *snapshot*. De seguida, todos os servidores-réplica ativos são contactados para verificar quais possuem o *template* em causa, caso em que é enviado um *snapshot* incremental que é depois aplicado sobre a imagem guardada no *cluster* da instância iCBD "remota".

O processo de administração dos nós do sistema de replicação permite adicionar ou remover instâncias (réplicas) a partir do servidor primário, e como estas operações implicam a alteração da lista de membros, é necessário replicá-las por todos os nós do sistema. Quando é adicionado um nó ao sistema, a inicialização do novo nó é realizada pelo servidor primário sendo que este é responsável pela criação dos ficheiros de configuração assim como o inicio do processo de *keep-alive* no novo nó.

O servidor de administração tem a capacidade de detetar faltas nos servidores-réplica mantendo localmente um processo de keep-alive para cada réplica e permitindo assim detetar faltas de comunicação ou indisponibilidade ((e.g., *crash*) de um nó. As faltas são

detetadas usando uma dupla verificação: a) recorrendo ao protocolo ICMP (via *ping*), o que permite verificar o estado da ligação entre o primário e a réplica; e b) verificando o estado do *cluster* Ceph gerido pelo servidor-réplica. Quando uma destas condições não se confirma para um determinado nó, este é marcado como indisponível devido a uma falta e, durante o período de indisponibilidade, não recebe qualquer tipo de actualizações. É então iniciado um processo de recuperação que observa o estado do nó até que a falta seja eliminada e o nó volte a estar disponível no sistema. Quando um servidor réplica volta a estar disponível, este é responsável por contactar o servidor primário afim de verificar se existem novas actualizações para os *templates* que o nó-réplica contém.

Tal como o servidor primário, cada réplica possui um processo de detecção e recuperação de faltas e um processo keep-alive que "dialoga" com o seu par no servidor de administração, podendo assim, também, observar o seu estado.

Importa contudo referir que a falta do servidor primário não impede o funcionamento das restantes réplicas desde estas contenham os *templates* que são maioritariamente utilizados nos seus *sites*, apenas impossibilita a receção de atualizações de *templates* efetuadas no primário; mas estas podem ser transferidas quando a falta for tratada.

4.3 Descrição de comandos

Os *scripts* desenvolvidos proporcionam um conjunto de comandos para serem utilizados nos nós de replicação e suportam-se em a) utilitários do sistema de operação para manipular ficheiros, criar processos, e ligações para comunicação (SSH) com *hosts* remotos, e b) utilitários do Ceph para administrar os objectos de armazenamento que albergam iMIs, *templates*, *snapshots* e *clones*.

Para simplificar a gestão do sistema de replicação, foram desenvolvidos dois *scripts* para apresentar o estado do sistema que acedem a todos os nós, coligem a informação guardada nos ficheiros de configuração, e exibem-na numa forma mais amigável.

icbd-list-membership Este comando permite apresentar os vários nós que constituem um sistema. Quando é executado no servidor primário é possível observar o estado (*up/down*) de todos os nós do sistema; no caso da execução ser efetuada num servidor-réplica, apenas é apresentado o estado do primário.

icbd-list-templates A listagem dos *templates* existentes num nó é realizada através deste comando. Quando este é executado no servidor primário, são apresentados os *templates* de todos os nós do sistema; no caso da execução ser efetuada num servidor-réplica, são exibidos os *templates* guardados nessa réplica e ainda os disponíveis no servidor primário

Como referido anteriormente, existe um conjunto de comandos que se destina ser executado diretamente no servidor primário, para gestão do conjunto de nós e *templates* sob alçada do sistema de replicação, e que são os seguintes:

icbd-add-primary –ip <host-ip> –cluster <cluster-name> Este comando permite a inicialização de um sistema de replicação, e deve ser executado uma única vez no *host* que contém o IP definido nos argumentos do comando. Essencialmente, este comando gera os ficheiros de configuração para o servidor primário criando a associação entre o *host-ip* e o *cluster* Ceph. Os comandos listados a seguir utilizam estes ficheiros de configuração e portanto requerem que a inicialização tenha sido efetuada.

icbd-add-replica –ip <host-ip> –cluster <cluster-name> A adição de novas réplicas ao sistema é realizada com recurso a este comando que permite o registo e inicialização de um novo nó no sistema de replicação. Este processo divide-se em três partes: 1) criação dos ficheiros de configuração no *host* definido; 2) atualização de todos os outros nós com informação sobre o novo nó; 3) incialização do processo de keepalive no servidor primário (para diálogo com o novo nó) e no novo servidor-réplica (para diálogo com o servidor primário).

icbd-rm-replica –ip <host-ip> Remove o *host* referido da lista de nós do sistema de replicação.

icbd-add-template –template-file <file> –rbd-path <poolname/rbd-name> Transfere o *template* iCBD indicado, que está armazenado no disco local de uma estação de administração iCBD para o *cluster* Ceph primário. Este comando tira partido da API Ceph RADOS e permite ao servidor primário armazenar o ficheiro definido como uma imagem RBD no seu *cluster* ficando disponível para todos os nós da iCBD.

icbd-register-template –rbd-path <poolname/rbd-name> Regista no sistema de replicação um *template* já armazenado no Ceph.

icbd-rm-template –rbd-path <poolname/rbd-name> Remove um *template* da alçada (de todos os nós) do sistema de replicação, apagando-o da lista e do *cluster* Ceph.

icbd-update-template –rbd-path <poolname/rbd-name> Este comando deve ser utilizado após uma atualização de uma imagem para replicar as alterações por todos os nós que subscreveram essa imagem. Como referido anteriormente, as replicações são realizadas com recurso a *snapshots* incrementais para transferir apenas um delta da imagem total correspondente aos dados alterados com o *update*.

Cada servidor réplica dispõe de duas operações para gerir os seus *templates*, e esses comandos devem ser apenas executados em servidores-réplica:

icbd-replica-fetch –rbd-path <poolname/rbd-name> Transfere (copia) para o servidor-réplica onde é executado o comando um *template* existente no primário, especificado pela sua localização no Ceph.

icbd-replica-rm-template –rbd-path <poolname/rbd-name> Permite remover um *template* de um servidor réplica, apagando-o da lista de *templates* desse *site* e do *cluster* Ceph.

4.4 Requisitos e Instalação do Sistema de Replicação

Para que a replicação funcione bem é necessário realizar a instalação do módulo de replicação no primário e configurar correctamente os nós. Um aspecto importante a ter em conta é que o módulo desenvolvido pode ser executado em qualquer um dos nós pertencentes ao *cluster*, de forma aproveitar o máximo dos recursos usados, pagando-se o preço da introdução nos nós de software *estranho* ao Ceph. Como anteriormente referimos, a segregação de papéis é uma boa prática.

