UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

THIAGO ADDEVICO PRESA

Towards Clouds@home: Integration of Virtualization in Desktop Grids with OAR and BOINC

Final Report presented in partial fulfillment of the requirements for the degree of Bachelor of Computer Science.

Prof. Dr. Nicolas Maillard
Orientador

Porto Alegre, dezembro de 2010.

ACKNOWLEDGEMENTS

I've been lucky in so many ways in my life that I have many people to thank.

I can't help but begin with my mother, Rejane. She would be very proud of me doing this work, and I know that this work can only exist because of her loving effort in raising me.

I must also thank my friends who helped me a lot in the course of graduation. Fabio Correa, Ewerton Miglioranza, Vicente Cruz and Joao Vicente Lima. They taught me a lot about life and friendship, and I can't help but to be friend of them in retribution.

I must also thank Barbara Santinon Lago, for being a patient and loving partner while I was doing this work.

On the work side, I must thank Bruno Bzeznik. His advice on technical decisions and writing taught me a lot and certainly made me a better engineer. I must also thank Derrick Kondo for the opportunity of developing this work at INRIA and for the guidance during this work.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

OS	Operating ystem
VM	Virtual Machine
VMM	Virtual Machine Monitor
API	Application Programming Interface
BOINC	Berkeley Open Infrastructure for Network Computing
DBMS	Database Management System
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
REST	Representational State Transfer
BLCR	Berkeley Lab Checkpoint/Restart
VPN	Virtual Private Network
FOSS	Free and Open Source Software
LDAP	Lightweight Directory Access Protocol
GUI	Graphical User Interface
TCP	Transfer Control Protocol
KVM	Kernel-Based Virtual Machine
MPI	Message Passing Interface
SQL	Structured Query Language
ABI	Application Binary Interface
NFS	Network File System
LVM	Logical Volume Manager
HTTPS	Hypertext Transfer Protocol Secure
LHC	Large Hadron Collider
RPC	Remote Procedure Call
RMI	Remote Method Invocation
URI	Uniform Resource Identifier
XML	eXtensible Markup Language

XML-RPC	eXtensible Markup Language – Remote Procedure Call
URL	Uniform Resource Locator
PXE	Preboot Execution Environment
SSH	Secure Shell

LIST OF FIGURES

ABSTRACT

Volunteer computing is the name of the use of end-user resources to do high-performance computations. Two of the biggest challenges of volunteer computing nowadays is to facilitate the application development work of the scientists, and to allow the client to donate in an isolated and transparent way. If a volunteer computing system isn't easy to use by the scientist, scientists will avoid using it regardless of the computing power available. If it isn't easy to use by the volunteer, it will drive volunteers away and harness less computing power. This work describes clouds@home, which is a project to integrate virtual machines running on desktop hosts into the OAR scheduler to make them computing nodes of a virtual cluster. With clouds@home, the scientist must keep only a single version for a single compilation target of its application, thereby reducing the burden of application development. He also isn't bound to any particular API. The volunteer is further protected by the virtual machine monitor isolation layer.

Keywords: virtualization, desktop grids, volunteer computing, resource management, portability.

Clouds@home: Integração de Virtualização em Desktop Grids com OAR e BOINC

RESUMO

Computação voluntária é o nome dado ao uso de recursos dos usuários finais para executar computações de alto desempenho. Dois grandes desafios da computação voluntária atualmente são facilitar o desenvolvimento de aplicações pelo cientista e permitir aos clientes doarem seus recursos de maneira isolada e transparente. Se um sistema de computação voluntária não é fácil de usar para um cientista, cientistas vão evitar empregá-lo, indiferentemente da quantidade de poder computacional disponível. Se ele não é fácil de usar para um voluntário, menos voluntários doarão seus recursos e assim dispor-se-á de um poder computacional menor. Este trabalho descreve clouds@home, que é um projeto para integrar máquinas virtuais executando em máquinas desktop no escalonador OAR de forma a torná-las nodos computacionais de um cluster virtual. Com clouds@home, o cientista precisa manter apenas uma versão para um único alvo de compilação de sua aplicação, assim reduzindo o trabalho de desenvolver aplicações. Ele também não precisa escrever em nenhuma API em particular. A camada de virtualização melhora a proteção do voluntário por representar uma outra camada de isolamento.

Palavras-chave: Virtualização, desktop grids, computação voluntária, gerência de recursos, portabilidade.

1 INTRODUCTION

Volunteer computing is the name of the use of end-user resources to do high-performance computations. It has gained importance, since the end users now hold much more computing power than ever, much more than any centralized approach. To show this, we can compare the BOINC Project, which is the flagship of volunteer computing projects and now holds more than 3,775,961.8 GigaFLOPS (BOINCSTATS,2010) to the most powerful supercomputer, which has the max processing power of 2,566,000 GigaFLOPS (TOP500,2010). It is still worth pointing out that the BOINC network has lots of potential of increasing, accompanying the growth of the Internet and personal computing.

One of the biggest challenges of volunteer computing nowadays is to facilitate the work of the scientists. For instance, if a scientist wants to use BOINC today, he has to rewrite his application using BOINC's API, and also write a single version for each platform (e.g. Windows x86 and Linux amd64), not to mention test and maintain those variations. If we see that, even with the shown computing power of this sort of computation, only around 10 projects use most of the available volunteer computing power on the BOINC network, we have evidence that know-how is a limiting factor to access the volunteer computing technology.

Another challenge is to allow the client to donate in an easy and transparent way. This is also a big challenge, since the volunteer will be opening his computer to a foreign code stream, thus trusting the scientist to use his resource in a thoughtful way. This raises questions regarding isolation and resource usage.

This work advocates a virtualized approach for volunteer computing. To do so, we will describe our experiences and lessons learned so far from implementing clouds@home.

The goal of clouds@home is to integrate virtual machines running on desktop hosts into the OAR scheduler to make them computing nodes of a virtual cluster. We will describe which questions arose and how we've addressed them in this project, discussing considered alternatives.

Our contributions are the following: we’ve developed a desktop computing agent for the OAR batch scheduler, the vmlauncher, which is a BOINC application designed to launch a virtual machine when run, a virtual appliance designed to run the OAR desktop computing agent and we’ve improved the OAR REST API, so that the desktop computing agent interacts with the OAR scheduler by the REST API.

This work was developed as a Google Summer of Code project, and after that as an INRIA internship, specifically at the Laboratoire de Informatique de Grenoble. It was developed very closely with the OAR team.

2 BACKGROUND ON VOLUNTEER COMPUTING

This chapter provides background on volunteer computing. First we will discuss many computing forms related to volunteer computing, including volunteer computing. Then we discuss challenges and desirable properties of a volunteer computing system.

2.1 Forms of computing

This section describes many forms of computing related to volunteer computing. It is intended to define and contextualize volunteer computing. We address here cluster, grid, cloud, and volunteer computing.

The first form of parallel computation we'd like to address is **cluster computing**. For this discussion, we take the definition from (STERLING; LUSK; GROPP, 2003). A computer cluster is a parallel computer constructed of commodity components and runs (as its system software) commodity software. The idea is to build a supercomputer out of assembled computers, even though some vendors sell prepackaged clusters. Clusters might have different purposes, like high-availability, load-balancing, and high-performance. In this work we're interested in the high-performance variant.

The next stop in our series is **grid computing**. According to (FOSTER, 2002), a grid is "a system that coordinates resources which are not subject to centralized control, using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service". More recent definitions have focused on the combination of resources from different administrative domains in order to reach a common goal (BOTE-LORENZO; DIMITRIADIS; GÓMEZ-SÁNCHEZ, 2003).

The dream of grid computing is that computing becomes a utility much in the sense electricity and water are in a modern household. One could simply "plug in" and start computing. The following characteristics are differences between grid and cluster computing:

- Grid nodes are typically more heterogeneous;
- Grid nodes belong to different administrative domains;
- Grid operates in a bigger scale than cluster computing.

