TÉCNICO
LISBOA

Green-Cloud

Economics-inspired Scheduling, Energy and Resource Management in Cloud Infrastructures

Rodrigo Tavares Fernandes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Doctor Luís Manuel Antunes Veiga

Examination Committee

Chairperson: Doctor José Carlos Alves Pereira Monteiro
Supervisor: Doctor Luís Manuel Antunes Veiga
Member of the Committee: Doctor David Manuel Martins de Matos

November 2015

Acknowledgments

I would like to thank all the kind people that made it possible to conclude this thesis.

In no specific order:

I would like to thank my advisor, Luís Veiga for stimulating me and allowing me to work in this project and also José Simão for the help understanding Cloudsim and for his previous work on the scheduling problem. The obtained results would never be the same without their outstanding coordination.

I am also very grateful to all my Codacy colleagues who kept supporting me during my work on the thesis.

To my family and all my friends, for their understanding during all the days I could do nothing except my thesis.

Abstract

Cloud computing gained immense importance in the past decade, emerging as a new computing paradigm and aiming to provide reliable, scalable and customisable dynamic computing environments for end-users. The cloud relies on efficient algorithms to find resources for jobs by fulfilling the job's requirements and at the same time optimise an objective function. Utility is a measure of the client satisfaction that can be seen as an objective function maximised by schedulers based on the agreed service level agreement (SLA). Our *EcoScheduler* aims at saving energy by using dynamic voltage frequency scaling (Dynamic Voltage Frequency Scaling (DVFS)) and applying reductions of utility, different for classes of users and across different ranges of resource allocations. Using efficient data structures and a hierarchical architecture, we created a scalable solution for the fast growing heterogeneous cloud. *EcoScheduler* proved that we can delegate work in a hierarchy, and make decisions based on partial data and still be efficient.

Keywords: Cloud, Utility Scheduling, DVFS, Energy Efficiency, Partial Utility

Resumo

A computação na nuvem ganhou imensa importância durante a última década, emergindo como um novo paradigma de computação e com o objetivo de proporcionar ambientes de computação dinâmicos, fiáveis, escaláveis e personalizáveis para os utilizadores. O escalonamento de tarefas baseia-se em algoritmos eficientes que encontram recursos para alocar o trabalho definido nas tarefas, cumprindo as exigências do pedido e ao mesmo tempo optimizando uma função objetivo. Utilidade é uma medida da satisfação do cliente, que pode ser vista como uma função objetivo maximizada pelos escalonadores com base no nível de serviço acordado (SLA). O *EcoScheduler* é a nossa solução que visa poupar energia, utilizando o escalonamento da frequência do processador e aplicando reduções de utilidade, diferentes por classe de utilizadores e entre diferentes níveis de alocação de recursos. Usando estruturas de dados eficientes, e uma arquitetura hierárquica, criámos uma solução escalável para o rápido crescimento da nuvem. O *EcoScheduler* prova que se pode delegar o escalonamento usando uma hierarquia, e tomar decisões baseadas em dados parciais e ainda assim ser eficiente.

Palavras-Chave: Computação em nuvem, Infraestrutura como um Serviço, Máquinas Virtuais, Alocação de Recursos, Eficiência Energética, Escalonamento de Máquinas Virtuais

Contents

List of Tables

List of Figures

Many of life's failures are people who did not realize how close they were to success when they gave up.

— *Thomas Edison*

Chapter 1

Introduction

The cloud computing paradigm changed the way we perceive and use information technology services. These services aim to provide reliable, scalable and customisable dynamic computing environments for end-users. With the dynamic provision of resources, the cloud can provide better management of resources by optimising their usage with a pay-as-you-go pricing model.

1.1 Motivation

The cloud is built over datacenters spread all over the world usually containing large groups of servers connected to the Internet. These infrastructures have to be maintained by the providers that take care of all the working infrastructure, and have to keep in mind the environmental footprint and the wasted energy. To achieve better energy efficiency results, the providers rely on scheduling algorithms to manage the datacenters.

Scheduling algorithms try to find resources for a job by fulfilling its requirements and at the same time optimising an objective function, that takes into consideration the user satisfaction and the provider profits. Utility is a measure of a user's satisfaction that can be seen as an objective function that a scheduler tries to maximise based on the Service Level Agreement (SLA).

The performance issues of the scheduling algorithm include, not only execution times, but also resource utilisation. A better scheduler can use fewer resources and run jobs faster. Using fewer resources is very important, especially because it helps consume less energy, and energy consumption is one of the major issues for building large-scale clouds. At the same time, it is undesirable to have idle resources and waiting jobs.

1.2 Goals

The goal in this work is to develop a scheduling algorithm for cloud scenarios that takes into account resource-awareness (CPU cores and computing availability, available memory, and available network bandwidth) and declarative policies that express resource requirements and perceived satisfaction, with different resource allocation profiles awarded to users and/or classes of users.

We propose *EcoScheduler*, a scheduling algorithm for allocating jobs in the cloud with resource-awareness and user satisfaction, using different resource allocation profiles chosen by the clients. We enrich our model with the notions of partial utility and by incorporating DVFS for improved energy efficiency. Our scheduling algorithm proposes to efficiently assign proper resources to jobs according to their requirements.

1.3 Document Organisation

This document is organised as follows. In Chapter 2 we present the bases where our work is built upon and review the state of the art algorithms and techniques used in the literature. Chapter 3 is the description of our proposed solution: we present how we addressed the problems of the other existing alternatives and how our solution works. The solution was implemented in a state of the art simulator, Cloudsim [1], and all the implementation details of the solution are reported in Chapter 4. All the results and metrics of the test in Cloudsim can be found in Chapter 5. Some concluding remarks are presented in Chapter 6 together with some possible future work.