Requisitos

O primeiro passo é a criação de um utilizador para executar os *scripts*. Este tem de ser capaz de usar comandos com privilégios *root* sem necessitar de usar password, não só local como em nós remotos. Além disto, todos os nós têm de ser capazes de realizar ligações SSH entre eles, sem necessidade de inserir uma *password* para autenticar a ligação. Este passo também faz parte da sequência de pré-configuração dos nós de um *cluster* Ceph e, portanto, ao usar um nó pertencente a um *cluster* podemos tirar partido do utilizador já existente para executar o módulo. Contudo, continuamos a advogar a separação de papéis...

De seguida, dado que um nó de replicação pode gerir vários *clusters* em *sites* distintos - ou até no mesmo *site*, por exemplo quando se trata de uma instituição dispersa por um *campus* - embora não seja uma configuração recomendada, para além de ser necessário a instalação do cliente Ceph (através do pacote ceph-common), é necessário incluir também os ficheiros de configuração usados pelo cliente, nomeadamente o ficheiro de configuração para a conexão ao *cluster* (ceph.conf) e o ficheiro com a chave de autenticação. Assim, cada nó de replicação tem de conter os ficheiros de configuração de todos os *clusters* que vão ser manipulados pelo sistema de replicação.

O esquema de nomeação dos ficheiros (e directorias) do Ceph permite o uso de meta-variáveis que facilita a configuração do *cluster*. Quando é utilizada uma meta-variável, o Ceph expande o seu valor e transforma-o em algo concreto, como por exemplo:

```
ceph health --cluster $cluster
```

Neste comando, a meta-variável $cluster identifica o nome do *cluster*, e com isso o Ceph usa o valor presente na variável para determinar quais os ficheiros de configuração a usar na conexão da seguinte forma: /etc/$cluster.conf (ficheiro de configuração) e /etc/$cluster.keyring (ficheiro com a chave de autenticação).

Instalação

O software desenvolvido dispõe de um *script* (install.sh) que quando executado, permite a instalação do módulo de replicação num *host*.

A instalação passa pela criação das directorias que vão conter os ficheiros do sistema de replicação, assim como a cópia dos *scripts* para o local correcto e finalmente a adição de um comando (icbd-rep-restart) ao ficheiro /etc/rc.local para que o módulo possa ser inicializado quando o *host* é (re)iniciado.

Existe também de um *script* (uninstall.sh) para desinstalar o módulo do sistema, que remove todos os ficheiros e directorias associados ao software de replicação, bem como reverte as alterações realizadas em ficheiros pré-existentes efetuadas durante a instalação.

Capítulo 5

Avaliação

Este capítulo descreve o processo de avaliação da solução implementada no projecto, incluindo a configuração do ambiente de testes e os resultados obtidos nos testes realizados para medir o desempenho da solução.

Os testes realizados têm duas vertentes: a primeira destina-se à avaliação do desempenho da utilização do Ceph como candidato ao sistema de armazenamento de *templates*, iMIs e *clones* que suportam o funcionamento dos clientes iCBD. A segunda vertente corresponde à avaliação da solução de replicação desenvolvida.

5.1 Configuração

5.1.1 Infraestrutura iCBD

Para a implementação da infraestrutura utilizada neste trabalho, foi aproveitada uma grande parte da anteriormente usada em [11], nomeadamente as máquinas que compõem o serviço iCBD bem como os postos de trabalho - tanto reais como virtuais (VMs).

Para a execução das instâncias iCBD em clientes reais foram disponibilizados dois laboratórios, cada um equipado com 15 postos de trabalho (PCs) cujas especificações estão descritas na tabela 5.3. Os postos de cada laboratório são interligados a um switch Ethernet, e durante algum tempo foi possível experimentá-los com um switch de 1 Gbps, mas que depressa reverteu para o switch de 100 Mbps usado nos laboratórios durante o ano letivo. Note-se que estes laboratórios estão sujeitos a regras de controle de acessos (ACLs) muito estritas implementadas nos switches, sendo o tráfego para fora do laboratório suportado sobre VLANs (uma por laboratório), o que significa que há *overheads* na transferência de pacotes entre os PCs e os servidores, já que a passagem de pacotes de uma VLAN para outra passa pelo *firewall* da FCT/NOVA.

CPU	2x Intel Xeon E5-2670 v3 @ 2.30GHz
Memória	128GB
Tipo de Controlador	12Gbps SAS
Ethernet	2 portas a 10Gbps + 4 portas a 1Gbps
Hipervisor	VMware ESXi, 6.5.0, 4564106

Tabela 5.1: Especificação do hardware dos servidores

Controladores	2 MSA 2040 SAS
Capacidade Total	7,2TB
Discos	12 HP SAS 600GB 10k Rpms
Interfaces	8 portas SAS de 12Gbps (4 por controlador)
Ethernet	2 portas de 1Gbps (1 por controlador)

Tabela 5.2: Especificação do *array* de discos

CPU	Intel Core i3-7100 @ 3.90GHz
Memória	8GB
Armazenamento	275GB SSD
Largura de Banda	1 Gbps

Tabela 5.3: Especificações dos postos de trabalho

Todas as peças que compõem a iCBD, incluindo os postos de trabalho virtuais usados nos testes são suportadas por hipervisores de tipo 1 (VMWare ESXi 6.5) instalados em dois servidores HPE ProLiant DL380 Gen9 (especificações na tabela 5.1), formando um *cluster* VMware vSphere. Estes servidores usam como espaço de armazenamento volumes obtidos num *array* de discos (HPE MSA 2040 SAN, ao qual estão ligados por Serial-Attached SCSI, SAS) e são interligados por um *switch* (HPE Flexfabric 5700) que fornece uma largura de banda de 10 Gbps, as especificações do *array* de discos estão apresentadas na tabela 5.2.

No caso da plataforma iCBD, esta é composta por três servidores (albergados noutras tantas máquinas virtuais): o servidor de imagens (icbd-imgs) executa grande parte dos serviços essenciais ao funcionamento da iCBD e contém todos os *templates* disponíveis na plataforma (armazenados em btrfs); o servidor (icbd-rw) para o suporte dos volumes *read-write* das instâncias; e o servidor que contém as directorias *home* dos utilizadores e que são montadas nos clientes (icbd-home). No âmbito do nosso trabalho, a única peça da iCBD que vai ser usada será o servidor de imagens, dado que o suporte para o espaço de armazenamento das instâncias é realizado pelo *cluster* Ceph. A tabela 5.4 apresenta as especificações de cada máquina virtual que executa um serviço iCBD.

