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 raporteur
Name of another raporteur
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

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Lorem ipsum.

Acknowledgements

The acknowledgements. You are free to write this section at your own will. However, usually it starts with the institutional acknowledgements (adviser, institution, grants, workmates, ...) and then comes the personal acknowledgements (friends, family, ...).

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

Contents

List of Figures

List of Tables

Listings

Acronyms

VDI Virtual Desktop Infrastructure.

VM Virtual Machine.

Chapter 1

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, but not until the development of virtualization technologies for the x86 architecture and the introduction of *Intel VT* and *AMD SVM* 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 [13] model *(IaaS)*, 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 [3] and Google Cloud Platform [9], 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 [5], 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 [18]. 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 iCBD, Infrastructure for Client-Based (Virtual) Desktop (Computing), 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. [11]

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. [1, 11, 12]

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 3.3**.

1.5 Document Structure

The remnant document is structured as follows:

- *Chapter 2* **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.

Chapter 2

Related Work

The focal point of this dissertation is the challenges of implementing a distributed system based on a file system, that can store *VMs* images and leverage the benefits of snapshot and caching techniques. Moreover, it is intended that the proposed solution integrates smoothly into a broader infrastructure.

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

Section 2.1 overviews virtualization as a core concept and discuss in particular the data and metadata storage method.

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

2.1 Virtualization

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 efficiently used.

Virtualization 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, virtualization is an integral part of many *IT* sectors with applications ranging from hardware-level virtualization, operating system-level virtualization, and high-level language virtual machines.

A virtual machine (VM) by design is an efficient, isolated duplicate of a real machine [15], in that order, it was the capacity to virtualize all of the hardware resources, including processors, memory, storage, and network connectivity.

To manage the VMs, there is a need for a layer of software that has certain 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 is known as a virtual machine monitor (VMM), or more common at this moment a *hypervisor*.

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 hypervisor which has that task and should do it in an efficient manner at the same time providing this three properties [15]:

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 hypervisor must have complete control over the resources.

An hypervisor can be classified into two different types, symbolising two different design strategies to virtualization, as shown in Figure **??**:

Type 1 hypervisor: sometimes referred as a bare-metal hypervisor, are very similar to an operating system since they run directly on the hardware and are 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.

Type 2 hypervisor: In comparison, this kind of hypervisor relies on an already installed operating system, and acts very similarly as any conventional process. Having the need to request resources to the OS underneath. On the other hand, there are some advantages. Mainly this solution is easier to deploy since the work of supporting the hardware is already done by the OS below.

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 the management and load balancing. The settings of all these components are compiled in a VM configuration file. In the case of VMware hypervisors, the file employs the `.vmx` extension.[16, 27]

With a virtualized infrastructure there is an opening for a substantial reduction in the number of servers. Which in turn diminishes the setup time of a server as those VMs are commonly created with a resource to cloning techniques. Software updates can be greatly 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 required for the day to day operations.

One solution to the predicament above is to use virtualization as a mechanism for virtualizing the complete workstation. The implementation with more relevance and with more expression at the moment is the virtual desktop infrastructure (VDI).

The concept encompasses a series of techniques, providing on demand availability of desktops, in which, all computing is performed employing virtual machines [23]. Typically this solution presents a centralised architecture, where the user's environment resides on a server in a data centre, as shown on Figure **??**. 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) [17] or the Remote Framebuffer Protocol (RFB) [22] 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 [25], XenDesktop from Citrix [31] and Microsoft with Microsoft Remote Desktop [14].

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 [31], needed to erase the user machine disk completely [6].

As discussed earlier virtualization has paved the way for a class of services called IaaS. That leads to an apanage 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 [27], 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 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 havew:

- 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) [28], the New Technology File System (NTFS) [30] from Microsoft, the Apple's Hierarchical File System Plus (HFS+) [29] 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. [7] We can give as example of file systems the well-known Network File System (NFS) [19] originally developed by Sun Microsystems, and the notable Andrew File System (AFS) [20] 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 [26], OpenStack Swift [21], and in a IaaS flavour the Amazon S3 [2] and Google Cloud Storage [8].

2.2.2 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.

2.2.3 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 does 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 o maintain the correctness guarantees of the data in the appearance of failures, which may occur at any time.

Chapter 3

Proposed Work

This chapter firstly presents the technologies that will serve as the basis for the development of the work proposed in the following section. Then is given an overview of the solution and finally, there is a presentation of a detailed action plan.