These items are also the challenges of grid computing. Heterogeneity raises portability challenges; the involvement of different administrative domains raises questions of privacy, security, authentication and authorization. And there are still the challenges of scalability and interoperability due to the third point.

A new avatar of grid computing arose recently. **Cloud computing**, in particular in its Infrastructure as a Service (IaaS) form, is a form of grid computing, where virtualized resources are allocated dynamically on a pay-as-you-go scheme (VAQUERO et al., 2009). It is important to reference because it is a trend in distributed systems, and also because virtualization is a key technology to allow cloud computing, thus motivating us to explore the benefits of virtualization for other computing forms, in this case, volunteer computing.

The idea behind **volunteer computing** (also seen in the literature as desktop grids or global computing) is to maximize the ease with which people can donate their computing resource to a scientific project (SARMENTA, 2001). There are a couple of differences between grid and volunteer computing. First, grid computing assumes a network of clusters, supercomputers, and regular computers owned by universities, research labs and companies. Secondly, grid computing assumes specialist management of these resources. Naturally, if you involve desktop computers from regular users of the Internet, these assumptions don't hold.

The computing form being advocated here is the **Volunteer Cloud Computing** approach (SEGAL et al., 2009). In comparison with cloud computing, we no longer have the pay-as-you-go component, nor are the virtual machines dynamically allocated. It is similar if we take into account the dynamic nature of the donated virtual machines.

2.2 Challenges of volunteer computing

As already mentioned, the main difference between grid and volunteer computing is that volunteer computing demands a simple setup, while grid computing can sacrifice this for other desirable properties.

In this section we will list and discuss briefly some of the challenges that arises on the design of a volunteer computing platform. This list certainly isn't comprehensive, but it prepares the reader to criticize this work, and also serves as a more concrete discussion of what is volunteer computing, enabling the reader to spot missing points.

We can divide our list in two parts: the challenges that arise from the volunteer perspective and the challenges that arise from the scientist's perspective.

2.2.1 Challenges on the volunteer side

This section describes some of the challenges that arise on the volunteer side. These challenges arise mostly from the model of volunteer assumed on volunteer computing: a non-technical, domestic user, but also from the number and potential variety of volunteers.

Probably the most central one is **ease of use**. We must assume that the volunteer-side software is easy to use, in order to attract volunteers from different technical backgrounds (possibly volunteers without a technical background!), and thus more volunteers and more computing power. In other words, we can't impose difficulties on the process, taking the risk of losing volunteers and therefore computing power.

Due to the varied nature of volunteers, we must take **platform independence** into account. Volunteers might own a variety of computers, running another variety of

operating systems with different packages installed on it. The goal here is, for instance, not to exclude those volunteers who own a Macintosh.

In order not to discourage the volunteer from donating his computing power, the system must have **volunteer-side security**. The idea here is that the volunteer should have to trust the project the least possible.

A requirement that is not typically present on grid systems is **user interface design**. Grid computing can assume that the system administration is done by a technician, whereas volunteer computing must present an interface designed for the non-technician volunteer, where he can manage his participation details and abstract the technical details as much as possible.

2.2.2 Challenges on the scientist's side

This section describes the challenges that arise on the scientist's side. They are similar to the ones on grid computing, but taken to a new plateau, due to the increased heterogeneity, the higher number of administrative domains, and the more likely presence of malicious users and unreliable nodes.

The first one is **programmability**. Developing parallel and distributed high-performance applications is hard, and we should make it as easy as possible. This property is often traded by performance, scalability, and other properties.

A particular case of programmability is **portability**. The volunteers own a potentially huge variety of platforms. Ideally, the scientist must not bother with the variety of hardware and platforms on which his application would run. Maintaining multiple versions of an application due to the variety of platforms is a big hurdle if the maintenance time is considered, especially from a scientist, which ultimately doesn't want to waste time on sheer programming.

A volunteer computing project usually operates with a huge amount of volunteers. Seti@Home, for instance, has over 3 million volunteers. If the project doesn't scale well it won't benefit from a big number of volunteers as it should. So **scalability** is a desired property.

A challenge that clashes with programmability is **adaptability**. Adaptability is the ability of the application to cope with changes in the computing infrastructure e.g. volatility, failures, etc. Since volunteer nodes are inherently volatile, we must use a programming model that addresses adaptability. For instance, MPI (GROPP; LUSK; SKJELLUM, 1999) wouldn't be appropriate; since MPI jobs are static i.e. they must keep the same set of nodes during the whole computation.

Lastly, **economy** is a good challenge. We wouldn't bother to do volunteer computing if it would be cheaper to buy or rent an infrastructure. For an interesting comparison between renting Amazon EC2 instances and starting a volunteer computing project, see (KONDO et al., 2009).

2.3 Final Remarks

In this chapter, we've discussed various forms of computation in order to present background on parallel computing forms. We've also discussed many of the technical challenges of volunteer computing itself, since it is the most relevant computing form to this work.

3 BACKGROUND ON VIRTUALIZATION FOR VOLUNTEER COMPUTING

This chapter provides background on virtualization for volunteer computing. Firstly we will describe what virtualization is, what a hypervisor is and which techniques are used to achieve virtualization. After that we'll summarize the advantages and challenges that virtualization brings for volunteer computing.

3.1 Virtualization

This section describes what virtualization is, what an hypervisor does and how it enables virtualization.

In essence, an operating system is a multiplexer, which multiplexes a virtual resource to a scarce physical one using different strategies. A process, for instance, is a virtualized view of a CPU; virtual memory works analogously for the physical memory and files for the persistent storage.

The idea of virtualization is to present the whole machine as a resource, and multiplex it onto the physical machine. The interface provided by the virtual machine duplicates that of the physical machine, so that one can run operating systems on top of it.

The piece of software that enables virtualization is called **hypervisor**, also referred as Virtual Machine Monitor or VMM in the literature. It sits on top of the hardware, replacing the operating system in a classical system. The guest operating systems are then located on top of the hypervisor.

The hypervisor also multiplexes the physical resources to the guest OS, enforcing protection and isolation as well.

The hypervisor also isolates the devices available on the physical machine by presenting an ideal device to the guest OS. This is important because operating systems are easily ported, since the interface is designed this way.

Our discussion doesn't include operating system virtualization, where one enforces that the host and the guest run the same OS, and present to the guests a virtualized OS interface. We are not addressing this form of virtualization, since we are interested in OS portability.

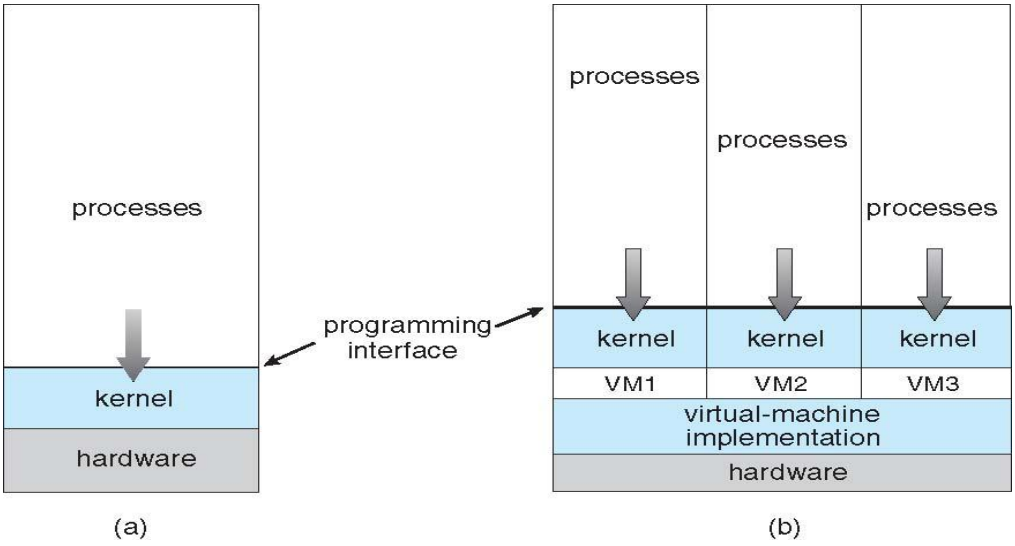

Figure 3.1: Diagram of a system stack (a) without virtualization (b) with virtualization

We now discuss the techniques for implementing virtualization.