Chapter 2

Related Work

This section describes the most relevant research work for the definition of *Eco-Scheduler*, our eco-friendly scheduling algorithm, organised according to a top-down approach. In Section 2.1 we present a background analysis of cloud computing with an overview of the concepts behind it, such as virtualisation, virtual machines, and hypervisors. Next, in Section 2.2 we describe various aspects about scheduling, such as types of virtual machine scheduling algorithms and how they differentiate from each other. Finally, in Section 2.3 we describe energy and environmental aware algorithms. We conclude with analysis and discussion.

2.1 Cloud Computing and Virtualisation

Today, most Information Technology (IT) companies face challenges related to fast changing environments with very specific requirements. These conditions happen for both modern and legacy applications which need reliability, security and sometimes strict assured quality of service (QoS). To mitigate these problems, some companies started providing on-demand services, self managed, offered through well-designed web platforms and paid-by-usage, this is called the cloud. All this flexibility is achieved using virtualisation, which is a technique that splits physical infrastructures of resources of isolated computing parts.

Figure 2.1 describes some very important events in the history of virtualisation that lead to its widespread use and the expansion of the cloud.

2.1.1 Cloud

The cloud is a way of delivering hosted services provided through the internet, sold on demand, typically measured in time periods, with great elasticity and scalability. Its recent expansion, since 2007 [2], occurred due to its reliability, scalability, and customisation that created a new market where the big companies fight to provide

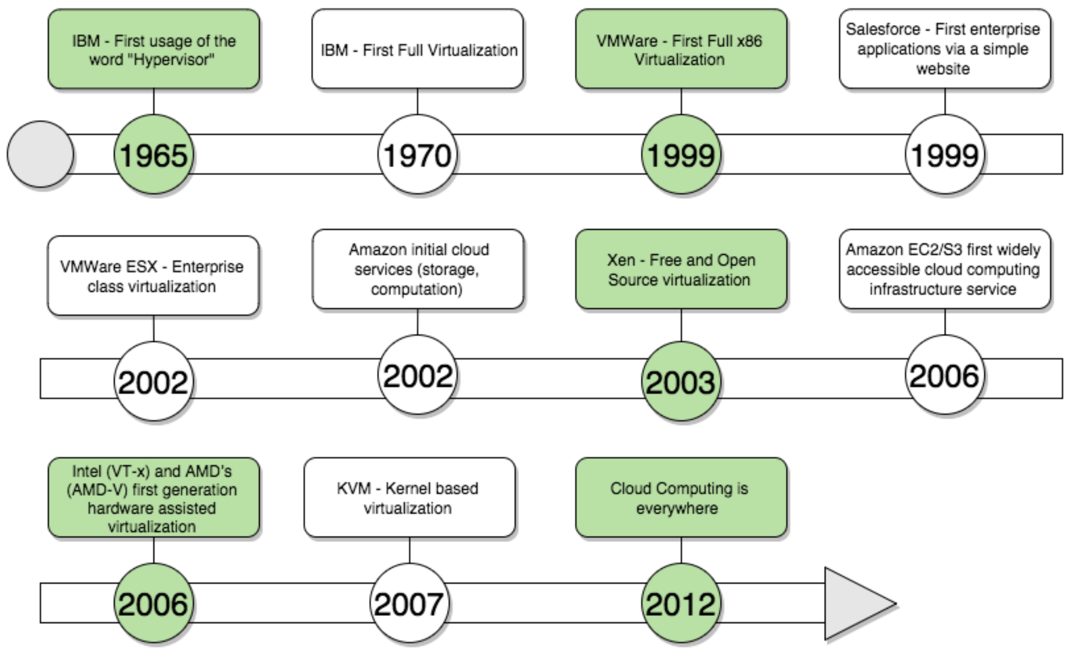

IBM - First usage of the word "Hypervisor"
1965
IBM - First Full Virtualization
1970
VMWare - First Full x86 Virtualization
1999
Salesforce - First enterprise applications via a simple website
1999
VMWare ESX - Enterprise class virtualization
2002
Amazon initial cloud services (storage, computation)
2002
Xen - Free and Open Source virtualization
2003
Amazon EC2/S3 first widely accessible cloud computing infrastructure service
2006
Intel (VT-x) and AMD's (AMD-V) first generation hardware assisted virtualization
2006
KVM - Kernel based virtualization
2007
Cloud Computing is everywhere
2012

Figure 2.1: History of Virtualisation

the best services. Examples of services can go from complete virtual machines to simple email clients or file hosting services.

The cloud computing services stack can be classified and organised into three major offerings [3]: (a) Infrastructure as a Service (IaaS) (also referred as Hardware as a Service (HaaS)), (b) Platform as a Service (PaaS), and (c) Software as a Service (SaaS).

Infrastructure as a Service is the type of service that offers complete virtual machines in the cloud, meaning that the client has remote access to the virtual machine and also some simplified interface with commands such as start and stop the machine. This service is the closest to having a physical machine but with the flexibility to change all its characteristics through a user or programming interface.

Platform as a service is defined for providing execution run times and software development tools hosted in the provider's infrastructure and accessible through some kind of portal. A well-known example of PaaS provider is Google with the GoogleApps[1].

Software as a service is the most common and provides some piece of software hosted in the provider's machine that is accessible through a web page, and goes from any web email client to team management software. This kind of service attracts more and more clients each day because all is hosted in the provider's infrastructure and is accessible everywhere with minimal setup.

