



Luís Miguel Teixeira da Silva

Bachelor of Science

Replication and Caching Systems for the support of VMs stored in File Systems with Snapshots

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser: Nuno Preguiça, Associate Professor,
NOVA University of Lisbon

Co-adviser: Pedro Medeiros, Associate Professor,
NOVA University of Lisbon

Examination Committee

Chairperson: Name of the committee chairperson

Raporteurs: Name of a rapporteur

Name of another rapporteur

Members: Another member of the committee

Yet another member of the committee



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2018

Replication and Caching Systems for the support of VMs stored in File Systems with Snapshots

Copyright © Luís Miguel Teixeira da Silva, Faculty of Sciences and Technology, NOVA University Lisbon.

The Faculty of Sciences and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

Lorem ipsum.

ACKNOWLEDGEMENTS

The work presented in this document would never see the light of day if not for the collaboration of several people to whom I wish to manifest my profound gratitude and recognition.

Prof. Nuno Preguiça, Prof. Pedro Medeiros, Prof. Paulo Lopes

I also would like to acknowledge the following institutions for their hosting and financial support: *Departamento de Informática and Faculdade de Ciências e Tecnologia of the Universidade NOVA de Lisboa* (DI-FCT NOVA); the *NOVA Laboratory of Computer Science and Informatics* (NOVA LINCS) in particular the Computer Systems group; *SolidNetworks – Business Consulting, LDA of the Reditus S.A. Group*; and the funding provided through the *COMPETE2020 / PORTUGAL2020* program for the *iCBD* project (POCI-01-0247-FEDER-011467).

ABSTRACT

Over the span of a few years, there were fundamental changes in the way computing power is handled. The heightening of virtualisation changed the infrastructure model of a *data centre* and the way physical computers are managed. This shift is the result of allowing for fast deployment of [Virtual Machines \(VMs\)](#) in a high consolidation ratio environment and with minimal need for management.

New approaches to virtualisation techniques are being developed at a never seen rate. Which leads to an exciting and vibrating ecosystem of platforms and services seeing the light of day. We see big industry players engaging in such problems as *Desktop Virtualisation* with moderate success, but completely ignoring the already present computation power in their clients, instead, opting for a costly solution of acquiring powerful new machines and software. There is still space for improvement and the development of technologies that take advantage of the onsite computation capabilities with minimum effort on the configuration side.

This thesis focuses on the development of mechanisms for the replication and caching of VM images stored in a conventional file system with the ability to perform snapshots. There are some particular items to address: like the solution needs to follow an entirely distributed architecture and fully integrate with a parallel implemented client-based [Virtual Desktop Infrastructure \(VDI\)](#) platform; needs to work with very large read-only files some of them resulting from the creation of snapshots while maintaining some versioning features. This work will also explore the challenges and advantages of deploying such system in a high throughput network, maintaining high availability and scalability properties while supporting a broad set of clients efficiently.

RESUMO

Nos últimos anos, tem-se assistido a mudanças fundamentais na forma como a capacidade computacional é gerida. Com o grande aumento da utilização da virtualização a forma como são geridas as máquinas físicas e os modelos de infraestruturas de um centro de dados sofreram grandes alterações. Esta mudança é o resultado de uma procura por uma forma de disponibilizar rapidamente uma VM num ambiente altamente consolidado e com a mínima necessidade de intervenção para a sua gestão.

Estão a ser desenvolvidas novas abordagens às técnicas de virtualização a um ritmo nunca visto. O que leva à existência de um ecossistema altamente volátil com novas plataformas e serviços a serem criados a todo o momento. É possível apreciar a entrega de grandes empresas da indústria das tecnologias de informação a problemas como a virtualização de desktops com algum sucesso, mas ignorando completamente o poder de computação que já está presente nos seus clientes. Optando ao em vez, por uma via de alto custo, adquirindo máquinas poderosas e vários softwares. Existe ainda espaço para melhores soluções e para o desenvolvimento de tecnologias que façam uso das capacidades de computação já se encontrem presentes com o mínimo de esforço na sua configuração.

Esta tese foca-se no desenvolvimento de mecanismos de replicação e caching para imagens de máquinas virtuais armazenadas num sistema de ficheiros convencional com a funcionalidade de fazer snapshots. Existem alguns pontos em particular a endereçar: a solução tem que seguir uma arquitectura distribuída e ser totalmente integrada numa solução client-based VDI; tem que funcionar com enormes ficheiros apenas de leitura alguns deles resultantes da criação de snapshots mantendo a característica de manutenção de versões. Este trabalho também incide na exploração dos benefícios de utilizar tal sistema numa rede com uma alta taxa de transferência de dados, em quanto mantém propriedades de alta disponibilidade e escalabilidade suportando um largo conjunto de clientes de forma eficiente.