Emulation is the simplest form of virtualization, and also the least performing. The technique is a simple switch statement for each instruction, and then the semantics of the instruction are implemented on code. The guest OS is run unmodified. The guest architecture can be different from the host one, as it is the case with video game emulation. Also, the emulator can emulate devices that aren't present on the physical machine.

The problem with emulation is that it is very slow. One can improve the performance by running most of the instructions natively, assuming that the host and guest architectures are the same. This approach isn't feasible on the x86 architecture: some instructions can't be run natively because they fail silently or behave differently than when run on kernel mode. (ADAMS; AGESEN, 2006) discuss this issue further. There are basically three workarounds: dynamic translation, paravirtualization and hardware-enabled virtualization.

Paravirtualization solves the difficulty of virtualizing the x86 architecture by offering an interface to the guest machines that is not identical to the interface offered natively, in order to accelerate and facilitate the support of multiple guest operating systems. The guest OS' kernel must be slightly adapted to support the modified interface. This is a drawback of this approach, since closed-source systems like Windows aren't still ported.

It was one of the first approaches to present good results on modern systems, as one can see in (BARHAM et al., 2003), which also discuss further the difficulties and the workaround adopted.

Another possibility to tackle the difficulty of virtualizing the x86 architecture is to modify slightly the architecture to include a hypervisor mode. Typically in a modern operating system we see the kernel mode and the user mode. These modifications add the hypervisor mode, so that the hypervisor can multiplex kernels into the hardware

without the aforementioned hardware difficulties. Examples of such modifications are the Intel-VT and AMD-V. This has the advantage of running unmodified guest OS kernels e.g. Windows.

Another possibility is to adopt a special cache, which maps each instruction on the virtualized side to a set of physical instructions to be run. The difficult instructions are then dynamically rewritten. This approach is the least performing of the three workarounds.

3.2 Virtualization and Volunteer Computing

This section summarizes the advantages of using virtualization in volunteer computing. It is a summary of the advantages discussed in (FIGUEIREDO; DINDA; FORTES, 2003), slightly adapted for the volunteer computing case.

The first advantage is **isolation**. We have another layer of isolation due to the virtual machine abstraction. This is good because a volunteer can trust less the project he is collaborating. The job assigned to the volunteer might contain bugs. Without a virtual machine, this bug would happen in the same level of the volunteer's processes, potentially affecting them. The extra layer of isolation can thus isolate further the bug.

We claim improvements on **security** due to the least privilege principle. If the job contains malicious code, it will compromise only the virtual machine, instead of the whole physical machine.

Another advantage is **customization**. We can run the job in a customized environment, without having to install software packages on the volunteer machine.

Another advantage is **simplified application development**. For instance, we can develop only for 32-bit Linux architecture and be able to deploy on an x64 Windows volunteer machine.

A corollary of customization is **legacy support**. If our job depends on a legacy piece of software, we can ship it into our virtual machine, without even the knowledge of the volunteer.

Sometimes the application wants to do legitimate use of some resource but it has no authorization. An example would be to mount a remote file system. With the virtualized approach this is not a hurdle anymore, and without breaking the aforementioned isolation and security.

We can have a better **control of resource usage** by the application by specifying the virtual machine configuration prior to execution. Examples of controllable parameters are RAM size and hard drive size. Also, there are some other parameters which are now easy to configure due to network access, like bandwidth usage, traffic shaping, etc. The argument here is that we can reuse the network administration's toolkit to do resource management.

One possibility that arises is to use **system-level checkpointing**. In some projects the computation can take hours or even days. If a volunteer gets some task and doesn't donate anymore, he might block the advance of the project in the whole. Traditionally this would be solved with scheduling the task again to another volunteer, but there is the

possibility that the migration of the VM doing the work would result in a more efficient usage of computing resource.

3.3 Challenges of Virtualization in Volunteer Computing

This section describes the challenges raised by virtualization.

A hypervisor installation may range in size from dozens to hundreds of megabytes, which added to the image size complicates the issue of data transfer. This can drive volunteers away, since volunteers might not be willing to download huge files. Also, it might be considered overly invasive, since the installation typically involves some sort of kernel module, possibly causing a system restart and violating the least privilege principle.

We can't afford to pay for a virtualization solution for each volunteer in our network. Also, there are functionalities that are missing on the free or open-source versions, so our decision must take **licensing** issues into account.

Hypervisors aren't still very interoperable due to the **lack of standards**. A point where this concern is better seen is the programming API of the hypervisors, where each one basically has its own syntax and semantics.

Since the hypervisor can and will be updated, the **versioning** problem arises. It arises in two particular cases: if the update changes the interface of the hypervisor and if the volunteer already uses virtualization, potentially with version clashes from the version used for the volunteer computing project. A simple example for this is that Virtualbox won't work on Linux if the kernel module for the KVM is loaded on a processor with Intel VT-x.

We must consider the problem of sending the image to the volunteer node. Typically those images are big (around hundreds of megabytes), so we must take action to make them short. Narrowing the image's size hinders the scientist/framework developer, since he's allowed to include fewer packages into the distribution, making it less flexible.

Also, we must address the problem of how the volunteer node gets the data which we'll compute upon. It also has the problems of transfer size, but here we must find a compromise solution, since it can affect the overall computing performance.

The need to transfer data to do the computations must be transparent to the volunteer. This means he shouldn’t get billed by bandwidth overuse, nor notices some slowness on his Internet connection.

The question here is whether the volunteer nodes should communicate, and how. The absence of communication allows the nodes only to do data-parallel or task-parallel computations. A naïve approach would lead to a central node routing the communication. The variety of networking solutions used by the volunteer makes a decision in this point difficult, as it might rule out volunteers or desired functionality.

The additional burden should be transparent for the volunteer. He shouldn't have to manually install the virtual machine monitor nor see any pop-up window due to the virtual machine.

3.4 Final Remarks

In this chapter we've discussed the challenges and tradeoffs involved in the virtualized approach for volunteer computing.

4 RELATED WORK

This chapter discusses related work on the subject.

4.1 Volunteer Computing Middlewares

(CECILE et al., 2001) describes XtremWeb, a global computing system. Volunteers must download a Java client in order to donate. This Java client can communicate with the server using TCP/IP, CORBA or RMI. The client keeps sending heartbeats to sign to the server that it is alive and computing. It also addresses the various platforms problems by providing a version of the application for each platform.

(ANDERSON, 2004) describes BOINC, a middleware for volunteer computing. BOINC's goals are the following: to reduce the barriers of entry to public-resource computing, to share resources among autonomous projects, to support diverse applications and, to reward participants. It is designed to require from the scientist only a modest hardware configuration and computer skills.

There are a couple of projects that already use BOINC as platform. SETI@Home, Einstein@Home are the most famous, but not the only ones. As already told on this document, the estimated computing power of the whole BOINC network is 3,775,961.8 GigaFLOPS (BOINCSTATS, 2010). We will discuss BOINC in more detail in chapter 5, since it is an essential part of our design.

4.2 Grid Middlewares

Cigri (GEORGIOU; RICHARD; CAPIT, 2007) is a lightweight grid middleware. The idea is to exploit idle cluster resources by running bag-of-tasks jobs. It must thus cope with the volatility of the nodes. It also allows for system-level checkpointing. They achieve system-level checkpointing by using BLCR (DUELL, 2003), and not by the means of virtualization.