[1]GoogleApps: https://www.google.com/work/apps/business/

Advantages

Cloud computing distinguishes itself from other services for various advantages [2, 4].

- **On-demand provisioning and elasticity:** Computing clouds provide on-demand resources and services. The client can create and customise the services, from software installation to network configuration.
- **QoS/SLA guarantees:** The computing environments provide guaranteed resources, such as hardware performance like CPU speed, I/O bandwidth, and memory size, specified through a SLA defined with the user. The SLA contains the minimum requirements the service has to fulfill and some penalties in case of any violation.
- **Legacy system support:** Virtual machines support any kind of system or software allowing the user to run legacy systems side by side with other systems in a simple and easily manageable environment.
- **Administration simplicity:** The lack of maintenance in a cloud setup favours the creation of complex infrastructures with minimal technical support, allowing the client to focus more on the product instead of maintaining the infrastructure.
- **Scalability and flexibility:** The scalability and flexibility are the most attracting features of the cloud, especially because they release the client from all the burdens of maintenance. The provider usually has a geographically distributed infrastructure with easy auto-scaling services that can adapt to various requirements and potential large number of users.

Disadvantages

The flexibility and simplicity of the cloud also have some disadvantages.

- **Overhead:** Virtualisation has evolved greatly during the past years, especially in terms of performance when compared to native physical execution. The developments in software and hardware assisted virtualisation made them very close, although in critical high performance applications virtualisation still has some overhead.
- **Resource interference:** One of the most common issues with virtualisation is to have many virtual servers within the same physical machine. Since it is very hard to completely separate their influence on each other, some of them will suffer from performance decrease.

Relevant Cloud Computing Services

There are lots of cloud service providers on the market, giving easy solutions for computing infrastructure management. Some of them are well-known and offer very complete suites of services. Providers such as Amazon, Microsoft, and Google offer mostly hosted solutions while OpenStack offers an installable version targeting on-premises installation.

Amazon Elastic Compute Cloud (EC2) [2] is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. It provides a web interface that allows the user to have complete control of the computing resources running on Amazon's proven computing environment. EC2 allows to quickly scale any system within minutes as the computing requirements change. The service is paid by usage, meaning that the client pays for each resource, such as computing time, storage, and traffic. Amazon provides its clients with tools to build failure resilient applications and to isolate themselves from common failure scenarios without minimal maintenance from the client.

Microsoft Azure [3], Microsoft's cloud platform, is a growing collection of integrated services, such as computation, storage, data, networking and applications. It offers a service list similar to Amazon EC2 but with a higher focus on large enterprise clients, offering easier integration with in-premises installations and more technical support.

Google Cloud Platform Compute [4] Google's cloud platform is the new alternative offering infrastructure services and trying to challenge the other solutions with fine-grained prices and better security measures and a easier to use interface.

OpenStack [5] is an open-source alternative software for creating private and public clouds. The software consists of a set of parts that control pools of processing, storage, and networking resources in a datacenter. It can be managed through a web-based dashboard, command-line tools, or an Application Programming Interface (API). It works with popular enterprise and open source technologies, making it ideal for heterogeneous infrastructures. Hundreds of technology companies around the world rely on OpenStack to manage their businesses every day, reducing costs and helping them develop features faster. One of its strong points is the big ecosystem which helps it grow and keep running with other big players.

[2]Amazon Elastic Compute Cloud (EC2): `https://aws.amazon.com/ec2/`
[3]Microsoft Azure: `https://azure.microsoft.com/en-us/`
[4]Google Cloud Platform - Compute Engine: `https://cloud.google.com/compute/`
[5]OpenStack Software: `https://www.openstack.org/`

2.1.2 Virtualisation

The concept of virtualisation had its origins in the late 1960s, when IBM was investigating a way of developing robust machine-sharing solutions [5] and developed M44/44X, the first system close enough to a virtual machine proving that virtualisation is not necessarily less efficient than other approaches.

MULTICS [6] was an important time-sharing solution created at MIT in the late sixties. It was developed with a goal of security and one of the first to have one stack per process and hierarchical file system.

TSS/360 [7] released by IBM in 1967, was one of the first operating system implementations of virtualisation. The system could share a common physical memory space, it would only run a single kernel, and one process launched by one processor could cause an interruption in another. Had special instructions to implement locks on critical sections of the code and had a unique implementation of a table driven scheduler that would use parameters such as current priority, working set size and time slices number to decide the priority of a thread.

Virtualisation was the solution for organisations or individuals to optimise their infrastructure resource utilisation and, at the same time, simplify datacenter management.

Today, every cloud company uses virtualisation in their datacenters to make abstraction of the physical hardware, creating large services to provide logical resources consisting of CPUs, storage or even complete applications, offering those resources to customers as simple and scalable solutions for their problems.

Virtual Machine Concept

A Virtual machine (VM) is a specialised software implementation that, like a physical computer, can run an operating system or execute programs as if they were running in their native system [8]. The virtual machine contains all the same files and configurations of a physical machine, including the virtual devices that map the functionality of the physical hardware.

There are two major classes of virtual machines: (a) System VMs, and (b) Process VMs (High Level Language VMs).

System virtual machines are a complete copy of a system so they can emulate an existing architecture, and run a full Operating System (OS). This kind of VM is built with the purpose of having multiple OSs running in the same physical machine, to have easier setup, minimise use of computing resources, and offer better confinement between applications.

Process virtual machines are less complex and platform-independent environments, designed to execute normal applications inside a host operating system.

This kind of VM enables programs in the same language to be executed in the same way on different systems. Common examples are the Java Virtual Machine (JVM) and the Common Language Runtime for .NET.