CONTENTS

List of Figures	xv
List of Tables	xvii
Listings	xix
Acronyms	xxi
1 Introduction	1
1.1 Context	1
1.2 Motivation	2
1.3 Project Presentation	3
1.3.1 iCBD Project	3
1.3.2 Previous Work	4
1.4 Project Contributions	4
1.4.1 Main Expected Contributions	5
1.5 Document Structure	5
2 Research Context	7
2.1 Virtualisation	7
2.1.1 Hypervisors	8
2.1.2 Virtual Desktop Infrastructure	10
2.1.3 Virtual Machine Image Storage	13
2.2 Storage	14
2.2.1 File Systems	14
2.2.2 Snapshots	16
2.3 Caching	16
2.4 Replication	17
3 iCBD - Infrastructure for Client-Based Desktop	19
3.1 The Concept	19
3.1.1 Client-based vs. Server-based VDI	20
3.2 The Architecture	20
3.2.1 Boot Layer	20

CONTENTS

3.2.2	Client Layer	20
3.2.3	Storage Layer	20
4	Replication and Caching - The Problem	21
4.1	The objective	21
4.2	The architecture	21
5	Implementation of iCBD-Replication and Cache Server	23
5.1	Implementation of a Replication Module	23
5.2	Functional Testing	23
5.3	Building a Cache Server	23
6	Conclusions & Future Work	25
6.1	Conclusions	25
6.2	Future Work	25
	Bibliography	27

LIST OF FIGURES

2.1	Virtualization architecture with type 1 and type 2 hypervisors	9
2.2	An exemple of a Virtual Desktop Infrastructure, adapted from AppDS [7] . .	11

LIST OF TABLES

LISTINGS

ACRONYMS

DaaS	Desktop as a Service.
IaaS	Infrastructure as a Service.
iCBD	Infrastructure for Client-Based Desktop.
KVM	Kernel-based Virtual Machine.
OS	Operating System.
VDI	Virtual Desktop Infrastructure.
VM	Virtual Machine.
VMM	Virtual Machine Monitor.

INTRODUCTION

1.1 Context

The concept of virtualization, despite all the recent discussion, isn't new. In fact, this technology has been around since the 1960s [8], but not until the development of virtualization technologies for the x86 architecture [1] and the introduction of *Intel VT* [26] and *AMD SVM* [5] in the 2000s entered the mainstream as the go-to technology solution for server deployment across many production environments.

With efficient techniques that take advantage of all available resources, and a lowering price point on hardware, an opportunity for the advance of new application models and a revamp in the supporting infrastructure was generated.

However, companies realised that the cost to run a fully fledged *data centre* in-house is unreasonable and a cumbersome task. Not only taking into account the cost of the machines, but factoring in the many requirements like the cooling systems that take care of the heat generated by the running machines, physical security to protect the rooms, fire suppressing systems in case of emergency, people to maintain the infrastructure, all added, result in considerable costs on a monthly basis. Adding to this, the demand for instantaneous access to information and the extensive resources needed to store it does not stop growing.

This fact created an opening for a [Infrastructure as a Service \(IaaS\)](#) [19] model, outsourcing all the responsibilities of storing the data and providing the needed computation resources from third parties, which are experts in maintaining huge data centres and even provide all this in various geographic regions.

With major industry players following this trend, supporting more and more types of services and with an increasing number of customers joining this model, new ways to store the growing number of files have emerged. New file systems with a focus on reliability,

consistency, performance, scalability, all in a distributed architecture are essential to a broad range of applications presenting a myriad of workloads.

1.2 Motivation

Virtualization is the pillar technology that allowed for the widespread of the IaaS cloud providers in a economy of scale model. These cloud providers, such as Amazon AWS [4], Microsoft Azure [20] and Google Cloud Platform [14], manage thousands of physical machines all over the globe, with the majority of the infrastructure being multi-tenant oriented.

The sheer magnitude of those numbers leads to an obvious problem. How to store efficiently all this data? Not only there is the need to store client generated data but also manage all the demands of the infrastructure and the many services offered. One approach taken by these companies was the development of their own storage solutions. For instance, Google uses BigTable distributed storage system [9], to store product specific data, and then serve it to users. This system relies on the Google File System underneath to provide a robust solution to store logs and data files, designed to be reliable, scalable and fault tolerance.

One characteristic in particular that stands out and is present in many of today's systems is the use of snapshots with copy-on-write techniques. The adoption of such methods allows for quick copy operations of large data sets but saving resources. At the same time it provides high-availability with read-only copies of the data always ready to use and allowing applications to continue execution of write operations simultaneously. All the above-mentioned properties joined with others such as replication and data distribution, to comprise the fundamentals to what is needed to run a highly distributed and scalable file system. For instance, the duplication of records across multiple machines, not only serves as a security net in case of a misfortune event avoiding having a single point of failure but can also be used to maximise availability and take advantage of network bandwidth.

One of these newer systems that have a significant adoption by the Linux community is the BTRFS [27]. At the start, this file system already adopts an efficient system of snapshots and it has as a primary design principle to maintain an excellent performance in a wide set of conditions. The combination of this file system with replication and partitioning techniques opens the way to a solution that serves the needs of an up to date storage system, consequently having the possibility of being easily integrated into an existing platform, serving a vast number of clients and presenting outstanding performance.

1.3 Project Presentation

This dissertation work is performed in the context of a larger project with the name [Infrastructure for Client-Based Desktop \(iCBD\)](#) [17], under development at Reditus S.A. in collaboration with DI - FCT/NOVA. The primary objective is to improve in a known model, the client-based Virtual Desktop Infrastructure, developing an infrastructure to support the execution, in a non-intrusive way, of virtualized desktops in conventional workstations.