5.1.2 Sistema de Armazenamento OBS da iCBD

Para a integração na iCBD do suporte de armazenamento de iMIs num sistema de armazenamento baseado em objectos (OBS), e ainda de um sistema de replicação, foram criados dois *clusters* Ceph. O primeiro (tabela 5.5) é usado para guardar todos os *templates* da

Host	vCPUs	vRAM	Armazenamento
icbd-home	4	8GB	16GB+100GB
icbd-imgs	4	32GB	16GB+600GB
icbd-rw	4	8GB	16GB+300GB

Tabela 5.4: Topologia do sistema iCBD

Tabela 5.5: Topologia do *cluster* Ceph primário

Host	vCPUs	vRAM	Armazenamento
ceph-mon	1	2GB	16GB
ceph-ost1	2	4GB	16GB+32GB+64GB
ceph-ost2	2	4GB	16GB+32GB+64GB
ceph-ost3	2	4GB	16GB+32GB+64GB
ceph-ost4	2	4GB	16GB+128GB+128GB

Tabela 5.6: Topologia do *cluster* Ceph secundário

Host	vCPUs	vRAM	Armazenamento
ceph2-mon	1	2GB	16GB
ceph2-ost1	2	2GB	16GB+32GB+128GB
ceph2-ost2	2	2GB	16GB+32GB+128GB
ceph2-ost3	2	2GB	16GB+32GB+128GB

plataforma que suporta instâncias iCBD armazenadas no Ceph, e corresponde à instância primária de replicação; o segundo (tabela 5.6) é a instância-réplica. Assim, testam-se as funcionalidades e desempenho do módulo de replicação, e prova-se que um cliente iCBD é capaz de arrancar tanto na instância primária como na réplica, fornecendo assim um serviço escalável, geograficamente distribuído se necessário, e facilmente administrável a partir de um ponto único.

De referir que ambos os *clusters* são realizados por máquinas virtuais hospedadas num dos servidores descritos na secção anterior. Cada VM especificada nas tabelas 5.5 e 5.6 executa os vários processos (descritos em 2.3.2.1) que permitem formar um *cluster* Ceph, e ambos os *clusters* seguem o mesmo esquema de configuração: a máquina ceph-mon executa os dois processos de monitorização do *cluster*: *monitor* e *manager*. Cada ceph-ost, Object Storage Target (OST), corre dois Object Storage Daemons (OSD)s que agregam dois discos (que não o disco do sistema de operação) ao espaço de armazenamento do *cluster* - por exemplo, no caso do *cluster* Ceph primário, os discos de 16GB são os discos do SO, e os de 32GB e 64GB discos "para a grande *pool* de armazenamento de objetos".

A figura 5.1 mostra a topologia da rede de todo o sistema. O *cluster* Ceph primário, representado à esquerda é composto por 5 máquinas, cada OSD (2 por cada ceph-ost*) está ligado a uma rede privada (10.0.10.0/24) para permitir a troca de mensagens referentes a tarefas de replicação, recuperação ou detecção de faltas (intrínsecas ao próprio

Ceph) permitindo assim distribuir a carga por diferentes redes. Os clientes interagem com o *cluster* através da rede pública do laboratório (10.171.110.0/24). O *cluster* "réplica", representado à direita na figura, tem essencialmente a mesma configuração que o anterior, tendo no entanto menos OSDs e não contém uma rede privada para transmitir o tráfego gerado pelos OSDs. Este *cluster* apenas vai ser usado pelos módulos de replicação e a rede publica do *cluster* (10.0.2.0/24). Cada módulo de replicação (icbd-ceph-replication) corre num *host* (VM) dedicado que tem acesso a ambas as redes públicas dos dois *clusters*. Estes servidores de replicação são muito simples, correm um CentOS 7 mínimo e executam o software de replicação. Como referido anteriormente, o único componente da infraestrutura iCBD que vai ser usado é a máquina icbd-imgs, que é acessível através da rede publica (10.171.110.0/24) do laboratório onde estão os postos de trabalho que simulam os clientes.

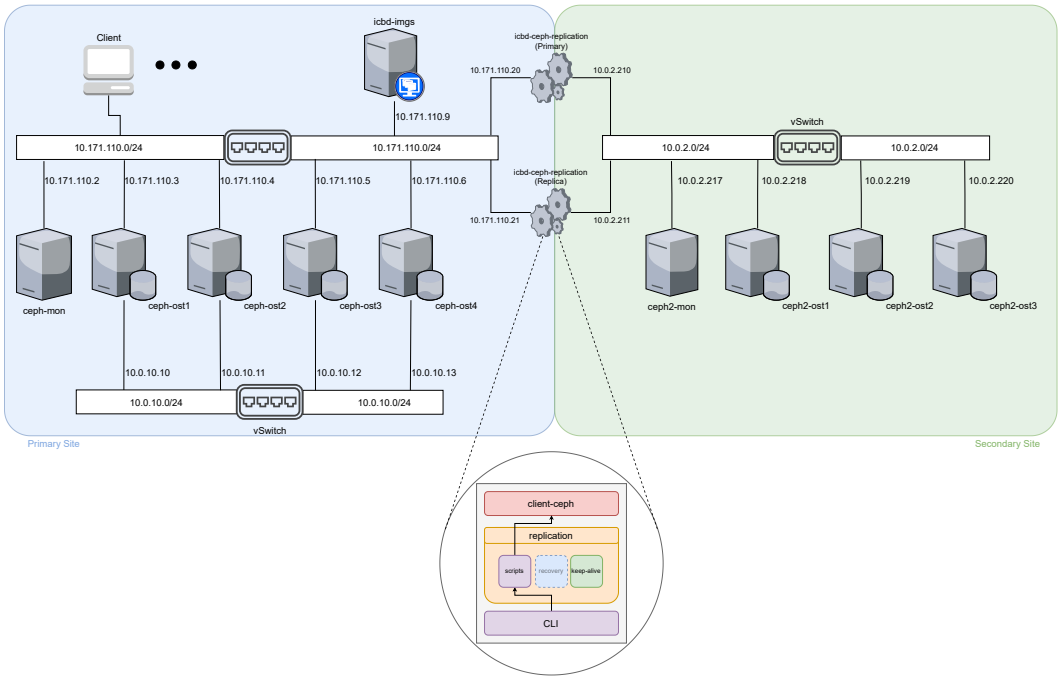


Figura 5.1: Topologia do Sistema

A instalação do Ceph segue todos os pontos descritos na documentação [30] para criar um *cluster* Ceph, documentação essa que está constantemente a ser actualizada e por este motivo deve-se sempre seguir as indicações que nela constam. Neste projecto o Ceph foi instalado sobre o SO CentOS 7 [31], pelo que todas as máquinas que pertencem ao *cluster* correm CentOS 7. É ainda importante referir que para o correto funcionamento do módulo de replicação, cada *cluster* criado tem que ter um nome único (sendo que o padrão para o prefixo é ceph), pelo que os *clusters* criados foram denominados ceph e ceph2.