Challenges in Virtualisation

As per Popek and Goldberg, there are three required properties for an architecture to be virtualisable [8].

- **Efficiency Property:** Provide the ability to execute innocuous instructions directly on hardware bypassing the hypervisor.
- **Resource Control Property:** Hypervisors should be able to completely control the system. When the guest operating systems tries to access resources, the access should be routed through the hypervisor that must verify all the requests and avoid unauthorised accesses.
- **Equivalence Property:** Any program running on top of the hypervisor should perform in a way that is indistinguishable from the case when the hypervisor does not exist.

Hypervisor – core of system VMs

The hypervisor, also called Virtual machine monitor (VMM), is the physical machine and the software that creates and controls the virtualisation, allowing multiple isolated guests to run concurrently within the same physical machine. It permits two or more operating systems to share a common computing system [9] and works as a control interface between the host operating system running, on the physical machine, and the guest virtual machines.

There are two types of hypervisors: (a) Type 1, native or bare-metal hypervisors, are executed in the physical machine; (b) Type 2, hosted hypervisors that execute from within the host OS as applications on an unmodified OS. Type 1 is the original model developed by IBM in the 60s [7, 5] and some implementation examples are Xen[6], KVM[7], Oracle VM[8], Microsoft Hyper-V[9] or VMware ESX [10], while for Type 2 we have solutions like VMware Workstation[10] and VirtualBox[11]. The hypervisor

[6]Xen http://www.xenproject.org/

[7]KVM http://www.linux-kvm.org/page/Main_Page

[8]Oracle VM http://www.oracle.com/us/technologies/virtualization/oraclevm/overview/index.html

[9]Microsoft Hyper-V http://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx

[10]VMware Workstation https://www.vmware.com/products/workstation

[11]VirtualBox https://www.virtualbox.org/

plays a very important role in cloud environments, since it is the key piece to manage the whole service infrastructure.

Optimisations for performance and efficiency

To perform better virtualisation, several optimisations were developed in hypervisors. For handling sensitive and privileged instructions to virtualise the CPU, there are currently three alternative techniques [11].

Full virtualisation or virtualisation using binary translation and direct execution is the approach used by VMWare. This approach does not rely on OS modifications to help execution. Instead, it translates kernel code replacing the instructions that cannot be directly translated to sequences of instructions that have the same intended effect on the virtual hardware. All the instructions are translated on the fly and the results are cached for future use. The implementation is optimised to execute user-level code directly on the processor.

This combination of binary translation and direct execution provides a complete abstraction for the guest OS to be completely decoupled from the underlying hardware. In this kind of virtualisation, the guest OS is not aware that it is being virtualised. Full virtualisation offers the best security and isolation for virtual machines and simplifies migration and portability, as the same guest OS instance can run virtualised or on native hardware.

Paravirtualisation or operating system assisted virtualisation refers to virtualisation that relies on modifications on the OS to improve performance and efficiency. The Xen open source project is an application example of this technique that is widely used by services such as Amazon EC2.

As opposed to full virtualisation, the non-virtualisable instructions are replaced with hypercalls that communicate directly with the virtualisation layer hypervisor. Other hypercall interfaces are also added for critical kernel operations, such as memory management, interrupt handling and time keeping. Since building sophisticated binary translation support necessary for full virtualisation is hard, modifying the guest OS to enable paravirtualisation is the most cost/effort effective solution.

Hardware-assisted virtualisation was created to solve the problems in previous solutions, hardware vendors are rapidly embracing virtualisation and starting to develop new features to simplify virtualisation techniques. Both Intel and AMD started working, in 2006, on CPU generations VT-x and AMD-V, respectively, with the objective to target privileged instructions with a new CPU execution mode feature, to allow the hypervisor to run in a new root mode below the most privileged protection domain (also known as Ring 0). This allows privileged and sensitive calls to trap automatically to the hypervisor, removing the need for either binary translation or paravirtualisation.

Due to high hypervisor-to-guest transition overhead and a rigid programming model, VMware's binary translation approach currently outperforms first generation hardware assist implementations in most circumstances. Still, some 64-bit instructions are used by VMWare solutions.

Memory hashing was a novel technique to share memory pages developed by VMware for their ESX Server. This technique [10] allowed to get around modifications to guest operating system internals or to application programming interfaces. The basic idea is to identify page copies by their contents. Pages with identical contents can be shared regardless of when, where, or how those contents were generated.

This approach brings two main advantages, (1) no need to modify, hook, or even understand guest OS code, (2) easy identification of possible pages to share. To avoid $O(n^2)$ search (all pages against all pages), it uses hashing as key to identify pages with identical contents efficiently. After positive hash match, it does a full comparison of the page contents to confirm that the pages are in fact identical and not just a hash collision. The page is only used for read and any subsequent attempt to write to the shared page will generate a fault, transparently creating a private copy of the page for the writer.

2.2 Virtual Machines Scheduling

Scheduling is a very important part of the cloud environments, it is the process in which the provider organizes its infrastructure and where all the process behind the service is defined. This is described by the scheduling algorithm.

2.2.1 Scheduling Algorithms

The scheduling algorithm is a program expressed as a set of well defined rules for determining the most adequate choices of where to allocate a new virtual machine. The scheduling process is very important in the cloud computing environment, because it is the way it efficiently manages the resources. All the inner factors like speed, utilisation percentage, and efficiency of the resources depend primarily on the kind of the scheduling algorithm being used in the system.