1.3.1 iCBD Project

There are some leading-edge aspects of the [iCBD](#) project which sets it apart from other solutions that already exist. Such the adoption of a diskless paradigm with a remote boot, the way virtual machine images are stored in the platform and the support for a virtualized or native execution on any workstation, depending on the user's choice. [16]

The Remote boot support is offered by HTTP, TFTP, and DHCP servers, and in turn, the image repository servers manage the storage of the VMs templates and the production of instances based on them. To address the process of communication between workstations and the platform it is used the HTTP protocol, providing flexibility and efficiency in the communication of the messages. [2, 16, 18]

It is also interesting to briefly discuss some of the primary objectives of the project, being:

- Offer a work environment and experience of use so close to the traditional one, that there is no disruption for the users when they begin to use this platform.
- Enable centralized management of the entire infrastructure including servers in their multiple roles, storage and network devices from a single point.
- Complete decoupling between users and workstations in order to promote mobility.
- Support the disconnected operation of mobile workstations.

With all the above in account, there is a clear separation from other solutions previously and currently available. As far as we know, no other solution is so comprehensive in the use of the resources offered by workstations whether they are PCs, laptops or similar devices.

1.3.2 Previous Work

There have previously been two dissertations involved in this project. That work has centred in the creation of the instances of virtual machines, more specifically in the creation supported by native snapshot mechanisms of the file system where the templates are stored. This way instead of using the hypervisor itself as a method to provision full or thin clones the work is done by the file system snapshot system.

As is happening now, the two theses have followed two different paths in an attempt to determine which file system best suits these objectives. Being that one used a local file system, the BTRFS, and the other followed the object-based storage path, adopting the CephFS.

1.4 Project Contributions

This work, as a part of a bigger project and building on previous contributions, has as premise a couple of existing technologies in the file systems field to create a replicated and distributed environment capable of storing large files consisting mainly of VMs templates and golden images. This work not only focuses on storage management aspects, as also attends the need of being integrated into a larger infrastructure and coexist with a wide variety of other systems.

1.4.1 Main Expected Contributions

The main expected contributions are:

- The study, develop, and evaluate an implementation of a distributed and replicated BTRFS file system for VM storage.
- Implement a server-side caching solution in order to increase availability, improve response time, and enable better management of resources.
- Integrate the solutions described above with the work previously developed and the existing infrastructure
- And finally, carry out a series of tests that lead to a meaningful conclusion and that provide help in the design of the remaining platform.

A detailed view of the planning can be found in **Chapter ??**.

1.5 Document Structure

The remnant document is structured as follows:

- *Chapter ?? Related Work* - This section presents existing technologies and theoretical approaches which were the target of study, such as, storage systems and several of its features, as well as several intrinsic characteristics of virtualization techniques.
- *Chapter 3 Proposed Work* - In this chapter, there is a presentation of the work plan for the elaboration of this dissertation. Giving also an overview of the solution to develop on the duration of this thesis.

RESEARCH CONTEXT

The focal point of this dissertation is the challenges of implementing a distributed system based on a file system, that can store *VMs* images while leveraging the benefits of snapshots and caching techniques. Moreover, the work done should integrate smoothly into a broader infrastructure illustrated in detail in Section 3. In this chapter, we bestow a survey of core concepts directly associated with the thesis and compliment with some analysis on the state-of-art in the relevant fields.

The organisation of this chapter is as follows:

Section 2.1 overviews virtualisation as a core concept ...

Section 2.2 studies the principal characteristics of a file system, with emphasis on snapshot techniques ...

Section 2.3 talks about ...

Section 2.4 expands on ...

2.1 Virtualisation

Most of today's machines have such a level of performance that allows the simultaneous execution of multiple applications and the sharing of these resources by several users. In this sense, it is natural to have a line of thought in which all available resources are taken advantage of efficiently.

Virtualisation is a technique that allows for the abstraction of the hardware layer and provides the ability to run multiple workloads on a shared set of resources. Nowadays, virtualisation is an integral part of many *IT* sectors with applications ranging from

hardware-level virtualisation, operating system-level virtualisation, and high-level language virtual machines.

A Virtual Machine by design is an efficient, isolated duplicate of a real machine [23], in that order, it was the capacity to virtualise all of the hardware resources, including processors, memory, storage, and network connectivity.

For the effort of managing the VMs, there is a need for a software layer that has specific characteristics. One of them is the capability to provide an environment in which VMs conduct operations, acting both as a controller and a translator between the VM and the hardware for all *IO* operations. This piece of software is known as a [Virtual Machine Monitor \(VMM\)](#).

In today's architectures, a modern term has been coined, the *Hypervisor*. It is common to mix both concepts (*VMMs* and *Hypervisors*), as being the same, but in fact, there are some details that make them not synonymous. [1]

2.1.1 Hypervisors

The most important aspect of running a VM is that it must provide the illusion of being a real machine, allowing to boot and install any Operating System (OS). It is the VMM which has that task and should do it efficiently at the same time providing this three properties [23]:

Fidelity: a program should behave on a VM the same way or in much the same way as if it were running on a physical machine.

Performance: much of the instructions in the virtual machine should be executed directly by the real processor without intervention by the hypervisor.