Globus (FOSTER; KESSELMAN, 1996) describes a metacomputing infrastructure toolkit. It describes the mechanisms of this toolkit to tackle the problems of scale, heterogeneity, lack of structure, dynamicity and interoperability. It does so by providing means to select resources according to restrictions given by the user, as well as providing mechanisms for authentication, data access and interfaces for parallel programming.

4.3 Resource Managers

Condor (LITZKOW; LIVNY; MUTKA, 1988) is a scheduling system for a workstation environment. It schedules backgrounds jobs into idle workstations. It uses the Remote Unix (LITZKOW, 1987). The placement of the jobs is transparent to the user, and it uses checkpointing in the case where a workstation running a background job stops being idle. This checkpointing facility works at the process level.

OAR (CAPIT et al. 2005) is an open-source resource manager (batch scheduler) for large clusters. It is developed with high-level tools, such as Perl as scripting language and MySQL as DBMS, without considerable loss of performance or scalability, and implementing most of the important features seen on other schedulers, such as reservations, and interactive jobs.

4.4 Virtual Machine Monitors

VirtualBox (ORACLE VM VIRTUALBOX USER MANUAL, 2010) is a cross-platform virtualization application. It is freely available as Open Source Software under the terms of the GNU General Public License (GPL). It runs on Windows, Linux, Macintosh and OpenSolaris hosts and supports a large number of guest hosts. It doesn't demand hardware support for virtualization, because it uses dynamic recompilation. But its performance improves if hardware supports virtualization. It allows for a shared folder between the host and the guest. It also can present up to 32 virtual CPUs to the virtual machine, enabling SMP on the virtual host. It also can save snapshots, being a useful feature for implementing checkpointing. It also has a command-line front end.

Xen (BARHAM et al., 2003) is a virtual machine monitor for IA-32, x86_64, Itanium and ARM architectures. It is free software and uses both the hardware-assisted approach and the paravirtualized if there's no hardware support. In Xen, one of the multiple virtual machines is called the domain0, and it is typically the operating system that has direct access to the hardware. This domain0 can (as of the version 3.0) Linux, *BSD, Solaris. In machines with support for x86 virtualization, Windows can also be used as domain0. Xen has checkpointing and live migration features. It also has a feature to map a Virtual Block Device to a physical one, where the physical one can be a hard disk, a partition, a NFS mount, a LVM volume or a file. Xen also has the XenAPI, an API for managing the hypervisor.

KVM is a virtual machine monitor for Linux. KVM stands for Kernel-based virtual machine. KVM can only run Linux as host operating system, and demands a virtualization-enabled hardware.

VMware has a handful of virtualization solutions which runs both on Windows and Linux. But their proprietary licensing makes it inappropriate for our purposes.

4.5 Virtualization for Grid Computing

(FIGUEIREDO; DINDA; FORTES, 2003) advocates a virtualized approach for grid computing. His discussion has been summarized in the former chapter. It identifies many qualitative arguments in favor of virtualization and new challenges that arise. It also present benchmarks on the performance of an application running inside a VM and the boot overhead. It also proposes an architecture for a grid using virtual machines. This architecture raises questions that every grid using virtual machines should answer.

4.6 Volunteer Computing Projects with Virtualization

(MAROSI et al., 2008) described their approach for application sandboxing (isolation the application inside a virtual machine). They've developed a wrapper for launching a virtual machine and manage a task inside this virtual machine. Communication direction is always from the wrapper to the task. They used QEMU as

virtual machine monitor, therefore trading performance achieved with other VMMs for features and flexibility. Their wrapper is intended to be portable across different volunteer computing infrastructures, and has been tested against BOINC and SZTAKI, which is a Hungarian BOINC project.

(ANDERSEN. 2006) describes how they scavenged idle desktops to the Minimum intrusion grid (VINTER, 2005). Their approach has some interesting points. They used an embedded Linux distribution for the virtual appliance, achieving a virtual image size of 3MB. To avoid the firewall issue, they've employed a pull architecture over HTTP and HTTPS only for the desktop nodes. The paper doesn't address licensing issues, or the easiness that the volunteer can install the software package to donate.

(DOMINGUES; ARAUJO; SILVA, 2009) addresses further the question of performance and overhead of virtualized nodes. Their findings include that a job running inside a virtual machine suffers a 10-35\% penalty due to virtualization. The CPU-intensive jobs are less affected than the I/O-intensive ones. The perceived overhead to the volunteer was also measured, and it is ranging between 15\% and 30\%. The paper concludes that I/O and network intensive applications should be avoided to run on virtualized environments due to the overhead.

4.7 Custom Execution Environments

(BUNCIC et al., 2010) describes CernVM, a virtual appliance for tackling the problem of deploying the software of an experiment on multiple platforms. They use CernVM-FS (BLOOMER; BUNCIC, 2010) to deploy the files of the experiment into the virtual machine. CernVM-FS is an on-demand read-only filesystem which makes use of aggressive caching. Deployment sizes into the LHC experiment ranges between 2 and 8 GB. They generated the appliance using rBuilder, a tool from rPath. The paper also describes the server infrastructure.

(FERREIRA; ARAUJO; DOMINGUES, 2010) describes libboincexec, a library which extends BOINC to run applications in a custom execution environment. This execution environment can be a virtual machine, or a simple fork()/exec().

5 SOLUTION'S DESIGN

This chapter describes the design of our solution. The goal of this work is to integrate virtual machines running on desktop hosts into the OAR scheduler to make them computing nodes of a virtual cluster. We use BOINC as a tunnel for our virtual appliance, and also to manage the client-side issues. We used BOINC to manage those issues in order to reuse BOINC's features of client preferences.

Our solution comprises 7 parts: the BOINC server and client, the VirtualBox hypervisor, the vmlauncher application, the virtual appliance, the OAR desktop computing agent and the OAR server.

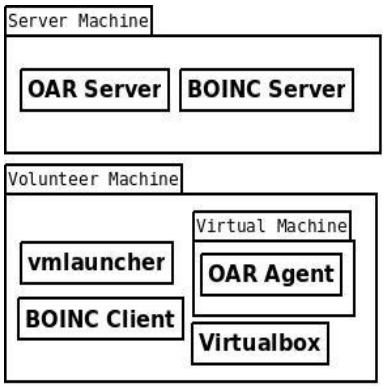

Figure 5.1: Static diagram of the solution

5.1 Structure of the Solution

We wrote a BOINC application that launches a virtual machine. The idea is to use the virtual machine as a sandbox, thereby providing an homogenous platform for the job programmer. It is written in C++ and uses the command-line interface for VirtualBox.

We've developed a desktop computing agent for OAR. The goal of the desktop computing feature is to enable nodes behind firewalls to join the grid. The agent fetches jobs in a pull architecture over HTTP. This means that the desktop computing agent operating in an infinite loop of fetching jobs, executing them and submitting their results. The agent connects to the OAR through the REST API.

We've developed a virtual appliance. A virtual appliance is a virtual machine image designed to run on a virtualization platform. This means that the virtual appliance, when executed, spawns virtual machines. Many virtual machines can be spawned from the same virtual appliance.

The goal is that this virtual appliance runs the desktop computing agent on boot time, and that its size is small.

5.2 Tools

This section describes the tools in which we based our solution upon.

5.2.1 BOINC

The volunteers must install the BOINC client and then attach to one or more projects through the URL. The BOINC client can be run as a screensaver, as a daemon or as a regular application, and the participant can control his preferences i.e. how much of his resources is the project allowed to use. The volunteer gains credit by his participation. It is the experience of the BOINC community that volunteers are highly motivated by credit.