Um aspecto importante da configuração do Ceph é que este não implementa todas as recomendações sugeridas pela documentação para se obter o melhor desempenho do

cluster, nomeadamente a utilização de SSDs para guardar os registos das escritas realizadas no *cluster* (esta estrutura é denominada de *journal*). A não utilização de SSDs implica que esta informação (journal) seja guardada no mesmo volume que contém os dados do *cluster* criando um pequeno *bottleneck* quando existem muitas escritas no sistema - mas estas recomendações, nesta PoC perdem a sua razão de ser pois os próprios nós Ceph são VMs. Por outro lado, durante os testes finais, já em tempo de pandemia e com alguns eventos de quebras de energia no *campus* da FCT/NOVA, o *cluster* primário sofreu a perda de dois OSDs que, por não estarem a funcionar correctamente e estarem a degradar bastante a performance do *cluster* no geral, foram removidos. Os OSDs retirados eram ambos do mesmo *host* (ceph-ost1) e foram removidos pois estavam na situação de *flapping*[32]: sempre que um cliente realizava uma conexão com o *cluster* os dois OSDs danificados falhavam e eram dados como mortos; no instante seguinte, estes OSDs recuperavam, para falhar novamente até serem reiniciados. Isto obrigava o *cluster* a accionar os mecanismos de recuperação de dados afectando consideravelmente a performance do sistema. Como não estava a ser exequível a resolução do erro, decidiu-se desligar o *host* (ceph-ost1), o que permitiu ao *cluster* obter um desempenho muito melhor. No entanto é fundamental referir que os testes documentados posteriormente não têm em conta esta falha.

5.1.3 Ambiente de teste

Foi usado o Suse Enterprise Linux 15 [33] (Suse 15) como SO para "bootar" nos vários clientes de teste, uma vez que esta é uma distribuição Linux bastante estável e contém disponíveis os vários pacotes Ceph que podem ser facilmente instalados através do gestor de pacotes da distribuição. Este SO será utilizado em dois cenários distintos:

Nativo (apenas disponível para imagens Linux) o SO corre directamente sobre o hardware do posto de trabalho cliente.

Virtualizado é executada uma imagem contendo um hipervisor (de momento o VMware Workstation) e este executa uma VM com um qualquer SO *guest* suportado - Linux, windows, etc..

A preparação do *cluster* Ceph para receber as iMIs passa pela criação de duas *pools* a primeira, com o nome "icbd", é usada armazenar as iMIs no tipo *read-only*, que são consideradas *golden-images*. É para esta *pool* que são importadas inicialmente as iMIs assim como as suas atualizações; nesta *pool* estão essencialmente os RBDs com os seus respetivos *snapshots*. A segunda *pool* criada, com o nome de "icbd-rw", corresponde à *pool* na qual se vão criar os volumes de armazenamento *read-write* (criados a partir dos *snapshots* presentes na primeira *pool*) que possibilitam o correcto funcionamento do SO nos clientes.

Para realizar o *boot* de uma imagem armazenada no Ceph foi realizada a cópia de uma iMI armazenada em btrfs para o *cluster* Ceph. No entanto a iMI teve de sofrer algumas

alterações, nomeadamente com a adição dos pacotes que constituem o Ceph e contêm as ferramentas (como o módulo de *kernel* rbd.ko que fornece acesso a um volume RBD) que possibilitam bootar a imagem armazenada num RBD, Além disso é necessário guardar (no pacote de *boot*) as chaves de autenticação para que um cliente que pretenda bootar uma imagem armazenada num *cluster* Ceph consiga autenticar-se e aceder aos dados guardados no *cluster* - e neste caso foi usada a chave padrão (de administração). Por fim, a última alteração necessária para usar um RBD para realizar o *boot* de uma máquina passa pela remoção das características padrão que estão associadas a um RBD recém criado num *cluster* Ceph - algumas destas características não são compatíveis com o módulo do *kernel* usado para realizar a ligação com o RBD pretendido, como as deep-flaten, fast-diff, object-map e exclusive-lock.

Pouco tempo depois de se dar início aos testes, foram realizadas algumas alterações na rede da faculdade que prejudicaram imenso o resultado dos testes: a imposição de novas restrições de segurança (como ACLs e regras de *firewall*) nos Laboratórios que estavam a ser usados para os testes, prejudicou o desempenho do Ceph em virtude dos elevados tempos de *boot*, tornando a solução impraticável. Neste momento, a situação vigente causada pela pandemia e confinamento obrigatório, impossibilita a realização de mais testes, no entanto existem resultados preliminares que serão apresentados nas próximas secções.

5.2 Metodologias e Métricas de Avaliação

5.2.1 Testes de Funcionalidade

Módulo de replicação

Tal como referido anteriormente foram criados dois *clusters* Ceph, para os testes de funcionamento do módulo de replicação desenvolvido. Para confirmar que o processo de replicação funcionava correctamente, deram-se os seguintes passos:

- **Criação de uma iMI:**
  1. Num cliente ("de administração") Ceph conectado ao *cluster* primário, é importado um *template* de uma iMI, ficando assim registado no módulo de replicação o *template* t1 com o *snapshot* s0
  2. Posteriormente é ligado um cliente iCBD para realizar o *boot* do *template* recém importado, podendo-se verificar assim o seu funcionamento.
- **Primeira Replicação da iMI (cópia integral):**
  1. No cliente-réplica Ceph, conectado ao *cluster* réplica, o volume RBD (correspondente ao *template*) é importado através do *script* `icbd-replica-fetch` descrito no capítulo anterior.

2. Depois de terminada a transferência, é ligado um cliente iCBD que realiza o *boot* da iMI replicada obtendo-se um resultado idêntico ao *boot* realizado por via do *cluster* primário.

- **Actualizações da iMI (replicação por *snapshot* incremental):**

1. Concluída a primeira replicação da iMI (por cópia integral), os dois *clusters* possuem cópias iguais da mesma iMI (t1) com um primeiro *snapshot* (s0).
2. Agora, através da plataforma de administração da iCBD é realizada uma nova actualização sobre o *template* t1 presente no *cluster* primário, originando um novo *snapshot* s1.
3. Depois de verificar a funcionalidade da nova actualização do *template*, é executado o *script* `icbd-update-template` no cliente ("de administração") Ceph conectado ao *cluster* primário, iniciando a transferência do delta entre os *snapshots* s0 e s1.
4. Terminada a transferência, é ligado um cliente iCBD para realizar o *boot* do *template* recém actualizado no *cluster*-réplica verificando que a actualização está correcta.