Isolation: the VMM must have complete control over the resources.

A hypervisor is the blend of an Operating System and a Virtual Machine Monitor. It can make use of a run-of-the-mill OS, such one of the several flavours of *Linux* and the *Microsoft Windows*, or a bare metal purpose-built one, such the *VMware ESX/ESXi* family.

Concerning the execution of a VM, the hypervisor kernel spins up a VMM, which holds the responsibility of virtualising the *x86* architecture and provide the platform where the VM will lie. This way, since the VM executes on top of the VMM, there is a layer of separation between the guest VM and the host hypervisor kernel, with the communications within being made through the VMM. This feature confers a necessary degree of isolation among the system. With the host kernel taking care of host-centric tasks as *CPU* and memory scheduling, and the network and storage *I/O* stacks, and the VMM assumes responsibility to provide those resources to the VM.

A hypervisor can be classified into two different types [6], symbolising two different design strategies to virtualisation, as shown in Figure 2.1:



Figure 2.1: Virtualization architecture with type 1 and type 2 hypervisors

Type 1 hypervisor: Sometimes referred as a bare-metal hypervisor, since there is no need to rely on a host operating system, as they run directly on the hardware. Moreover, this software is the only program executed by the CPU in the most privileged mode. As there isn't a go-between itself and the resources, being able to communicate directly to the hardware, this type of hypervisor presents as a more efficient solution than the Type 2 hypervisor.

In addition to the improved performance provided by the partitioning of devices between the several guest VMs, this kind of architecture provides the benefit of supporting the execution of real-time OSs. The low-level nature of these hypervisors with the broad access to the hardware is proven useful for use-cases that need to deploy a multiplicity of operating systems.

Recognizing all the facts above, we can point that there are also some disadvantages. Any drivers needed to support different hardware platforms must be covered by the hypervisor package. As do drivers for devices that have the capability of being shared amongst guests.

Furthermore, considering that Type 1 hypervisors do not have an underlying OS, the complexity of installing and configuring this type of solution increases.

Type 2 hypervisor: This second variant of the hypervisor model, relies on an already installed operating system and acts very similarly to any conventional process. Here,

the hypervisor layer is a union of a host operating system with specialised virtualisation software that will manage the guest VM. In this case, the hypervisor makes use of the services provided by the OS, which leads to a more significant memory footprint when compared to Type 1 but are integrated seamlessly with the remainder of the system. An excellent illustration of this kind of paradigm is [Kernel-based Virtual Machine \(KVM\)](#) [15] and VMware Workstation/Fusion [1].

In this architecture, the host operating system retains ownership of the physical components, with each VM having access to a confined subset of those devices, and the virtual machine monitor providing an environment that emulates the actual hardware per VM. One advantage is quickness of installation and configuration since the installation process for most of these type of hypervisors only requires the execution of an elemental installer, delivering the instantaneous ability to run a multitude of different operating systems.

All the above culminates in some advantages. Type 2 hypervisors are regularly deployed for the use of software testing, dismissing the requirement for a dedicated test machine. Also, provide support for running multiple OSs on the same physical machine, which can be valuable for those who rely on multiple applications written for a particular or legacy OS.

Either way, the challenge lays in the fact that the hypervisor needs to execute the guests OS instructions in a safe manner and at the same time provide possible different machine configurations to each of them. These characteristics, such as the number and architecture of virtual CPUs (vCPU), the amount and type of memory available (vRAM), the allowed space to store files (vDisk), and so on, are user configurable but is the hypervisor that is tasked to do all the management and load balancing. The settings of these components reside in a VM configuration file. In the case of VMware hypervisors, the file employs the `.vmx` extension.[24, 36] In a KVM environment, the configuration is in a `.xml` file. [10]

With a virtualised infrastructure there is an opening for a substantial reduction in the number of servers. Which in turn diminishes the setup time, as those VMs are, in a broad manner, created with the resource to cloning techniques. Software updates can be hugely simplified and made available to all users at once. Even availability is improved since it is an easy task to launch a new VM from a template and migrate all the services that were being made reachable by one that suffered a failure.

2.1.2 Virtual Desktop Infrastructure

It is common to find in a typical midsize corporate infrastructure hundreds of servers and thousands of workstations. All in a diverse ecosystem counting with many hardware configurations, different OSs and applications needs. Probably even supporting several versions of the same software is required for the day to day operations.



Figure 2.2: An exemple of a Virtual Desktop Infrastructure, adapted from AppDS [7]

Organizations debate themselves daily with the traditional problem of installing software in local workstations disks one-by-one (even if employing an automated process). This task tends to be daunting as a company escalates in size and leads to some other predicaments:

- A Systems Administrator and IT Staff burden with significant infrastructure administration responsibilities.
- A delay on the installation or reinstallation of new software and recovery from breakdowns.
- Installation processes may consume much of the available bandwidth in a network, so if this job is to be executed simultaneously on several workstations, it tends to be scheduled off work hours to produce the shortest disturbances.
- Periodical software updates (such Microsoft's famous Patch Tuesdays [22]) are ordinarily unleashed in the morning first boot of a workstation, which can bloat the traffic and render useless the workstation for the remainder of the update period.
- If an update proves to be undesirable, by introducing some unexpected behaviour, it is quite difficult to reverse this situation, and may even demand a new configuration infrastructure-wise.