3.1 Existing Technologies

This section briefly lists the main technologies the work will be built upon.

BTRFS is a modern file system built from scratch, initially designed by Oracle Corporation, based on the copy-on-write principle. This file system aims at implementing advanced features while also focusing on fault tolerance, repair and easy administration. But its most interesting feature is its ability to create snapshots of directories and files. Btrfs uses B + trees as the main structure for storing the data. Everything in this file system, from inodes, to data, directories, along with others, is an object in the B + tree.

Memcached [10] is an open source, high-performance, distributed memory object caching system. It acts as an in-memory key-value store for small arbitrary data (such as strings or objects) from results of database calls, API calls, or page rendering. This tool is constituted by four major components: 1) Client software that has a list of available memcached servers; 2) A client-based hashing algorithm, which chooses a server based on the key; 3) Server software, which stores values with their keys into an internal hash table; 4) Least Recently Used cache which determines when to throw out old data.

3.2 Proposed Solution

This work focuses on the problematic of storing virtual machines in a context of virtual desktop infrastructure, in this sequence the general idea is to introduce a replication and caching system with basis on a local file system. Moreover, this solution should integrate correctly with the existing platform using the part of the infrastructure which is already functional. There are three fundamental steps to the realisation of this project towards accomplishing all the goals.[1]

First, there is a need for some preliminary evaluation. As already mentioned, this project has been underway for quite some time. In this sense, now that it is the moment to change from a single-node paradigm to a multi-node one, even having in mind a multi-site distribution, the demand for a thorough study is in order. This examination is intended to understand in detail the current operation of the storage infrastructure, and what happens when users use the platform asking for files and put some pressure on the system. This phase is of great importance since its results will be decisive for the following design phases and general shape of the solution.

Addressing the second requirement originates the need to tackle the challenge of introducing a caching system to storage servers. In this design, a client of the storage system will first check if what is being requested is within the cache system and only in case of a cache miss will the request be routed to the layer below. The difficulty here is that this system must put up with reasonably large files resulting from the consolidation of snapshots in a distributed environment. There are some solutions currently in the market such as Memcached, where support is more focused on making small files available, but there is the possibility of tweaking the system for the sake of getting it to work with bigger files.

There are some relevant issues when talking about a single-node solution, such as low fault tolerance, a general scaling difficulty, and limited availability. Here the approach focuses on the idea of replicating a BTRFS-based file system and at the same time ensuring the properties just mentioned. Two methodologies can be followed to complete this objective. One is to use an existing middleware and integrate it with the different components of the infrastructure. The second is to build from scratch a solution that fulfils this requirement. At the moment it is not yet decided which of the approaches to take, but this will be one of the main objectives of the design phase of the solution.

We plan to evaluate our work in a setting that includes servers running on a platform provided by Reditus S.A and featuring the complete iCBD solution. Thus being able to conduct a series of benchmarks in a near real-world setting. Such test can be grouped into three parameter categories to study, the required bandwidth, latency presented and the number of input/output operations per second (IOPS). Comparing these categories

[1] At the time of writing, these are the goals that make sense to us. Still, the architecture of this solution is dependent on consultation with Reditus S.A, a discussion that will only occur after the deadline of this document.

according to various types of operations (sequential reading, random reading, sequential writing, random writing) in order to simulate multiple types of workload that may occur in a deployment of this platform. To this end, we expect to be able to use an open-source tool called Flexible I/O Tester (fio) [4] that mitigates the need of writing tailored test cases and allows the use of job files containing the operating environment that needs to be tested. In addition, it is necessary to evaluate the behaviour of the solution in a virtualized environment with several nodes, including fault tolerance correction and general availability.

3.3 Work Plan

In this section the work plan for the elaboration of the dissertation is described. The work plan is divided in four phases and a final one for writing the dissertation. The work is to be carried out in the period between the last week of February and the penultimate week of September. Table 3.1 shows the proposed durations for each of the tasks and Figure **??** depicts a Gantt chart displaying a summary of the described schedule, better explaining how dates overlap.

Task	Time (Weeks)
1) Preliminary Evaluation	3
2) Caching Feature	11
i) Design	4
ii) Implementation	5
iii) Evaluation	2
3) Replication Feature	13
i) Design	4
ii) Implementation	7
iii) Evaluation	2
4) Final Evaluation	6
i) Final Tests	4
ii) Result Validation	2
5) Writing	10

Table 3.1: Work plan summary.