Para provar a funcionalidade do trabalho realizado foi criado um RBD vazio (no *cluster* primário) para de simular uma iMI a ser replicada entre dois *clusters*. O RBD criado foi então mapeado como um volume de armazenamento num cliente Ceph, essencialmente este cliente é apenas um *host* que contém os pacotes Ceph assim como as chaves de autenticação, posteriormente foi montado nesse RBD um sistema de ficheiros (ext3) e adicionado um ficheiro (f0), depois de desligado o cliente foi criado um *snapshot* (s0) do novo RBD. Este processo simula um administrador a criar uma nova iMI no sistema de armazenamento.

Para testar a funcionalidade do módulo foram realizados dois tipos de cópias: integral e incremental. Para a primeira foi realizada uma exportação integral do volume RBD do *cluster* primário e importação para o secundário, depois de realizada a operação, o RBD presente no *cluster* secundário foi montado no mesmo cliente Ceph que permitiu verificar que o RBD continha o sistema de ficheiros com um único ficheiro (f0). Neste ponto ambos os *clusters* contêm os mesmo dados: um RBD como um unico *snapshot* (s0).

O teste da funcionalidade da cópia incremental implica que já exista o mesmo RBD em ambos os *clusters*, desta forma, foi novamente mapeado o RBD criado no *cluster* principal ao qual se adicionou um novo ficheiro (f1), e depois de desligada a conexão, foi criado um novo *snapshot* (s1). De seguida tirou-se partido das operações (`export-diff` e `import-diff`) para realizar a cópia incremental que corresponde ao delta de dados alterados entre os dois *snapshots* (s0 e s1). Após terminar a operação, o RBD do *cluster* secundário foi novamente mapeado num cliente Ceph no qual se verificou que contém ambos os ficheiros f0 e f1. Com isto ambos os *clusters* ficaram novamente com os dados iguais provando a funcionalidade do módulo de replicação.

Também foram testadas as componentes recuperação e detecção de faltas. Para este aspecto apenas basta verificar a falta e consequente recuperação de uma réplica, uma vez que todas as actualizações de iMIs passam pela instância primária, pelo que no caso desta faltar, não será possível registar novas iMIs ou actualizações das existentes, nem distribuir aos *sites* secundários actualizações às iMIs até a instância primária voltar a estar operacional. Assim sendo, o teste realizado passou por: a) distribuir a mesma iMIs pelas duas instâncias; b) desligou-se a instância correspondente à réplica e realizaram-se 2 actualizações consecutivas à iMI em causa; c) por fim voltou-se a ligar a instância réplica e, passado o tempo de replicação, pudemos observar no *cluster* secundário os dois novos *snapshots*, resultantes dos *updates* realizados durante o período de indisponibilidade do secundário.

Ceph como sistema de armazenamento de iMIs

Para testar a integração do Ceph como sistema de armazenamento de iMIs é suficiente conseguir realizar um *boot* de uma imagem guardada num RBD. Tal como descrito em 5.1.3 foi importada para o *cluster* primário a iMI referente ao sistema de operação Suse 15. Com a iMI armazenada no *cluster* usou-se um posto de trabalho físico (um PC presente nos laboratórios descritos anteriormente) para arrancar a imagem contida no iMI. Assim, depois de se ligar o posto de trabalho, configurado para realizar um *boot* via rede usando o protocolo PXE, é apresentada uma imagem com as imagens disponíveis 5.2; seleccionada no menu a opção que corresponde à iMI armazenada no *cluster* (na parte inferior da figura 5.2 é possível observar a linha de comando do processo de *boot*) que depois de terminado o processo de *boot* (cujo desenrolar se pode ver na figura 5.3) nos apresenta um ambiente de trabalho totalmente funcional (figura 5.4).

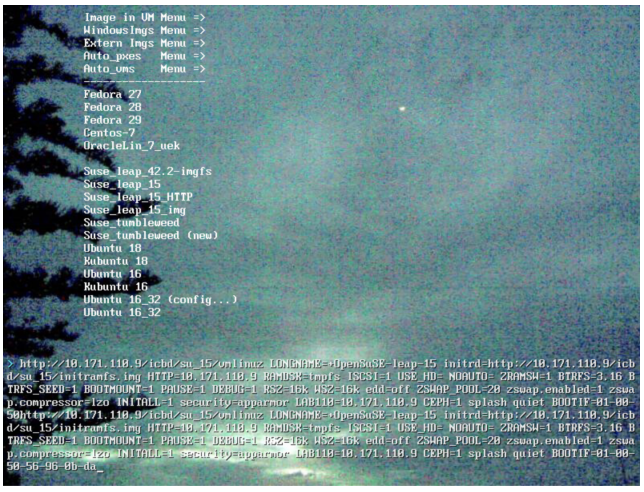


Figura 5.2: Menu PXE

É importante referir que foram realizados testes de funcionalidade para o arranque nativo do Suse 15 (ou seja, o PC executa o Suse 15 por *boot* via rede) assim como o arranque virtualizado do mesmo (ou seja, o PC executa um SO - que pode ser um Linux

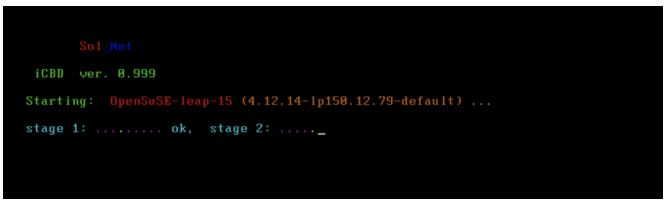


Figura 5.3: Progresso de um *boot* iCBD

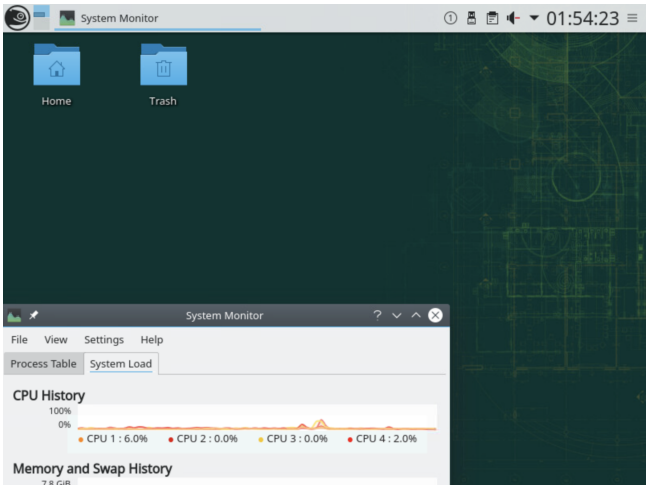


Figura 5.4: Máquina Linux funcional

qualquer, incluindo o Suse - por *boot* via rede, SO esse que contém o VMware Player, e este último executa o Suse 15 numa VM). Além disto foi também testada a funcionalidade do arranque virtualizado de uma imagem windows suportado por um hipervisor contido no Suse 15 (que é executado nativamente). O desempenho do arranque nativo comparativamente ao virtualizado é consideravelmente mais lento visto é necessário executar dois SOs um para criar e arrancar a VM que posteriormente executa a imagem (SO) final desejada.