One solution to the unpleasant situations above is to minimise the footprint of installed software having in account the locality of the data. It is possible to imagine a dimension where all the software required to execute a workstation (Operating System and applications) can be conjugated in a virtual machine. This mechanism allows for the

virtualisation of a workstation that can be performed either locally on a typical PC / Laptop, or on a server. The implementation with more relevance and with more expression at the moment is the [VDI](#).

The concept encompasses a series of techniques, providing on-demand availability of desktops, in which, all computing is performed employing virtual machines [32]. Typically this solution offers a centralised architecture, where the user's environment resides on a server in a data centre, as shown on Figure 2.2. However, other components are required, such as storage for the users and VMs data and a network capable of moving large data blocks quickly, all in a perspective where from the user's viewpoint there can't be any apparent difference between a virtual desktop and a local installation.

There are two antagonistic approaches to the architecture, one focused on the server-side and the other on the client-side:

Server-based VDI This is the most common approach, in which the VM runs remotely on a server using a hypervisor. Featuring such benefit, as the fact that only a low-performance thin client with support for a protocol such as Remote Desktop Protocol (RDP) [25] or the Remote Framebuffer Protocol (RFB) [31] is required to interact with the virtual desktop.

The downside involves the costs necessary to maintain the service. A powerful support infrastructure is needed (computing, storage, networking and power). With a need in some use cases to add high-end graphics processors to satisfy the workflow of customers using multimedia tools. Moreover, the computational strength of the hardware present in the clients is not harnessed.

There are plenty of commercial solutions that use this principle, with the three largest players being VMware's Horizon platform [34], XenDesktop from Citrix [37] and Microsoft with Microsoft Remote Desktop [21].

Client-based VDI There is another case, where the VM is executed directly on the client machine. Here the local hardware is fully handled by one hypervisor, giving some slack to the infrastructure servers that just need to perform management and storage tasks on the platform.

Although this approach presents itself as significantly more cost restrained, there isn't a notable adoption by software houses in developing products in this family. One of the possible reasons may be that previously existing solutions, such as Citrix's XenClient [37], needed to erase the user machine disk completely [11].

As discussed earlier virtualization has paved the way for a class of services called IaaS. That leads to an apange of modern times in which there is a desire to link all services to the cloud; so it is natural that an idea came to take the VDI paradigm to this medium. Thus giving rise to the emergence of Desktop as a Service (DaaS), with companies such as Amazon, VMware and Citrix entering the market.

This model has a real potential for cost minimization since there is no need for concern in regards to the maintenance and acquisition of the infrastructure. Since the organisation only has to make available the workstations and a way of establishing a connection to the service. Although, there is always a dependence on the network conditions such as large latencies and low bandwidth.

2.1.3 Virtual Machine Image Storage

The data storage is the focal point to address in this work. Therefore, it is important to understand how a virtual machine is composed and how is translated to a representation in a storage device.

The representation of a machine's settings and state more the information about snapshots comprise to a set of metadata. On the other hand, the Operating System, applications, logs and snapshots constitute the remaining data. Such data and metadata must be saved in a storage system, regardless of type, but is typically defined by a set of files.

Given the architecture presented by VMware software [36], the main files required for the operation of a VM are:

- The VM configuration file - The `.vmx` file holds the fundamental configuration options, describing every aspect of the VM.
- The virtual disk files - Embodying multiple `.vmdk`, which stores the contents of the virtual machine's hard disk drive.
- The file that stores the BIOS - The `.nvram` file stores the state of the virtual machine's BIOS.
- The suspended state file - The `.vmss` saves contains the state of a suspended virtual machine.

- Log files - A collection of .log files is created to log information about the virtual machine and often handled for troubleshooting purposes.

In addition to the records described above, there may be some more related to the use of snapshots. The implementation of snapshots can be described as follows: first, the state of the resource is stored in the form of an immutable and persistent object, and second all modifications that transform the state of the resource are saved in a different object. To save this objects the .vmsn extension is employed. The snapshotting technique is discussed in a more comprehensive sense in the Section [2.2.2](#).

2.2 Storage

As stated in previous sections, the main problem to be addressed in this work is the storage concerning virtual machines. That could be either images, snapshots, files or data structures that are needed to support the execution of a VM.

When applied to the VDI concept some demands appear in the form of a specific care at planning the storage system architecture, as well as the supporting infrastructure: the hardware picked, network topology, protocols used, and software implemented.

At the end of the day, the idea is to present a solution that offers an appropriate cost to performance ratio, and that with little effort can scale when the need emerges.

2.2.1 File Systems

The traditional and perhaps most common way of storing files and, in turn, VMs is the use of file systems. This kind of system is used to manage the way information is stored and accessed on storage devices. A file system can be divided into three broad layers, from a top-down perspective we have:

- The **Application Layer** is responsible for mediating the interaction with user's applications, providing an API for file operations. This layer gives file and directory access matching external names adopted by the user to the internal identifiers of the files. Also, manages the metadata necessary to identify each file in the appropriate organisational format.
- Then the **Logic Layer** is engaged in creating a hardware abstraction through the creation of logical volumes resulting from the use of partitions, RAID volumes, LUNs, among others.
- The last one is the **Physical Layer**. This layer is in charge with the physical operations of the storage device, typically a disk. Handling the placement of blocks in specific locations, buffering and memory management.