The scheduler can act upon a large variety of factors, such as CPU usage, available memory, or energy consumption. Various issues arise from scheduling multiple heterogeneous systems, the predictability is usually very low, and the algorithm has a difficult job managing allocations.

Scheduling algorithms are characterised by three parts: the input which defines their initial state, the policies they use to achieve their objective, and, finally, their final choice. The efficiency of job scheduling has a direct impact on the performance of the entire cloud environment and many heuristic scheduling algorithms were used to optimise it. Scheduling in cloud computing environments can be performed at various levels such as workflow, VM level, or task level.

Figure 2.2 describes the participants in the scheduling process and the flow of the requests. Upon the arrival of a request, the scheduler will try to find a physical machine where it can allocate the requested resources.

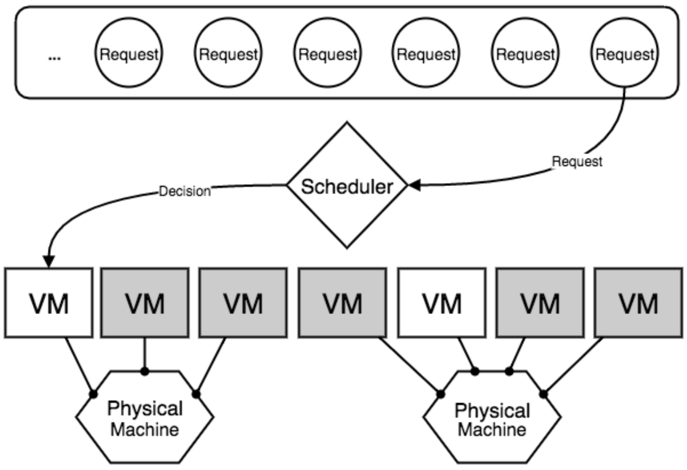


Figure 2.2: Virtual machine scheduling overview

Scheduling Phases

The scheduling process can be divided in three major phases [12]: resource discovery, system selection, and allocation.

Resource Discovery is the first phase and consists of searching and locating resource candidates that are suitable for allocating the VM. The dynamic and heterogeneous nature of the VMs makes efficient resource discovery a challenging issue. The next step is the application requirement definition which consists of using the user-specified requirements and filtering the resources to eliminate the resources that do not meet the minimal requirements.

System Selection, the second phase, has the objective of selecting a resource where to schedule the VM. In this phase, some algorithms gather dynamic information which is important to make the best mapping between allocation and resource, especially in heterogeneous and rapidly changing environments. The final selection consists of choosing a resource with the gathered information that best fits the clients needs and better fulfils its objectives such us revenue, energy efficiency, or performance.

VM Allocation is the last phase and consists of submitting the VM to allocation. To assure service quality, some precautions must be taken, such as preparing the resource to deploy the VM and as soon as it is deployed, monitor its activity, and keep track of its state. By monitoring tasks, the scheduler can conclude that a given job is not working correctly and may need to be re-schedule. The next step is to notify the user and clean any temporary files used during scheduling.

2.2.2 Algorithm Classes

The algorithms can be classified by several major parameters. In Casavant et al. [13] they suggest several classes that we grouped by purpose in three major groups: architectural, scope, and flexibility.

a) Architecture/Design

i) Local vs. Global: The scale of the scheduling algorithm defines the level at which it is done, either at the hypervisor level or at a higher level, such as datacenter or cloud level. This factor is very important in terms of architecture because it will decide the scope in which the system will do scheduling. Local scheduling allocates VMs to a single physical machine, while global scheduling allocates VMs to multiple physical machines, being able to optimise a system-wide performance goal. Considering what was said before, we can conclude that cloud scheduling for IaaS is mostly global.

ii) Centralised vs. Distributed vs Hierarchical: The scheduling can be done by a single node or multiple nodes in the system, centralised, and distributed respectively. On the centralised approaches, there is only one scheduler for the system, making it easier to monitor and make decisions about the current state. Another advantage of centralised scheduling is the easy implementation of the scheduler. Similarly to other types of centralised types solutions, we have a single point of failure, lack of scalability, and fault tolerance.

The decentralised solution consists of having no central master, by creating a communication network between the lower level schedulers to make decisions. A very common approach [14] is to use a Peer-to-peer network that communicates using the Chord algorithm [15].

Hierarchical are very similar to centralised solutions but with several lower levels of delegation. The levels of delegation split the computation and allow for more scalable and fault-tolerant solutions, but not as fault-tolerant as the distributed approach.

iii) Immediate vs Batch: In the immediate approach, VMs are scheduled as they enter the system, using the system's scheduling algorithm. Batch allocation runs in scheduled intervals of time and VMs are grouped in batches to be scheduled as a group. This technique helps to do a better allocation, since we have more information to do the distribution allowing better matching in the long run.

b) Scope

i) Approximate vs. Heuristic: An approximate approach is used when we have a solution evaluation function. This function grades the solutions, and by searching only a subset, it may be possible to find a good candidate. In large solution spaces, this approach may have a good impact, since it avoids full searches that could take much more time. The heuristic tries to solve the same problem with approximation, search through big solution sets, but it does not have the same guarantee of success. It is used to obtain faster results.

ii) Load Balancing: Load balancing is a technique used to balance the load on the system, in a way that allows the same performance on all nodes. This solution is more effective in systems where the nodes are homogeneous, since it allows making decisions with more precision. Information about the load can circulate in the network periodically or be sent on-demand. With this information, the nodes coordinate the process of removing work from heavily loaded nodes and placing it at lightly loaded nodes.

c) Flexibility

i) Static vs. Dynamic: In this class, the distinction is made based on the time at which the scheduling or assignment decisions are made. In static scheduling, the information relative to the system configuration is available before allocation, meaning that the same allocation would be made within a certain period of time. In dynamic scheduling is common for resources to join and leave the setup or even for the resource usage policies to change.

ii) Adaptive vs. Non-Adaptive: Adaptive approaches take into consideration the history of the system. Measures such as previous and current behaviour of the system are used to better assign the VM. This kind of scheduler usually has some logic that makes it make decisions based on the information it is gathering. They can, for instance, change priorities of some parameter values if they are perceived as being wrongly influencing the system. In contrast with adaptive schedulers, non-adaptive schedulers do not modify their behaviour based on system history. The same parameter will always weight the same regardless of system history.