5.2.2 Metodologia para Avaliação do Desempenho do Ceph

A primeira parte desta secção destina-se a avaliar o desempenho do Ceph como sistema de armazenamento da iCBD, uma vez que esta integração é uma nova funcionalidade do projecto, e importante contributo desta dissertação. É portanto necessário documentar o seu desempenho e compará-lo com as implementações de outros sistemas de armazenamento já existentes no iCBD, como é o caso do btrfs.

Assim, esta parte corresponde a uma avaliação do serviço iCBD (sobre Ceph), pelo que a melhor forma de medir o desempenho é escolher as componentes que melhor definem a experiência do utilizador. Neste caso, a componente escolhida foi o tempo de *boot* de uma instância iCBD, pois permite facilmente tirar conclusões acerca da qualidade do serviço fornecido. Para obter este tempo é usada a ferramenta de sistema `systemd-analyse` que nos apresenta o tempo que a máquina demorou a ficar funcional, tal como mostra a figura

5.5.

```
w-6ebd:~ # systemd-analyze
Startup finished in 14.946s (kernel) + 22.252s (userspace) = 37.198s
w-6ebd:~ #
```

Figura 5.5: Exemplo do comando `systemd-analyze`

5.2.3 Metodologia para Avaliação do Sistema de Replicação

Para a avaliação do sistema de replicação foram realizados alguns testes de desempenho para confirmar a correcção da solução. O objectivo destes testes é calcular o tempo médio de replicação de uma imagem total e parcial (quando existe um *update*). Para obter o tempo de replicação é usado o comando `time` que apresenta o tempo decorrido desde que o comando foi executado até terminar a tarefa submetida. Assim, executamos, por exemplo, `time icbd-cache-fetch` e, depois de terminada a tarefa, é apresentado o tempo, tal como mostra a figura 5.6.

```
Exporting image: 100% complete...done.
Importing image: 100% complete...done.
template transfer complete!
icbd/su_15.flat.seed version: 13
done!

real    3m7.611s
user    0m20.091s
sys     0m33.189s
```

Figura 5.6: Tempo apresentado pelo comando `time`

5.3 Testes

5.3.1 Ceph

Como referido anteriormente, durante a realização dos testes deparamos com vários obstáculos que impossibilitaram uma avaliação completa. No entanto, foi possível recuperar alguns resultados preliminares registados antes de começarem a surgir os referidos obstáculos. Assim, começamos por apresentar a figura 5.7 onde estão representados os tempos médios de *boot* referentes aos 15 postos de trabalho de um laboratório actual do DI-FCT/NOVA, quando são iniciados (efectuado o *power-up*) todos "ao mesmo tempo".

Os dados apresentados na figura 5.7 correspondem aos resultados preliminares obtidos, para 5 tipos diferentes de *boot*. O primeiro tipo, "nativo" corresponde ao *boot* nativo do Linux Suse 15, sendo que a diferença entre os dois primeiros casos é a forma como está armazenada a iMI do cliente no Ceph: como um volume *read-only* (RO) em conjunto com um volume *read-write* (RW) ou como um único *thin-clone* que permite a escrita por parte do cliente. Os testes referentes ao segundo tipo de *boot*, virtualizado, não puderam ser realizados nos 15 postos (PCs) dos laboratórios, pelo que os valores apresentados na figura representam os testes iniciais, realizados num único PC.

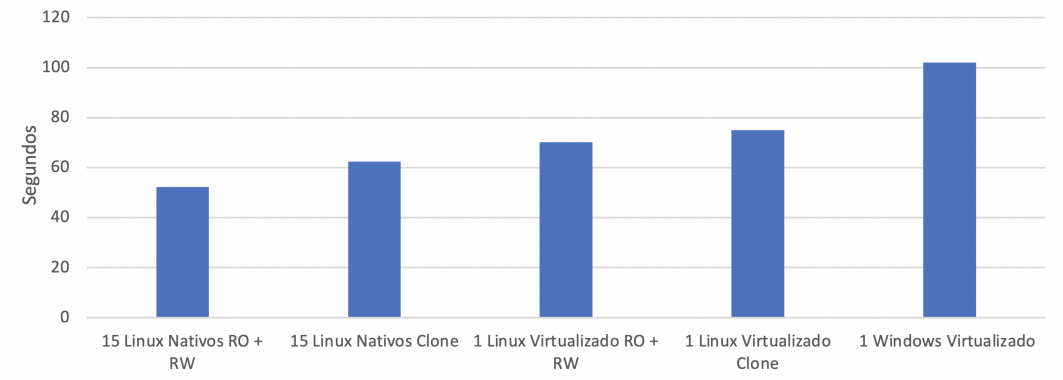


Figura 5.7: Tempos de *boot* nativo (Suse 15)

5.3.2 Replicação

A avaliação do desempenho do sistema de replicação tem duas vertentes: a primeira avalia a replicação por transferência integral de todo um *template*, situação que se verifica quando um *site*-réplica requisita um *template* pela primeira vez; a segunda, avalia a replicação quando no primário há uma actualização de um *template* que já existe no secundário, caso em que apenas são transferidas as porções do *template* que correspondem ao conjunto de dados que foi alterado.

Para ambos os casos, foram realizadas várias execuções (i.e. transferências) entre os dois *clusters* representados na figura 5.1. Neste caso, foi realizada uma importação para o Ceph de uma imagem btrfs designada su_15.flat.seed com 24 GB que originou um primeiro *snapshot* su_15.flat.seed@15. Posteriormente, com uma nova actualização da mesma imagem, foi criado um novo *snapshot* su_15.flat.seed@16, que é a versão mais recente. Assim a primeira parte do teste consistiu em transferir apenas a versão mais antiga (@15) do *template*, presente no *cluster* primário, para o *cluster* secundário, várias vezes (para identificar situações de tempos de transferência extremos). Note-se que o Ceph reporta 24 GB como espaço provisionado e 9.4 GB como espaço realmente utilizado. Na segunda parte, depois de realizada uma actualização no *template*, que se traduziu num ficheiro delta com 2GB, foram novamente realizadas várias transferências.