Bibliography

[1] N. Alves. "Linked clones baseados em funcionalidades de snapshot do sistema de ficheiros." Master's thesis. Universidade NOVA de Lisboa, 2016.

[2] Amazon Web Services. *Amazon Simple Storage Service (S3)*. 2017. URL: https://aws.amazon.com/s3/ (visited on 02/10/2017).

[3] *Amazon Web Services (AWS) - Cloud Computing Services*. 2017. URL: https://aws.amazon.com/ (visited on 02/05/2017).

[4] J. Axboe. *Flexible I/O Tester*. 2017. URL: https://github.com/axboe/fio (visited on 02/15/2017).

[5] 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. ISSN: 07342071. DOI: 10.1145/1365815.1365816. URL: http://research.google.com/archive/bigtable-osdi06.pdf.

[6] *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).

[7] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *Distributed Systems: Concepts and Design*. 5th. USA: Addison-Wesley Publishing Company, 2011. ISBN: 0132143011, 9780132143011.

[8] Google. *Google Cloud Platform - Cloud Storage*. 2017. URL: https://cloud.google.com/storage/ (visited on 02/10/2017).

[9] *Google Cloud Platform*. 2017. URL: https://cloud.google.com/ (visited on 02/05/2017).

[10] D. Interactive. *Memcached*. 2017. URL: https://github.com/memcached/memcached (visited on 02/15/2017).

[11] P. Lopes. *Proposta de Candidatura ao programa P2020*. Tech. rep. DI-FCT/NOVA, Reditus S.A, 2015, pp. 1–26.

[12] E. Martins. "Object-Base Storage for the support of Linked-Clone Virtual Machines." Master's thesis. Universidade NOVA de Lisboa, 2016.

[13] P. Mell and T. Grance. "The NIST definition of cloud computing." In: *NIST Special Publication* 145 (2011), p. 7. ISSN: 00845612. DOI: 10.1136/emj.2010.096966. arXiv: 2305-0543.

[14] *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).

[15] 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. ISSN: 01635980. DOI: 10.1145/957195.808061. URL: http://doi.acm.org/10.1145/361011.361073.

[16] M. Portnoy. *Virtualization Essentials*. 1st. Alameda, CA, USA: SYBEX Inc., 2012. ISBN: 1118176715, 9781118176719.

[17] *Remote Desktop Protocol*. 2017. URL: https://msdn.microsoft.com/en-us/library/aa383015(v=vs.85).aspx (visited on 02/07/2017).

[18] O. Rodeh, J. Bacik, and C. Mason. "BTRFS: The Linux B-Tree Filesystem." In: *ACM Transactions on Storage* 9.3 (2013), pp. 1–32. ISSN: 15533077. DOI: 10.1145/2501620.2501623. URL: http://doi.acm.org/10.1145/2501620.2501623{\%}5Cnhttp://dl.acm.org/citation.cfm?id=2501620.2501623.

[19] 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.

[20] M Satyanarayanan. "A Survey of Distributed File Systems." In: *Annu. Rev. Comput. Sci.* 4.4976 (1990), pp. 73–104.

[21] SwiftStack. *OpenStack Swift*. 2017. URL: https://www.swiftstack.com/product/openstack-swift (visited on 02/10/2017).

[22] 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.

[23] VMWare. *VDI : A New Desktop Strategy*. Tech. rep. 2006, pp. 1–19. URL: https://www.vmware.com/pdf/vdi_strategy.pdf.

[24] VMware. *Virtualization overview*. Tech. rep. 2006, pp. 1–11. URL: http://www.vmware.com/pdf/virtualization.pdf.

[25] *VMware Horizon*. 2017. URL: http://www.vmware.com/products/horizon.html (visited on 02/07/2017).

[26] 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. DOI: 10.1.1.110.4574.

[27] *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).

[28] Wikipedia. *File Allocation Table*. 2017. URL: http://en.wikipedia.org/w/index.php?title=File%20Allocation%20Table&oldid=758818482 (visited on 02/12/2017).

[29] Wikipedia. *HFS Plus*. 2017. URL: http://en.wikipedia.org/w/index.php?title=HFS%20Plus&oldid=764535752 (visited on 02/12/2017).

[30] Wikipedia. *NTFS*. 2017. URL: http://en.wikipedia.org/w/index.php?title=NTFS&oldid=762037695 (visited on 02/12/2017).

[31] *XenApp & XenDesktop*. 2017. URL: https://www.citrix.co.uk/products/xenapp-xendesktop/ (visited on 02/07/2017).