2.2.3 Classical Algorithms

Over the years, a large number of algorithms has been developed and described in the literature. We classified them in five categories (by their goal): capacity driven, deadline/goals, interference free, and economics/utility economics. Next, we provide an overview of all the categories giving some descriptions and examples of existing implementations.

Capacity driven:

Capacity driven algorithms are a very common type of algorithms, usually very simple and straightforward. They were the first kind of scheduling policy, mainly because of their simple logic and implementations. With the evolution of the cloud, more complex scheduling policies started to arise. There are several types of algorithms following this type of approach, such as Greedy, Round-Robin, and Bag-of-Tasks.

Greedy is one of the simplest algorithms. The logic behind this type of algorithm is to find a match in the resources of the system where the requested VM can fit. The first node that meets the requirements is identified and the VM is allocated there.

This means that the greedy algorithm exhausts a node before it goes on to the next node. As an example, if there are 3 nodes, the first node usage is 60% while the other two are underloaded, if there are two VMs to be allocated, both are allocated to the first node. This might result in the increase of its usage to high values while the other two nodes will still be underloaded. The main advantage is the simplicity: it is both simple to implement and allocate VMs. One important disadvantage is the low utilisation and distribution of the available resources.

Round Robin is also a very simple scheduling policy, but in contrast with the Greedy approach, it mainly focuses on distributing the load equally between all the nodes. Using this algorithm, the scheduler allocates one VM to a node in a cyclic way. The scheduler loops through the nodes, one after the other assigning VMs. This process is repeated while all the nodes have not been allocated at least once, after that the scheduler returns to the initial node and restarts the process.

For example, if there are 3 nodes and 3 VMs to be allocated, each node would allocate one of the VMs, equally distributing amongst them. One advantage of this algorithm is that it utilises the resources in a uniform way, helping to balance the system's load. However, the algorithm can waste more power than needed if all the machines and under light load.

Weighted Round Robin is an optimised version of the previous algorithm which takes into consideration the weight of the tasks and the load of the machines. Instead of equally distributing the number tasks among the machines, it equally distributes their weight, contributing for better performance in heterogeneous sets of tasks.

Bag-of-Tasks (BoT) is another type of scheme common in problems such as image rendering and software testing. In BoT jobs, tasks are usually independent and, thus, they can be executed in parallel since they have no need for intercommunication or for sharing data.

The main problem in this type of scheduling is the number of hosts to allocate since usually the total number of tasks is unknown, creating a lot of difficulties for optimisation of cost or performance. Silva et al. [16] use adaptive heuristics to optimise this process and to predict the number of tasks. The approach is based on adapting the parameters at each task completion, allowing the system to reduce execution and idle time.

Deadlines/SLAs:

SLAs are a very common way for a user to define the required Quality of Service (QoS) parameters for a requested cloud service. QoS are parameters which represent constrains or bounds that are related to the provided service. QoS usually appears related to aspects on computer networks such as service response time. In Cloud environments, there are some different QoS aspects to consider, such as deadline, execution time, and overhead. This type of algorithm tries to maximise the parameters to meet the QoS defined previously.

Abrishami et al. [17] propose an evolution of Partial Critical Paths (PCP) [18] which aims to minimise the cost of workflow execution while meeting a user-defined deadline in grids. PCP divides the deadline in several tasks and assigns them to nodes starting by the exit node. The new solution proposes a one-phase algorithm which is called IaaS Cloud Partial Critical Paths (IC-PCP), and a two-phase algorithm which is called IaaS Cloud Partial Critical Paths with Deadline Distribution (IC-PCPD2). Both the solutions try to fit the previous grid algorithm in the cloud paradigm, having in mind several differences, such as on-demand resource provisioning, homogeneous networks, and the pay-as-you-go pricing model.

The IC-PCPD2 algorithm replaces the previous assigning policies by the new pricing model and tries to assign the tasks to currently running machines: it only launches another as a last resort.

In contrast, IC-PCP tries to find an available or new machine to assign the entire path at once.

Particle Swarm Optimization (PSO) [19] bases its evaluation on the velocity of a task, which is represented as a vector (magnitude and direction). This velocity is determined based on best position in which the particle has been and the best position in which any of the particles has been. The algorithm will continue to iterate until the fitness function objective is considered to be good enough for the defined objectives.

Interference-free:

Resource interference on OSs is one of the hardest problems impacting virtualisation. Either because VMs are being under-provisioned for cost effectiveness, because we underestimate the VM needs, or because the workloads are very incompatible and clash in terms of memory access requests. While co-locating virtual machines we aim to improve resource utilisation, but this ends up resulting in performance interference between the VMs. The most common practice to reduce this effect is over-provisioning resources to help avoid performance interference. This practice is not very interesting because it goes against the main objective, optimise resource usage, simply because we will be giving more resources than VMs need and they will be under-utilised.