It is necessary to develop applications specifically for BOINC, or to use its wrapper. Even when using the wrapper, it is necessary to develop versions of the applications for each of the desired platforms. Then one must create workunits, which are ways to describe the data to be computed. Each workunit has a result, which represents the result of the application run upon the workunit.

BOINC server implements mechanisms for redundant computing in order to avoid fake results. It also uses a decentralized architecture so that its daemons can be distributed through different hosts and can cope with failure. It also uses an exponential backfoff on the clients to cope with server overload and failure.

5.2.2 Virtualbox

Virtualbox is a virtual machine monitor for x86 and Intel64/AMD64 platforms. We already discussed VirtualBox and other virtual machine monitors in section 4.4.

We've chosen VirtualBox because it is cross-platform, licensed under the terms of the GPL and because it doesn't demand special hardware support.

5.2.3 OAR

OAR is a batch scheduler and a resource manager. This means that OAR receives requests for execution of jobs and it must assign resources from a pool of resources to actually execute those jobs. It provides command-line tools for managing resources (e.g. adding and removing nodes, setting node properties, etc.) and submitting jobs. It manages the dynamicity of resources i.e. when a resource is unreachable, OAR will mark the node as suspected. OAR manages its resources in a push architecture over SSH i.e. when OAR schedules a job to run in a node, it will access that node though SSH and then spawn the job for execution. OAR back end can be either MySQL or PostgreSQL.

Recently, the OAR team developed a REST API, so that one can manage resources and jobs also through a lean web service layer that works over HTTP. We've further developed this API so that the desktop computing agent communicates with the OAR server also through this API. We've done so because of the code reuse and interoperability provided by the REST API.

5.2.4 Kameleon

Kameleon is a virtual appliance generator. It interprets a recipe YAML file in order to generate the corresponding image described in the recipe. The recipe is described as a

list of steps, where each step describes a set of kameleon commands to achieve its semantic. A kameleon command is roughly a shell command to be executed.

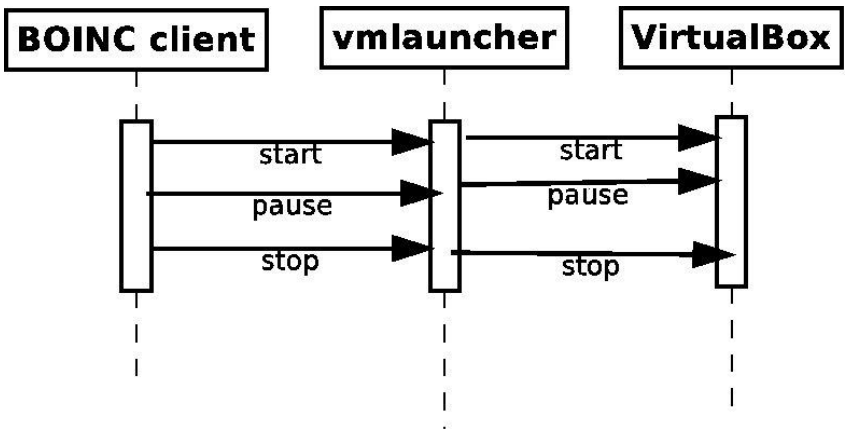

Figure 5.2: Interactions of vmlauncher with VirtualBox and the BOINC client

5.3 Contributions

This section describes the contributions made in this project. In summary, we built a BOINC application to launch virtual machines in VirtualBox, a virtual appliance to act as a desktop computing host in the volunteer computer, the OAR desktop computing agent and its corresponding REST API calls.

5.3.1 vmlauncher

vmlauncher is a BOINC application written in C++ that starts a virtual machine when run. It is based on the BOINC Wrapper. The figure 5.2 explains its mechanics.

As we can see on the figure 5.2, the application forwards and translates the BOINC client's commands to VirtualBox. The BOINC client issues those commands using the OS functionality e.g. signals on UNIX. The vmlauncher application translates them to commands to the VirtualBox command-line front end. This interaction takes place when the volunteer starts donating e.g. when the volunteer's computer is idle.

Another option would have been to modify BOINC's clients, but then the volunteer would have to download our modified client in order to donate, thus imposing another difficulty to the volunteer, without bringing any advantage in contrast to writing an application.

It is also important to explain that we didn't use libboincexec (FERREIRA; ARAUJO; DOMINGUES, 2010) because it would require the virtual machine to be reachable from the host, thus making it more prone to security hazards.

5.3.2 Virtual Appliance

We've developed a Virtual Appliance using Kameleon. This appliance contains a lean debootstrapped debian image with the ruby interpreter and the OAR desktop computing agent installed into it. We used kameleon in order to make it easier to maintain: if we want to change the virtual appliance to include a package, we just have to change the recipe file and re-generate the appliance.

The virtual machine is going to be launched by the BOINC client and the vmlauncher application. When it boots, it must start the OAR desktop computing agent. It is important to the virtual appliance to be small, so that we don't consume the bandwidth of the volunteer.

We could have used CernVM, but the size (800MB) is too big, and we are afraid to lose volunteers due to it. Also, we could have used an embedded Linux distribution based on uclibc (UCLIBC HOMEPAGE, 2010) and busybox (BUSYBOX HOMEPAGE, 2010), which would render our distribution in around 10MB. But the lack of stable packaging systems and big software repositories makes the option a little harder, since it makes the virtual appliance maintenance harder.

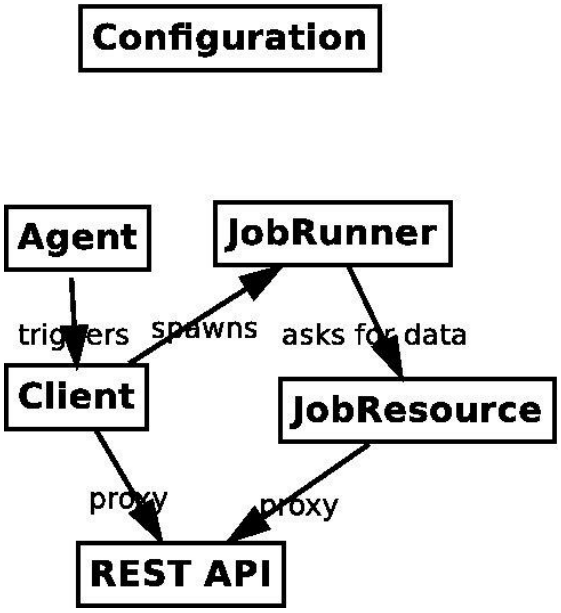

Figure 5.3: Conceptual diagram of the desktop computing agent

5.3.3 OAR desktop computing agent

We've developed a desktop computing agent for OAR. The difference between a desktop host and a service host is that the desktop host can be behind a firewall. This means that a desktop host can't receive connections from outside. In particular, the OAR server can't access a desktop host through SSH, which is what it would normally do to a service node. A way to handle the presence of firewalls is to make the desktop host initiate the connections. That is the goal of the desktop computing feature.

The agent polls the server through the REST API every 30 seconds asking for jobs to run and to kill. Whenever there are jobs for the agent to run, it spawns another process, which will in turn ask the REST API for the details of the job e.g. command to be run, stagein file, and also update the state of the job. It will then run the job and submit the results as a stageout, even in the case of error.

We wrote the agent in ruby, so that we could benefit from a dynamic and object-oriented design style. This improves its readability and makes it easier to modify. The figure 5.3 pictures the classes written and their responsibilities.

The agent class triggers the client class every 30 seconds to run and kill jobs. The client class will then acts as a proxy for the REST API, asking for jobs and spawning a JobRunner class for each job, passing to it a JobResource. The JobRunner merely executes the job, asking for the job details to the JobResource, which acts as well as a proxy for the REST API. The configuration is a singleton that parses the configuration file.