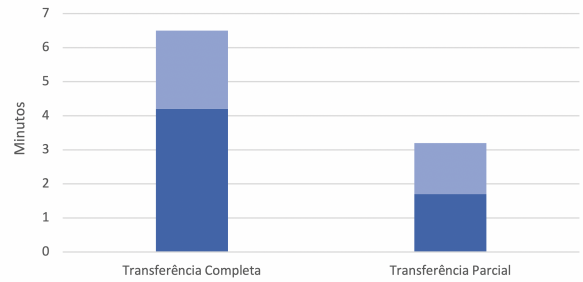


Figura 5.8: Tempos de replicação

Na figura 5.8 estão representados os tempos mínimo e máximo relativos às replicações

completa e parcial de um *template*. Os valores apresentados para os tempos para replicação total de um volume de dados de 24GB (imagem original); no caso da replicação parcial, o volume de dados replicado foi de 2GB (delta correspondente à actualização efectuada sobre a imagem anterior).

Capítulo 6

Conclusões e Trabalho Futuro

Este capítulo apresentas conclusões do trabalho realizado, nomeadamente uma muito sumária análise de resultados, comparando a utilização do Ceph como sistema de armazenamento da iCBD com a utilização do btrfs, e deixa algumas sugestões para melhorias que podem ser concretizadas em trabalhos futuros.

6.1 Análise de Resultados

A figura 6.1 apresenta a comparação dos resultados entre as duas implementações distintas de sistemas de armazenamento da iCBD, ou seja, a iCBD usando btrfs - para a qual os resultados foram extraídos de [11] - e usando Ceph.

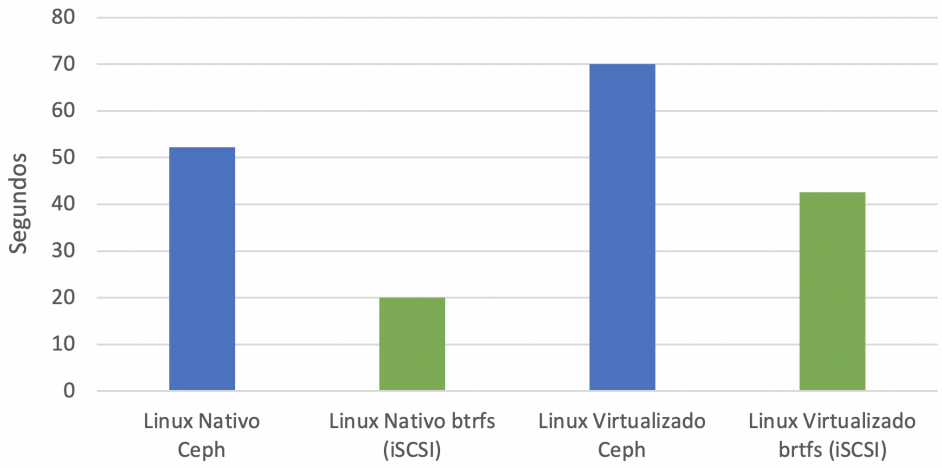


Figura 6.1: Tempos de boot de clientes usando armazenamento btrfs ou Ceph

À primeira vista os resultados podem parecer decepcionantes, mas são facilmente explicáveis: a) no caso do btrfs, foi usada uma VM de armazenamento para os *templates*

(icbd-imgs) e uma VM para armazenamento dos clones (icbd-rw) privados de cada estação de trabalho, e é possível que essas VMs tivessem sido instanciadas em servidores distintos, distribuindo a carga tanto em termos de CPU e RAM, como em termos de armazenamento físico (por exemplo, dois volumes distintos no *array* HP MSA); b) no caso do Ceph, o *cluster* tem 5 nós (VMs), dos quais 4 (os OSTs) participam no "seccionamento" de um RBD em "fatias" que são distribuídas entre os OSTs, sendo que estes residiam todos no mesmo servidor HP - ou seja, os pedidos ao sistema de armazenamento Ceph eram logicamente distribuídos mas fisicamente servidos por um único nó. Assim, o maior *overhead* de acesso ao Ceph, que decorre da utilização do protocolo RADOS quando comparado com o iSCSI usado para servir as imagens btrfs, não é compensado pela distribuição dos nós OST por várias máquinas físicas, com discos independentes.

Um outro aspecto relevante a favor do Ceph é o da disponibilidade e redundância: o *cluster* Ceph foi configurado para tolerar a falha de 1 nó, mantendo o serviço activo, e o btrfs, como sistema de ficheiros local que é, não suporta alta disponibilidade (que poderá ser naturalmente acrescentada por mecanismo de *failover* para outro nó) nem redundância (que poderá ser naturalmente acrescentada com recurso a um *disk array*).

Finalmente, um aspecto muitissimo importante, que é o da escalabilidade da solução: com o Ceph é possível aumentar o desempenho e, simultaneamente, o espaço de armazenamento, por simples adição de mais nós do tipo OST; no caso do btrfs, o aumento de desempenho é muito mais difícil e/ou moroso de conseguir, pois requer aumento da capacidade do nó btrfs, utilização de tecnologias mais rápidas (SSD), partição da informação em vários volumes (com alteração do software e/ou da topologia), etc.

6.2 Trabalho Futuro

Este trabalho dotou a iCBD com suporte multi-*site* (através da replicação) e com armazenamento de *templates* em OBS (Ceph) e uma gestão simples, porque centralizada. No entanto, apesar de neste momento ser um sistema funcional, ainda necessita de algum trabalho adicional antes de poder passar para a fase de produção. Neste momento, o Ceph apenas armazena os *templates* (*golden-images*); para que fosse possível uma solução totalmente suportada em Ceph seria necessário que os restantes ficheiros (pacote de boot e ficheiros de configuração) fossem fornecidos através do Ceph, mas não existiu oportunidade para integrar essas funcionalidades no projecto, pelo que terão de ser implementadas futuramente. Além disto, alguns passos do processo de gestão do sistema de replicação implementado ainda têm de ser realizados manualmente e poderiam de ser automatizados, e.g. depois de terminada a actualização de um *template* na iCBD é necessário iniciar manualmente a replicação da nova actualização.