Recent work still relies on static approaches that suffer from several limitations due to assumptions about application behaviour that are not certain *a priori*. **DeepDive** [20] is a three-phase approach to the interference problem. It starts by using the hypervisor to generate warnings about possible interference, then it starts a extensive analysis and finally it reassesses the distribution of the VMs. It transparently deals with interference by using low-level metrics, including hardware performance counters and readily available hypervisor statistics about each VM.

Stay-Away [21] is a novel approach to the interference-free problem. Providing a generic and adaptive mechanism to mitigate the performance interference on responsiveness-sensitive applications, it is able to reduce the interference. This solution continuously collects information about the states of execution and tries to predict and prevent any transition that may cause interference by proactively throttling task execution.

Economics Driven/Utility economics:

In modern society, essential services, also called utilities, are commonly provided in a way that makes them available to everyone who can pay. Services such as water, electricity, gas, and telephony are classified as essential for daily life routines. These utility services are accessed so frequently that they need to be available whenever the consumer requires them, at any time.

Utility is a concept that evaluates the satisfaction of a consumer while using a service. In a Cloud environment, utility can be combined with QoS constrains, in order to have a quantitative evaluation of a user's satisfaction and system performance. This concept allows clients to be able to pay for the services based on their usage and quality of the service provided [22].

To achieve this, cloud providers cannot continue to focus their systems on the traditional resource management architecture that treats all service requests as being of equal importance. Most the works in literature treat users based only on SLA parameters, and this means that two users with different characteristics, such as stricter deadlines, but with similar SLAs have equal importance for the service provider. Instead, providers need to provide utility based services, that can achieve equilibrium between demand and supply, providing incentives for consumer-based QoS resource allocation mechanisms that differentiate service requests based on their utility.

Utility-based services provide more flexibility for both clients and providers but usually require the adoption of some kind of economic or cost-theoretical model.

Cloudpack [23] tries to disrupt static pricing models for resources. A typical datacenter has different costs, influenced by the cost of the energy, the cooling strategies used, and the current demand of the service. This framework tries to disconnect the dynamic cost incurred by the providers and the fixed price paid by a customer, with the ability to formally express workload flexibilities, using Directed Acyclic Graphs (DAGs).

It is very hard to minimise the probability of failure in tasks without losing revenue by over-provisioning the infrastructure. Macias et al. [24] maximise the fulfilment rate of the SLAs by considering risk of failure in the decision process. Clients choose the SLAs based on several classes of risk: the higher the risk, the lower the price. They are, however, unable to explicitly select the VM or set of VMs to degrade.

Morshedlou et al. [25] propose two user-hidden characteristics used to create a proactive resource allocation approach. Willingness to pay for service and willingness to pay for certainty aim to decrease impact of SLA violations. The presented method decides which VMs should release resources, based on each client's willingness to pay. This approach is similar to Partial Utility SLAs [26], but they assume, in the solution, that some amount of SLA violations will occur due to the release of all the resources. They also assume VMs of homogeneous types, which is very uncommon in cloud deployments.

2.3 Energy and Environmental Awareness

As cloud computing evolves and establishes its paradigm, concerns about energy waste and environment awareness arise. Cloud computing and energy are closely related. The energy efficiency in the cloud became an issue and a large number researchers has been working on it in recent years.

2.3.1 Scheduling Aspects

Hardware keeps evolving [27] and with new technologies, such as low–power CPUs, solid state drives and other energy efficient components, the energy footprint got smaller. All these improvements were not enough and, for that reason, there has also been a high amount of research done trying new software approaches, such as energy efficient scheduling and resource allocation to reduce this problem.

Green Scheduling

Green scheduling is new paradigm for cloud computing infrastructures that is concerned about energy waste and environment awareness. Energy-aware approaches can be split into two categories [28], characterised by how they want to reduce energy. Power-aware [4, 28, 29, 30] and Thermal-aware [31] focus on computer and cooling system power reduction, respectively.

In **Power-aware** solutions, the target is the physical machine, and the algorithms usually aim for aspects such as resource usage and try to maximise the performance without maximising power.

Thermal-aware solutions target reducing the emissions of heat from the computer components and, indirectly, it reduce the wasted energy in cooling the machines. Although the thermal maintenance of the datacenters does not seem to be directly related to energy efficiency, the cooling in these computing facilities consumes large amounts of energy.

Figure 2.3, inspired in [28], groups the areas of study for energy aware algorithms and highlights the topics in which we focus our solution.

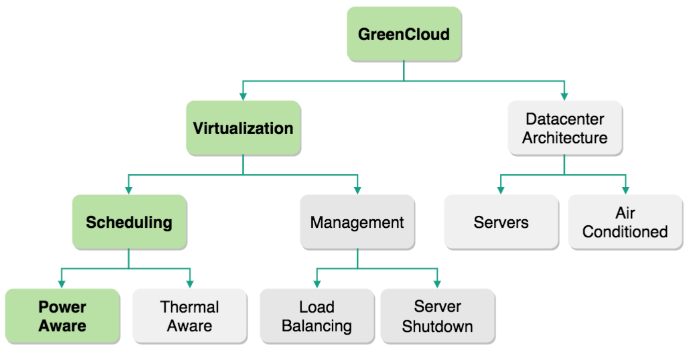


Figure 2.3: GreenCloud: Green items represent areas in range for this thesis

Power-Aware Scheduling

Power-aware scheduling is the focus of our work and also a very broad research area. From the power of the CPU, to the ventilation of the machines, there are several ways to approach the study of energy dissipation in a physical machine. As explained before, some works focus on resource management and try to optimise their usage, reducing the number of machines on-line, being able to cut a big part of the energy spent.