Once developed, we developed a debian package for the agent. This way, one can install the agent on his computer by typing apt-get install oar-desktop-computing-agent from the OAR repository.

5.3.4 OAR REST API Improvements

We decided to make the communications between the agent and the server through the REST API. The OAR REST API is an API accessible through HTTP that follows the REST architecture. A REST architecture over HTTP means that each URI identifies a resource and that the action on this resource is identified by the HTTP method used to access the URI. The advantages of using the REST API for the communication are to have a high-level, interoperable and scalable communication medium. Also, the API calls developed here can be used for other purposes, enhancing code reuse.

In order to design a REST API, we must first identify the resources. A resource is an entity or concept of the problem domain. Then we must identify the relationship between the resources. Finally, we must determine which operations we want to perform on those resources.

We identified the following resources: job, stagein, stageout, job state and node. Then we identified the relationships between the resources: a job can have one stagein and one stageout, and must have a state. A node is said to have many jobs if those jobs are assigned to this node.

Finally, we identified the operations. Thus, we've developed the following REST API calls:

- GET /jobs/<job_id>/stagein to download the stagein file assigned to the job whose id is job_id
- POST /jobs/<job_id>/stageout to upload the stageout file related to the job whose id is job_id. It supports multipart upload.
- GET /resources/nodes/<node_id>/jobs(.*)?state=<job_state> to list the jobs scheduled under the state job_state to the node whose id is node_id
- GET /desktop/agents to get a newly generated node_id for this node. This is actually a bad example of RESTful design, because the URI doesn't identify a resource.
- POST /jobs/<job_id>/state to change the state of the job whose id is job_id to the one given in the message body

Other options for the communication would be sockets, RPC/RMI or other web services, like XML-RPC. It would be harder to develop using sockets, since it is lower level than a RESTful web service. RPC and RMI would not be as interoperable as a RESTful web service, since they are typically bound to a single programming language.

Other web services approaches are typically not as simple to implement as RESTful web services.

5.4 Operation

The purpose of this section is to explain how each of the users of the system view it. We then present the system from the perspective of the volunteer, of the scientist and of the project administrator. We also present briefly how the system reacts to the actions of those users, in order to explain better the system's internals.

5.4.1 Volunteer's View

Volunteers want to support projects by allowing those projects to do computation with the resources of the volunteer i.e. donate to the project. A volunteer must not be technically skilled to collaborate with the project.

In our solution, volunteers must install the BOINC client and VirtualBox to donate to the project. He must then attach to our BOINC project through its master URL. This all can be done through the GUI, since both BOINC and VirtualBox installations are graphical, and attaching to a project is also a graphical process. Nevertheless, it is a manual process, requiring the volunteer to install each one separately and attach manually.

Volunteers also want to set rules on this collaboration. He wants to tell the computer that, for instance, he doesn't want his computer to be used during office hours, or if he wants to collaborate only when the computer stays idle for more than 5 minutes. BOINC enables that by the user preferences, and our application complies with it.

When the volunteer starts donating, according to his preferences, the BOINC client downloads the vmlauncher application and the virtual appliance, and then run the vmlauncher application. The vmlauncher application imports the virtual appliance into the hypervisor and then launches it. The desktop computing agent is then launched at the boot of the virtual appliance, running an infinite loop of fetching jobs from the OAR server, running jobs and submiting results. When the volunteer stops donating, the virtual machine is powered off instantly.

5.4.2 Scientist's View

Scientists want to write the application and submit the job to be computed by the desktop hosts. We should assume that the scientists' knowledge of computer goes little beyond the scope of his application.

In our solution, the scientist must submit a job to OAR. If he is already used to do so, the only differences are the flags he must specify on the submission command. There are two flags that are important for the scientist to make desktop computing jobs work: the desktop_computing flag and the stagein flag.

The desktop_computing flag marks the job as a desktop computing job. This means that it won't get scheduled to a service node, but only for the nodes marked as desktop_computing nodes. Nodes joining through the REST API as described above are automatically marked as desktop_computing nodes.

The stagein flag signalizes for the submission tool that it must prepare the stagein file for the job. The stagein consists in a tar package containing the directory in which

the command is to be executed. It is necessary for the desktop computing jobs that are more complex than a simple command on the virtual appliance.

5.4.3 Administrator's View

Administrators want something easy to manage, stable and secure. Although the administrator has a technical expertise, we should try to minimize what he has to learn in order to deploy our solution.

When the administrator gets the request to host a project, he must deploy both BOINC and OAR servers on a machine. It is not necessary for them to be in the same machine.

To deploy the BOINC server, he must compile the BOINC server. Other options are downloading a virtual appliance for VMWare or using the prebuilt packages on the debian repository.

Once compiled, he has to use the make_project tool to create the project. He can pass the test app flag to add a test application. Then he can test with a vanilla BOINC client to see if there are no networking problems, and the application is downloaded, run and uploaded successfully. Typically naming issues arise here e.g. the hostname of the machine isn't the same that's on the DNS server.

To deploy the OAR server, he must edit the souces of the apt repository to point to the OAR repository. Then the packages oar-server and oar-api must be installed on the server. Also, apache's modules rewrite and ident must be set. It is also necessary to configure apache for authentication mechanisms. The default is to use LDAP. Once installed it can be tested by generating a virtual appliance with the OAR desktop computing agent, point it to the server hostname and see if the host appears at the oarnodes.

In the BOINC server, he must copy the vmlauncher app properly compiled for each architecture to the apps directory.

It is also necessary to create a workunit that states that the virtual machine's files are to be copied to the client, as described in the same link.

5.5 Final Remarks

This chapter presented the design of our solution, whose goal is to integrate virtual machines running on desktop hosts into the OAR scheduler to make them computing nodes of a virtual cluster. We described each part of the solution individually, separating the given tools and the contributions. We also discussed the operation of the solution from different perspectives to give a sense of the whole working of the solution.

6 SCALABILITY TEST

We needed to assess that our solution indeed scales well for usage on the Internet. To do so, we designed a scalability experiment: we want to run a ray tracing job using our project. We used the Grid'5000 infrastructure to run the tests.

We will first describe Grid'5000 infrastructure and software ecosystem in order to make the test procedure description meaningful. Then we will describe our test job. After that we will describe the tasks performed in order to run the test and then we will discuss its evaluation.

6.1 Tools

6.1.1 Grid'5000

Grid'5000 is a French nation-wide infrastructure for research in Grid computing. It is designed exclusively for computer science research, allowing much more flexibility than traditional grids. We used it because it is freely available for doing the experiments, and because we could simulate loads in it e.g. many requests at the same time, many clients.

Grid'5000 is composed of 10 sites distributed geographically. A site is a geographical location that aggregates one or more clusters. A cluster is an aggregate of one or more nodes. A node can be of two types: service node or computing node. The computing nodes are those where the actual computation takes place. The service nodes provide infrastructure service e.g. the frontend node. All hosts on Grid'5000 run a x86-64 architecture. Each node has at least 80GB of local storage. Each site features its own NFS server.

Each cluster has at least 1Gbps Ethernet connecting the nodes. The clusters might also feature Myrinet or Infiniband, which features 10Gbps and low-latency. Nodes can't access the internet directly; they must go through a proxy. This proxy has a list of allowed hosts, blocking the access to any host not present in the list.

Grid'5000 uses OAR for resource management and job scheduling and Kadeploy for images deployment into the nodes.

6.1.2 Kadeploy

Kadeploy (GEORGIOU et al., 2006) is a tool for deploying custom operating systems in a large-scale infrastructure. It is based on the preboot execution environment (PXE).

Kadeploy operates in a master-slave scheme. When the slaves are rebooted, they fetch and run a basic boot program, which create the partitions and connect to the server. When all slaves are connected to the server, the server arranges them into a chain of TCP connections, and then uses this chain to broadcast the image to the slaves, which will in turn deploy the image into their hard drives.