There are many different types of file systems, each one boasting unique features, which can range from security aspects, a regard for scalability or even the structure followed to manage storage space.

Local file systems: A local filesystem can establish and destroy directories, files can be written and read, both can move from place to place in the hierarchy but everything contained within a single computing node. Good performance can be improved in certain ways, incorporating caching techniques, read ahead, and carefully placing the blocks of the same file close to each other, although scalability will always be reduced. There are too many file systems of this genre to be here listed. Nevertheless, some of the most renowned may be mentioned. As the industry-standard File Allocation Table (FAT), the New Technology File System (NTFS) from Microsoft, the Apple's Hierarchical File System Plus (HFS+) also called Mac OS Extended and the B-tree file system (BTRFS) initially designed by Oracle.

Distributed file system: A distributed file system enables access to remote files using the same interfaces and semantics as local files, allowing users to access files from any computer on a network. Distributed file systems are being massively employed in today's model of computing. They offer state-of-the-art implementations that are highly scalable, provide great performance across all kinds of network topologies and recover from failures. Because these file systems carry a level of complexity considerably higher than a local file system, there is a need to define various requirements such being transparent in many forms (access, location, mobility, performance, scaling). As well as, handle file replication, offer consistency and provide some sort of access-control mechanisms. All of these requirements are declared and discussed in more detail in the book "Distributed Systems: Concepts and Design" by George Coulouris et al. [12] We can give as example of file systems the well-known Network File System (NFS) [28] originally developed by Sun Microsystems, and the notable Andrew File System (AFS) [29] developed at Carnegie Mellon University.

In this work, the snapshot functionality of the file system itself is a valuable asset. This technique is present in some of the most recently designed file systems, such as the BTRFS. It has already been mentioned that previous work has been done to use the file system snapshot features as a base feature. This way the creation of linked-clones handled by the file system capabilities as an alternative to linked-clones created by virtualization software itself.

There are numerous types of additional file systems not mentioned since they are not in the domain of this work. Still, it is important to note the existence of an architecture that is not similar to the traditional file hierarchy adopted in file systems, which is the object-based storage.

This structure, as opposed to the ones presented above, manages data into evenly sized blocks within sectors of the physical disk. It is possible to verify that it has gained

traction leading to the advent of the concept of cloud storage. There are numerous implementations of this architecture, whether in small local deployments or large-scale data centres supporting hundreds of petabytes of data. This type of file system is being studied in the context of a parallel thesis but inserted in the same project already presented.

It is worthwhile to enumerate some examples such as CephFS [35], OpenStack Swift [30], and in a IaaS flavour the Amazon S3 [3] and Google Cloud Storage [13].

2.2.2 Snapshots

TO DO - Expand

2.3 Caching

A cache can be defined as a store of recently used data objects that is nearby one client or a particular set of clients than the objects themselves. The inner works of one of these systems are rather simple. When a new object is obtained from a server, it is added to the local cache, replacing some existing objects if needed. That way when an object is requested by a client, the caching service first checks the cache and supplies the object from there if an up-to-date copy is available. If not, an up-to-date copy is fetched, then served to the client and stored in the cache.

Caching often plays a crucial role in the performance and scalability of a file system and is used extensively in practice.

Caches may be found beside each client or they may be located on a server that can be shared by numerous clients.

Server-side Cache: Server side caching is when the caching data occur on the server.

There is no right way to the approach of caching data; it can be cached anywhere and at any point on the server assuming it makes sense. It is common to cache frequently used data from a DataBase to prevent connecting to the DB every time some data is requested. In a web context, it is common to cache entire pages or page fragments so that there is no need to generate a web page every single time a visitor arrives.

Client-side Cache: Maintaining the analogy to the Web environment, caches are also used on the client side. For instances, Web browsers keep a cache of lately visited web pages and other web resources in the client's local file system. Then when the time comes to serve a page that is stored in the cache, a special HTTP request is used to check, with the corresponding server, if the cached page is up-to-date. In a positive response the page is simply displayed from the cache, if not, the client just needs to make a normal request.

2.4 Replication

At the storage level, replication is focused on a block of binary data. Replication may be done either on block devices or at the file-system level. In both cases, replication is dealing with unstructured binary data. The variety of technologies for storage-level replication is very extensive, from commodity RAID arrays to network file system. File-based replication works at a logical level of the storage system rather than replicating at the storage block level. There are multiple different methods of performing this. And, unlike with storage-level replication, these solutions almost exclusively rely on software.

Replication is a key technology for providing high availability and fault tolerance in distributed systems. Nowadays, high availability is of increasing interest with the current tendency towards mobile computing and consequently the appearance of disconnected operation. Fault tolerance is an enduring concern for those who provide services in critical and other important systems.

There are several arguments for which replication techniques are widely adopted; these three are of significant importance:

Performance improvement: Performance improvement: Replication of immutable data is a trivial subject, is nothing more than a copy of data from one place to another. This increases performance, sharing the workload with more machines with little cost within the infrastructure.