Num outro plano, o próprio software do Ceph estando sempre em desenvolvimento pela comunidade irá sempre ser alvo de novas melhorias. Numa fase mais tardia do projecto foi testada a utilização de uma nova funcionalidade, introduzida na versão *estável* mais recente (*octopus* 15.2.x), a ceph-immutable-object-cache [34]. Esta permite criar

uma cache que guarde persistentemente conjuntos de dados que sejam do tipo *read-only*, aumentando drasticamente o desempenho do serviço, particularmente se este usar tecnologia SSD para armazenar os dados que são constantemente pedidos pelos clientes como forma de responder mais rapidamente a futuros pedidos. Apesar de não existirem SSDs disponíveis para este projecto, existe uma grande quantidade de RAM que poderia ser usada. Assim testou-se funcionalidade sobre um sistema de ficheiros em memória, tmpfs [35]. Infelizmente, como "é novidade", ainda não existe muita documentação, e tal tornou impossível a resolução dos problemas que surgiram, e a ideia foi abandonada. No entanto, num futuro próximo, esta funcionalidade pode vir a ser essencial para aumentar o desempenho do serviço.

Bibliografia

[1] *Citrix Virtual Apps and Desktops*. URL: https://www.citrix.com/en-gb/products/citrix-virtual-apps-and-desktops/ (acedido em 16/02/2018).

[2] *Windows Virtual Desktops*. URL: https://azure.microsoft.com/en-us/services/virtual-desktop/ (acedido em 16/02/2018).

[3] *VMware Horizon*. URL: https://www.vmware.com/products/horizon.html (acedido em 16/02/2018).

[4] N. R. C. Alves. "Linked clones baseados em funcionalidades de snapshot do sistema de ficheiros". Tese de Mestrado, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, dez. de 2016.

[5] *Amazon Web Services (AWS)*. URL: https://aws.amazon.com (acedido em 16/02/2018).

[6] *Cloud Computing Services | Google Cloud*. URL: https://cloud.google.com (acedido em 16/02/2018).

[7] *IBM Cloud*. URL: https://www.ibm.com/cloud (acedido em 16/02/2018).

[8] *Ceph Homepage*. URL: https://ceph.io.

[9] E. J.P. B. Martins. "Object-Based Storage for support of Linked-Clone Virtual Machine". Tese de Mestrado, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, jan. de 2017.

[10] P. A. Lopes, N. Preguiça, P. Medeiros e M. M. Martins. "iCBD: Uma infraestrutura baseada nos clientes para execução de Desktops Virtuais". Em: *Actas do 8º Congresso Luso-Moçambicano de Engenharia, Maputo/Moçambique, 4-8 Setembro* (2017).

[11] L. M. T. da Silva. "Replication and Caching Systems for the support of VMs stored in File Systems with Snapshots". Tese de Mestrado, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, nov. de 2018.

[12] L. Silva, P. Lopes, N. Preguiça, P. Medeiros, J. Leitão e M. Martins. "Replicação e Caching num Sistema Moderno de Virtualização de Desktops". Em: *Comunicações do INForum 2019*. 2019, pp. 122–133.

[13] O. Agesen, A. Garthwaite, J. Sheldon e P. Subrahmanyam. "The evolution of an x86 virtual machine monitor". Em: *ACM SIGOPS Operating Systems Review* 44.4 (2010), p. 3. ISSN: 01635980.

[14] *ESXi Bare Metal Hypervisor. VMware*. URL: https://www.vmware.com/products/esxi-and-esx.html (acedido em 16/02/2018).

[15] A. Kivity Qumranet, Y. K. Qumranet, D. L. Qumranet, U. L. Qumranet e A. Liguori. "kvm: the Linux Virtual Machine Monitor". Em: (2007).

[16] *Hyper-V overview. Microsoft Docs*. URL: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/hh831531(v=ws.11) (acedido em 16/02/2018).

[17] *XenServer | Open Source Server Virtualization*. URL: https://xenserver.org/ (acedido em 16/02/2018).

[18] *Oracle VM VirtualBox | Oracle Technology Network | Oracle*. URL: http://www.oracle.com/technetwork/server-storage/virtualbox/overview/index.html (acedido em 16/02/2018).

[19] *Windows VM | VMware Workstation Pro*. URL: https://www.vmware.com/products/workstation-pro.html (acedido em 16/02/2018).

[20] G. J. Popek e R. P. Goldberg. "Formal Requirements for Virtualizable Third Generation Architectures". Em: *Communications of the ACM* 17.7 (1974), pp. 412–421.

[21] *Understanding Full Virtualization, Paravirtualization, and Hardware Assisted Virtualization*. URL: https://www.vmware.com/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html.

[22] R. Sandberg. "The Sun Network Filesystem: Design, Implementation and Experience". Em: *Proceedings of the Summer 1986 USENIX Technical Conference and Exhibition* (), pp. 1–16.

[23] S. B. Vaghani. "Virtual machine file system". Em: *ACM SIGOPS Operating Systems Review* 44.4 (2010), p. 57. ISSN: 01635980.

[24] *OpenStack Docs: Welcome to Swift's documentation!* URL: https://docs.openstack.org/swift/latest/ (acedido em 16/02/2018).

[25] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long e C. Maltzahn. "Ceph: A Scalable, High-Performance Distributed File System". Em: *Proceedings of USENIX Symposium on Operating Systems Design and Implementation* (2006), pp. 307–320. ISSN: 1-931971-47-1.

[26] "EMC SCALEIO Basic Architecture Documentation". Em: (2017). URL: https://www.emc.com/collateral/white-papers/h14344-emc-scaleio-basic-architecture.pdf.

[27] *Ceph - Architecture*. URL: https://docs.ceph.com/en/latest/architecture/.

[28] L. Lamport. "Paxos Made Simple". Em: *ACM SIGACT News* 32.4 (2001), pp. 51–58. ISSN: 01635700.

[29] S. Weil, S. Brandt, E. Miller e C. Maltzahn. "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data". Em: *ACM/IEEE SC 2006 Conference (SC'06)* November (2006), pp. 31–31. ISSN: 0769527000.

[30] *Welcome to Ceph — Ceph Documentation*. URL: http://docs.ceph.com/docs/master/.

[31] *The CentOS Project*. URL: https://www.centos.org (acedido em 16/02/2018).

[32] *Flapping OSDs - Ceph Documentation*. URL: https://docs.ceph.com/en/latest/rados/troubleshooting/troubleshooting-osd/#flapping-osds (acedido em 10/01/2021).

[33] *SUSE Linux Enterprise 15 is Generally Available | SUSE Communities*. URL: https://www.centos.org (acedido em 16/02/2018).

[34] *RBD Persistent Read-Only Cache*. URL: https://docs.ceph.com/en/latest/rbd/rbd-persistent-read-only-cache/ (acedido em 12/01/2021).

[35] *Arch Linux Wiki - tmpfs*. URL: https://wiki.archlinux.org/index.php/tmpfs (acedido em 10/01/2021).