6.1.3 Taktuk

We start the desktop computing agents using Taktuk (CLAUDEL; HUARD; RICHARD, 2009). Taktuk is a remote execution tool i.e. a tool to execute commands on a group of remote hosts. It uses a work-stealing strategy to achieve better scalability. The work-stealing algorithm is an algorithm to distribute the job in a tree-like manner, in this case the job being the command to be executed remotely.

6.2 Test Job Details

6.2.1 POVRay

The Persistence of Vision Ray-Tracer is a ray-tracing tool used to render three-dimensional images. It reads a scene description file, which describes the objects, the lighting and the camera position. Ray-tracing is a time-consuming process, but it produces very high quality and realistic images. It is also embarrasingly parallel.

6.2.2 The Job Campaign

We rendered a complex 3D scene in a definition of 1280x720 based on a camera movement inside the scene. The camera rotates 2 times and elevate slightly. We rendered a frame for each degree of the 2 rotations, which makes 720 frames. At 25 frames per second, this is a 28-second video. Each job renders a frame, and afterwards we build the video out of the frames.

6.3 Method

This section describes the tasks performed in order to run the test and then we will discuss its evaluation.

The first task was to develop the environments. We developed two environments: an OAR server environment and an OAR agent environment. The OAR server environment contains the OAR server and an Apache server publishing the REST API. The OAR agent environment contains the OAR desktop computing agent.

To develop those environments, we deployed with Kadeploy a default debian Lenny environment, modified it as specified above, and then saved it using the tgz-g5k tool, which is a tool for creating the image of a running environment on Grid'5000.

The second task was to do the submissions. We did two OAR interactive submissions: one for the server and another for the clients. In the client submissions we asked for many nodes. Then we would deploy the agent environment in the group of nodes using Kadeploy and then launch our OAR desktop computing agent on them using Taktuk.

The third task is to run the agents. Since we can't know in which host our OAR server environment will be deployed, we must use Taktuk to pass this information to the agent machines, and then start the OAR desktop computing agent.

The fourth and final task is to submit the job once the agents start being managed by the OAR server. Our job is embarrassingly task-parallel: our job campaign consists of the same command invoked each time with different parameters.

6.4 Evaluation

For the evaluation we did the process aforementioned for a varying number of nodes. We've started with 10 nodes/80 cores and increased repeatedly by 10 nodes. It is important to point that Kadeploy often can't deploy the environment on all the machines allocated. For instance, even though we had 20 nodes allocated in one of the points of the graph, Kadeploy could deploy the agent environment on only 17 nodes. We proceeded as it is because the exact number of nodes shouldn't impact on scalability i.e. regardless of the exact number of nodes we expect the scale-up to be almost linear.

We consider the execution time to be the interval between the start time of the first job of the job campaign and the stop time of the last job. We retrieved the start time and stop time from the OAR server.

As of the test results, from 320 cores on, it stopped working. When we investigated the cause of the malfunction, we discovered that it was due to excessive number of concurrent connections to the database, which is a problem that affects the whole OAR REST API. This is because it creates a connection to the database for each HTTP request received.

The test revealed that our solution is still unfit for the Internet, but can already be run on a group of workstations. Two workarounds are planned for the database connection problem: a connection pool mechanism in the OAR persistence layer and a backoff mechanism for the agent.

6.5 Final Remarks

This chapter described how we tested our project for scalability. We described Grid'5000 and then two of its associated software packages: Kadeploy and Taktuk. We then described our job, which uses the POVray ray tracer to render a complex 3D scene. We then described our steps into setting up the test and executing it. We've found that our solution doesn't work yet beyond 40 nodes/320 cores, due to excessive simultaneous connections, and we proposed workarounds for the problem.

7 CONCLUSION

In this chapter we discuss some qualitative questions about our design and some suggested future work. We then summarize the report in order to finish the document.

7.1 Qualitative evaluation

In this section we evaluate our design qualitatively. We discuss improvements and new challenges raised by our solution according to the properties discussed in chapters 2 and 3. We present our arguments for the claims of improvement and future directions to face the new challenges.

7.1.1 Programmability

We claim improvements in programmability and portability, since the developer doesn't have to write his application in any particular API. This means that he can develop using the programming library that is more suited for him. We also claim such improvements because now the developer can program for a single compilation target, and thus doesn't need to keep multiple versions of his application.

Customization is also enabled by virtualization. The virtual appliance can contain the software package necessary for the jobs, and since it would be deployed in the virtual appliance, it is transparent for the volunteer. This software package can be a legacy software package, so we also claim improvements on legacy support based on the same argument.

7.1.2 Ease of use

Our proposed solution is harder to use than the vanilla BOINC client, since the volunteer needs to install VirtualBox manually. To solve this issue, there are efforts to bundle VirtualBox with BOINC. Another possibility is to consider QEMU with KQEMU, since it runs on user space, but the additional overhead and the fact that KQEMU is experimental for Windows and unavailable for Mac is to be further investigated.

7.1.3 Security

Security can drive volunteers away if a virtual machine represents a security hazard for them. We claim improvement in security due to the extra isolation layer, which means that if the job contains malicious code it will be isolated from the other processes of the volunteer. Another point in favor is the fact that the VM is unreachable through the network, which means that the VM doesn't represent additional network security risk.

7.1.4 Adaptability

OAR doesn't handle adaptability in the job level, only in the resource level. This means that if a node fails or stops donating, the computation scheduled won't be rescheduled. It is not the intent of the OAR to handle adaptability in the job level, so we propose an integration with Cigri, which we discuss below.

7.1.5 Instalation Size and Licensing

By choosing VirtualBox, we've compromised with a modest installation size (around 50 MB), a FOSS license and we don't face standards issues due to the single virtualization technology. The virtual appliance size was of 200MB, which is considered modest in contrast to CernVM (800MB).

One could use an embedded Linux distribution, but the lack of packaging systems and big package repositories is an obstacle for the programmability.

7.2 Future work

This section discusses future directions in the course of exploring improvements of volunteer computing systems by the means of virtualization.

7.2.1 B-OAR and VM wrapper

While using BOINC as the tunnel for our virtual appliance we've wondered about the advantages and disadvantages of another approach: integrating BOINC and OAR at the job level, and handle the virtualization layer fully in BOINC.

B-OAR is a bridge interface for submission of BOINC jobs through OAR. We can combine that with the upcoming default VM wrapper for BOINC or libboincexec.

With this approach, the resource management of the volunteer nodes is done fully in BOINC, being thus potentially simpler. Also, BOINC would handle the dynamicity of the nodes, resubmitting the jobs when necessary. Checkpointing would also be managed inside BOINC, being transparent to OAR.

7.2.2 Cigri integration

Another option to tackle the adaptability of jobs is to use Cigri.

Cigri is a job manager for bag-of-tasks jobs. It uses OAR as resource manager, but it is designed to interoperate with other resource managers as well.

This approach would leave the validation of volunteer's result open yet, so another layer for solving the validation issues would be necessary.

7.2.3 On-demand file system

The problem of file staging is that if you have a task-parallel job, you will have many times the same stagein staged. If you have a data-parallel job, you will probably ship unneeded data to the agents.

A possible improvement is to use an on-demand file system, like CernVM-FS. It is a mountable Linux filesystem that downloads the demanded files only when then files are actually read by the job. It operates over HTTP in order to avoid firewalls.

For writing the results we can still use stageouts, since CernVM-FS mounted directory is read-only.

7.2.4 Other possible works

We are unaware of any system that actually uses VMs for checkpointing in volunteer computing, although checkpointing is greatly simplified by the use of virtualization technology. It is relevant, since checkpointing is hard to implement, and

often relegated to the scientist. Thus, by achieving VM checkpointing we would be making the job of the scientist even lighter.