Increased availability: Replication presents itself as a technique for automatically keeping the availability of data despite server failures. If data is replicated in additional servers, then clients may be able to access that data from the servers that didn't experience a failure. Another factors that must be taken into account are network partitions and disconnected operation.

Fault tolerance: There is the need to maintain the correctness guarantees of the data in the appearance of failures, which may occur at any time.

iCBD - INFRASTRUCTURE FOR CLIENT-BASED DESKTOP

The acronym **iCBD** stands for Infrastructure for Client-Based (Virtual) Desktop (Computing). Is a platform being developed by an R&D partnership between *NOVA LINC*S, the Computer Science research unit hosted at the *Departamento de Informática of Faculdade de Ciências e Tecnologia of Universidade NOVA de Lisboa* (DI-FCT NOVA) and *SolidNetworks – Business Consulting, LDA* part of the *Reditus S.A.* group.

Where the primary goal is to achieve a particular kind of **VDI** infrastructure, a client based VDI, where client's computations are performed directly on the client hardware opposed to on big and expensive servers.

This chapter will address the central concepts and associated technologies encompassed in this project, particularly:

Section 3.1 overviews the core concepts of the project and particularly note the limitations and peculiarities of current implementations in contrast with the chosen approach.

Section 3.2 studies the principal architectural components of the platform, with emphasis on the different layers and how they act together to serve the end-user.

3.1 The Concept

The iCBD as a project pretends to investigate and develop an architecture that leads to the birth of a platform that can operate desktop virtualisation (**VDI**). In a sense, the goal is similar to a client-based VDI, but with the distinction of maintaining all the benefits of both client-based and server-based VDI. Additionally, it should present the power

of working as a Cloud [Desktop as a Service \(DaaS\)](#) without any of the bad traits of the approaches as mentioned earlier.

The aim is to preserve the convenience and simplicity of a fully centralised management platform for Linux and Windows desktops, instantiating those in each physical workstation from virtual machine templates (VMs) kept in repositories. We will talk more about this subject in section [3.2](#)

To summarise the platform should be able to:

- Tuning to a wide range of server configurations, without prejudice to the defined architecture.
- Minimize disruption in the use of workstations for end-users. Offering a work environment and experience of use so close to the traditional one that they should not be able to tell from a standard local installation of an [Operating System \(OS\)](#) to the use of this platform.
- Simplify installation, maintenance and platform management tasks for the entire infrastructure, including servers in their multiple roles, storage and network devices from a single point.
- Allow for a highly competitive per workstation cost.
- Maintain an inter-site solution; such a geographically disperse multi-office structure.

3.1.1 Client-based vs. Server-based VDI

3.2 The Architecture

3.2.1 Boot Layer

3.2.2 Client Layer

3.2.3 Storage Layer

REPLICATION AND CACHING - THE PROBLEM

4.1 The objective

4.2 The architecture

IMPLEMENTATION OF iCBD-REPLICATION AND CACHE SERVER

5.1 Implementation of a Replication Module

5.2 Functional Testing

<https://stackoverflow.com/questions/5357601/whats-the-difference-between-unit-tests-and-integration-tests>

5.3 Building a Cache Server

CONCLUSIONS & FUTURE WORK

6.1 Conclusions

6.2 Future Work

BIBLIOGRAPHY

- [1] O. Agesen, A. Garthwaite, J. Sheldon, and P. Subrahmanyam. “The Evolution of an x86 Virtual Machine Monitor.” In: *SIGOPS Oper. Syst. Rev.* 44.4 (Dec. 2010), pp. 3–18.
- [2] N. Alves. “Linked clones baseados em funcionalidades de snapshot do sistema de ficheiros.” Master’s thesis. Universidade NOVA de Lisboa, 2016.
- [3] Amazon Web Services. *Amazon Simple Storage Service (S3)*. 2017. URL: <https://aws.amazon.com/s3/> (visited on 02/10/2017).
- [4] Amazon Web Services (AWS) - Cloud Computing Services. 2017. URL: <https://aws.amazon.com/> (visited on 02/05/2017).
- [5] AMD. *AMD-V Nested Paging*. Tech. rep. 2008, pp. 1–19. URL: <http://developer.amd.com/wordpress/media/2012/10/NPT-WP-1%201-final-TM.pdf>.
- [6] A. Aneja. *Designing Embedded Virtualized Intel® Architecture Platforms with the right Embedded Hypervisor*. Tech. rep., pp. 1–14. URL: <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-embedded-virtualized-hypervisor-paper.pdf>.
- [7] *AppDelivery Solutions - Desktop Virtualization*. 2017. URL: <https://appds.eu/Home/DesktopVirt> (visited on 02/05/2017).
- [8] J. P. Buzen and U. O. Gagliardi. “The Evolution of Virtual Machine Architecture.” In: *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition* (1973), pp. 291–299.
- [9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. “Bigtable: A distributed storage system for structured data.” In: *7th Symposium on Operating Systems Design and Implementation (OSDI ’06), November 6-8, Seattle, WA, USA* (2006), pp. 205–218.
- [10] H. Chirammal, P. Mukhedkar, and A. Vettathu. *Mastering KVM Virtualization*. Packt Publishing, 2016. ISBN: 9781784396916.
- [11] *Citrix Bids Adieu to XenClient*. 2015. URL: <http://vmblog.com/archive/2015/09/24/citrix-bids-adieu-to-xenclient.aspx> (visited on 02/07/2017).

- [12] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *Distributed Systems: Concepts and Design*. 5th. USA: Addison-Wesley Publishing Company, 2011. ISBN: 0132143011, 9780132143011.
- [13] Google. *Google Cloud Platform - Cloud Storage*. 2017. URL: <https://cloud.google.com/storage/> (visited on 02/10/2017).
- [14] Google Cloud Platform. 2017. URL: <https://cloud.google.com/> (visited on 02/05/2017).
- [15] A. Kivity, U. Lublin, A. Liguori, Y. Kamay, and D. Laor. “kvm: the Linux virtual machine monitor.” In: *Proceedings of the Linux Symposium 1* (2007), pp. 225–230. URL: <https://www.kernel.org/doc/mirror/ols2007v1.pdf{\#}page=225>.
- [16] P. Lopes. *Proposta de Candidatura ao programa P2020*. Tech. rep. DI-FCT/NOVA, Reditus S.A, 2015, pp. 1–26.
- [17] P. Lopes, N. Preguiça, P. Medeiros, and M. Martins. “iCBD: Uma Infraestrutura Baseada nos Clientes para Execução de Desktops Virtuais.” In: *Proceedings CLME2017/VCEM 8º Congresso Luso-Moçambicano de Engenharia / V Congresso de Engenharia de Moçambique* (2017), pp. 13–18.
- [18] E. Martins. “Object-Base Storage for the support of Linked-Clone Virtual Machines.” Master’s thesis. Universidade NOVA de Lisboa, 2016.
- [19] P. Mell and T. Grance. “The NIST definition of Cloud Computing.” In: *NIST Special Publication 145* (2011), p. 7.
- [20] *Microsoft Cloud Computing Platform and Services*. 2017. URL: <https://azure.microsoft.com/> (visited on 02/05/2017).
- [21] *Microsoft Remote Desktop Services (RDS) Explained*. 2010. URL: <https://technet.microsoft.com/en-us/video/remote-desktop-services-rds-explained.aspx> (visited on 02/07/2017).
- [22] *Microsoft Security TechCenter - Microsoft Security Updates*. 2017. URL: <https://technet.microsoft.com/en-us/security/bulletins.aspx> (visited on 01/29/2018).
- [23] G. J. Popek and R. P. Goldberg. “Formal Requirements for Virtualizable Third Generation Architectures.” In: *Communications of the ACM* 17.7 (1974), pp. 412–421.
- [24] M. Portnoy. *Virtualization Essentials*. 1st. Alameda, CA, USA: SYBEX Inc., 2012. ISBN: 1118176715, 9781118176719.
- [25] *Remote Desktop Protocol*. 2017. URL: [https://msdn.microsoft.com/en-us/library/aa383015\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa383015(v=vs.85).aspx) (visited on 02/07/2017).
- [26] M. Righini. *Enabling Intel Virtualization Technology Features and Benefits*. Tech. rep. 2010, pp. 1–9. URL: <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/virtualization-enabling-intel-virtualization-technology-features-and-benefits-paper.pdf>.

- [27] O. Rodeh, J. Bacik, and C. Mason. “BTRFS: The Linux B-Tree Filesystem.” In: *ACM Transactions on Storage* 9.3 (2013), pp. 1–32.
- [28] D. N. S. Shepler M. Eisler. *Network File System (NFS) Version 4 Minor Version 1 Protocol*. RFC 5661. Internet Engineering Task Force (IETF), 2010, pp. 1–617. URL: <https://tools.ietf.org/html/rfc6143>.
- [29] M Satyanarayanan. “A Survey of Distributed File Systems.” In: *Annu. Rev. Comput. Sci.* 4.4976 (1990), pp. 73–104.
- [30] SwiftStack. *OpenStack Swift*. 2017. URL: <https://www.swiftstack.com/product/openstack-swift> (visited on 02/10/2017).
- [31] J. L. T. Richardson. *The Remote Framebuffer Protocol*. RFC 6143. Internet Engineering Task Force (IETF), 2011, pp. 1–39. URL: <https://tools.ietf.org/html/rfc6143>.
- [32] VMWare. *VDI : A New Desktop Strategy*. Tech. rep. 2006, pp. 1–19. URL: https://www.vmware.com/pdf/vdi_strategy.pdf.
- [33] VMware. *Virtualization overview*. Tech. rep. 2006, pp. 1–11. URL: <http://www.vmware.com/pdf/virtualization.pdf>.
- [34] VMware Horizon. 2017. URL: <http://www.vmware.com/products/horizon.html> (visited on 02/07/2017).
- [35] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C Maltzahn. “Ceph: A Scalable, High-Performance Distributed File System.” In: *Proceedings of USENIX Symposium on Operating Systems Design and Implementation* (2006), pp. 307–320.
- [36] *What Files Make Up a Virtual Machine?* 2006. URL: https://www.vmware.com/support/ws55/doc/ws_learning_files_in_a_vm.html (visited on 02/05/2017).
- [37] *XenApp & XenDesktop*. 2017. URL: <https://www.citrix.co.uk/products/xenapp-xendesktop/> (visited on 02/07/2017).