Another possibility is to use Virtual Private Networks to present to the nodes the illusion of being at the same IP network. Currently the nodes can't communicate between themselves. The goal would be to run the n-queens problem in a volunteer network.

We can now explore further scheduling strategies for desktop grids, since OAR is very flexible in this sense. In particular, we'd like to cite the possibility of fair sharing, reservations, priorities and job containers.

Further control of resource usage is possible e.g. traffic shaping. Also, management of the number of virtual CPUs seen by the virtual machine is also possible.

A KQEMU port for Mac would make KQEMU suitable for volunteer computing, and thus we could explore further which hypervisor would be a better fit.

7.3 Summary

This work presented clouds@home, whose goal is to integrate virtual machines running on desktop hosts into the OAR scheduler to make them computing nodes of a virtual cluster. We also ran scalability tests and found that it still doesn't scale well enough for Internet usage. Nevertheless, the root causes of the scalability problems are merely technical, and thus not inherent of any part of our design. We would like to proceed further and achieve better scalability results, and to address questions such as adaptability better, as proposed on the future works.

As a personal conclusion, working in this project was a nice experience with scalable design, since I had resources to test if my design was indeed scalable, and I could learn a few things about good and bad design in terms of scalability. I also learned a good deal about reading research papers (particularly about reading them and the value of keeping a journal with summaries of papers read) and writing this report, about the need to structure my ideas and strategies to explain them.

REFERENCES

ADAMS, K.; AGESEN, O. A comparison of software and hardware techniques for x86 virtualization. In: ASPLOS-XII: PROCEEDINGS OF THE 12TH INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, New York, NY, USA. . . .

ANDERSEN, R. Harvesting Idle Windows CPU Cycles for Grid Computing. 2006.

ANDERSON, D. P. BOINC: a system for public-resource computing and storage. In: IEEE/ACM INTERNATIONAL WORKSHOP ON GRID COMPUTING, 5., Washington, DC, USA. Proceedings. . . IEEE Computer Society, 2004. p.4–10. (GRID '04).

BARHAM, P. et al. Xen and the art of virtualization. In: SOSP '03: PROCEEDINGS OF THE NINETEENTH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, New York, NY, USA. . . . ACM, 2003. p.164–177.

BLOMER, J.; BUNCIC, P. The CernVM File System. [S.l.]: CERN, 2010.

BOINCSTATS. Available at boincstats.com. Accessed in November 2010

BOTE-LORENZO,M. L.; DIMITRIADIS, Y. A.; GóMEZ-SáNCHEZ, E. Grid Characteristics and Uses: a grid definition. In: CROSS GRIDS 2003, LNCS 2970. . . . [S.l.: s.n.], 2003. p.291–298.

BUNCIC, P. et al. CernVM – a virtual software appliance for LHC applications. Journal of Physics: Conference Series, [S.l.], v.219, n.4, p.042003, 2010.

BUSYBOX homepage. Available at www.busybox.net. Accessed in November 2010

CAPIT, N. et al. A batch scheduler with high level components. In: FIFTH IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID (CCGRID'05) - VOLUME 2 - VOLUME 02, Washington, DC, USA. Proceedings. . . IEEE Computer Society, 2005. p.776–783. (CCGRID '05).

CECILE, G. F. et al. XtremWeb : a generic global computing system. In: IN PROCEEDINGS OF THE IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID (CCGRID'01. . . . [S.l.: s.n.], 2001. p.582–587.

CLAUDEL, B.; HUARD, G.; RICHARD, O. TakTuk, adaptive deployment of remote executions. In: ACM INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE

DISTRIBUTED COMPUTING, 18., New York, NY, USA. Proceedings. . . ACM, 2009. p.91–100.

DOMINGUES, P.; ARAUJO, F.; SILVA, L. Evaluating the performance and intrusiveness of virtual machines for desktop grid computing. Parallel and Distributed Processing Symposium, International, Los Alamitos, CA, USA, v.0, p.1–8, 2009.

DUELL, J. The design and implementation of Berkeley Lab's linux Checkpoint/Restart. [S.l.: s.n.], 2003.

FERREIRA, D.; ARAUJO, F.; DOMINGUES, P. Custom execution environments in the BOINC middleware. In: IBERIAN GRID INFRASTRUCTURE CONFERENCE, 4., Braga, Portugal. Proceedings. . . [S.l.: s.n.], 2010.

FIGUEIREDO, R. J.; DINDA, P. A.; FORTES, J. A. B. A Case For Grid Computing On Virtual Machines. In: ICDCS '03: PROCEEDINGS OF THE 23RD INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, Washington, DC, USA. . . . IEEE Computer Society, 2003. p.550.

FOSTER, I. What is the Grid? A Three Point Checklist. GRIDtoday, Vol. 1, No. 6., [S.l.], June 2002.

FOSTER, I.; KESSELMAN, C. Globus: a metacomputing infrastructure toolkit. International Journal of Supercomputer Applications, [S.l.], v.11, p.115–128, 1996.

GEORGIOU, Y. et al. A tool for environment deployment in clusters and light grids. Parallel and Distributed Processing Symposium, International, Los Alamitos, CA, USA, v.0, p.434, 2006.

GEORGIOU, Y.; RICHARD, O.; CAPIT, N. Evaluations of the Lightweight Grid CIGRI upon the Grid5000 Platform. In: SCIENCE AND GRID COMPUTING, IEEE INTERNATIONAL CONFERENCE ON. . . . [S.l.: s.n.], 2007. p.279 –286.

GROPP, W.; LUSK, E.; SKJELLUM, A. Using MPI (2nd ed.): portable parallel programming with the message-passing interface. Cambridge, MA, USA: MIT Press, 1999.

KONDO, D. et al. Cost-benefit analysis of Cloud Computing versus desktop grids. In: IPDPS '09: PROCEEDINGS OF THE 2009 IEEE INTERNATIONAL SYMPOSIUM ON PARALLEL&DISTRIBUTED PROCESSING, Washington, DC, USA. . . . IEEE Computer Society, 2009. p.1–12.

LITZKOW, M. J. Remote UNIX: turning idle workstations into cycle servers. In: SUMMER USENIX CONFERENCES, 1987., Phoenix, Arizona, USA. Proceedings. . . [S.l.: s.n.], 1987.

LITZKOW, M.; LIVNY, M.; MUTKA, M. Condor-a hunter of idle workstations. In: DISTRIBUTED COMPUTING SYSTEMS, 1988., 8TH INTERNATIONAL CONFERENCE ON. . . . [S.l.: s.n.], 1988. p.104 –111.

MAROSI, A. C. et al. Using virtual machines in desktop grid clients for application sandboxing Technical Report TR-0140. [S.l.]: Institute on Architectural Issues: Scalability, Dependability, Adaptability, CoreGRID - Network of Excellence, 2008.

ORACLE VM VirtualBox User Manual. [S.l.]: Oracle Corporation.

SARMENTA, L. F. G. Volunteer Computing. 2001. Thesia (PhD in Computer Science) — Massachusetts Institute of Technology.

SEGAL, B. M. et al. Building a volunteer cloud. In: CLCAR 2009: CONFERENCIA LATINOAMERICANA DE COMPUTACI´oN DE ALTO RENDIMIENTO. . . . Universidad de Los Andes, 2009.

STERLING, T.; LUSK, E.; GROPP, W. (Ed.). Beowulf Cluster Computing with Linux. Cambridge, MA, USA: MIT Press, 2003.

TOP500. Available at www.top500.org. Accessed in November 2010.

UCLIBC home page. Available at www.uclibc.org. Accessed in November 2010.

VAQUERO, L. M. et al. A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev., New York, NY, USA, v.39, n.1, p.50–55, 2009.

VINTER, B. The Architecture of the Minimum intrusion Grid: mig. In: COMMUNICATING PROCESS ARCHITECTURES. . . . IOS Press, 2005. p.189–201.