Usually, approaches focus especially on CPU and how it is being used, while others also take into consideration RAM, and even the communications between machines. With techniques such as load balancing, providers try to distribute machines in the datacenters in such a way that, not only the workloads complement each other and do not keep competing for memory access, but also communication between them is reduced.

Virtual machine consolidation is crucial for power management, but it is always a hard problem to solve. Gather any VMs in the same physical machine to avoid spending energy is not possible because we will overcommit its resources. The best solutions try to gather all available information, such as SLAs and also previous usage footprints, to better allocate the jobs.

2.3.2 Energy Aspects

The way machines consume energy can be classified into two categories, based on the fact of the energy consumption changing over time or being constant all the time [4].

Static energy consumption is the part of the energy used by a machine without load. This is the energy used by the components when they are in idle mode, not performing any kind of work.

Dynamic energy consumption, in contrast, is calculated in terms of the proportion of resources being utilised by the machine. In this case, we are measuring the energy used by the components while doing some work.

Another important factor about energy is its sources, not only because it can influence the price but also because the origin can impact on the environment.

Based on the SLAs defined with the client, the provider can improve energy usage, for example, by delaying the schedule of a VM for some time, waiting for a time when the energy being used comes from clean sources.

Other approaches can also use daytime to decide: during the night, the energy is usually cheaper because the demand is smaller and that can be a good reason to delay a job for a client that does not have a deadline. This kind of approach is not common, but is definitely a good opportunity to reduce the price for the client and still reduce costs for the provider.

2.3.3 Efficiency Aspects

Multiple approaches have been developed and described in the literature concerning energy waste and environmental footprint. Next, we characterise several relevant solutions with interest for our work.

Dynamic Voltage Frequency Scaling

Some of these approaches' primary target is dynamic voltage frequency scaling (DVFS)[4, 32, 31, 33]. DVFS dynamically scales the processor frequency according to the global CPU load and regardless of the VM local loads, and hence, it helps reducing power consumption.

This kind of approach has several problems, since it targets a very sensitive part of the machines, the CPU. The classical example is the heterogeneous environment where two virtual machines have opposite needs, for example, one needs 200MHz and the other 1000MHz. The common scheduler will probably try to find a middle spot, 600MHz, but that is far from good for the second machine. The first is getting the triple needed while the second if almost at half of the needed computing power.

To solve this, some solutions try to distribute workloads by their profile and are able to have CPU similar VMs in the same physical machine, while others will still reduce the power but will give more execution time to the VMs being underloaded.

Energy Efficiency Algorithms

Efficiency is a major goal in scheduling, especially when even a minimal improvement can lead to major effects in the whole system. The typical factors that are targeted in terms of efficiency are resources and energy.

Younge et al. [28] try to achieve maximum energy efficiency by combining a greedy algorithm with live migration. It minimises power consumption within the datacenter by allocating in each node as many VMs as possible. It runs through each VM in the queue waiting to be scheduled and the first node in the priority pool is selected if it has enough virtual cores and capacity available for the new VM.

Beloglazov et al. [34] present a decentralised architecture of a resource management system for cloud datacenters that aims to use continuous optimisation policies of VM placement. They look at factors such as CPU, RAM, network bandwidth utilisation, and physical machines temperature, to better reallocate machines and improve overall efficiency.

Beloglazov et al. [29] detect overutilisation and underutilisation peaks to migrate VMs between hosts and minimise the power consumption in the datacenter.

Von et al. [4] use a batch scheduling approach that is based on DVFS. VMs are allocated starting with the ones with more CPU requirements and in each round it tries to reduce frequencies to reduce power consumption.

EQVMP (Energy-efficient and QoS-aware Virtual Machine Placement) [30] is a solution with three objectives, inter-machine communication and energy reduction with load balancing. They group the machines to reduce communication and the allocation is done by finding the machine with the resource availability closer to the request. By controlling the information flow, they manage to migrate VMs and keep improving the disposition of the VMs.

ThaS (Thermal-aware Scheduler) [31] is different from the previous solutions, it is thermal-aware. Their scheduling policies take into consideration the temperature of the machines and, together with CPU frequency, they apply DVFS and load balancing techniques. Their main limitation is the model they use for CPU temperature which only works for single core CPUs.

On Appendix A we have a summary of the preceding algorithms and their characteristics.

All these approaches have tried to reduce energy consumption and improve resource usage, but none used the concept of DVFS in conjunction with partial utility in a hierarchical architecture. *EcoScheduler* does DVFS DVFS scheduling with resource awareness (mainly CPU performance) and tries to achieve maximum request satisfaction.

Chapter 3

Solution

The algorithms presented in Chapter 2 have several problems that prevent them from being scalable, energy efficient and more performant. The following list describes those problems and proposes how they were solved in our solution.

- **Centralised allocation** Most studied solutions approach the allocation problem with a centralised entity that is responsible for all the work of processing the request until it is assigned in a host. In a large size datacenter (e.g. tens of thousands of hosts) with very active client base (e.g., hundreds of thousands requests per minute) having only one entity handling requests is going to be a significant bottleneck. One type of solution to work this problem is to create a fully distributed architecture with multiple nodes working as entry points. A different approach is to create a hierarchical datacenter which is less complex to maintain and only losing some fault-tolerance if we compare it with a more complex fully distributed a architecture.

- **Aggressive CPU scaling** There are several different ways to use DVFS to control energy efficiency on the host. Some algorithms try to keep the frequency lower to consume less energy, but this leads to slower executions; others try to take it to the maximum spending more energy but with faster execution. A good balance between the two is OnDemand, as seen in [35], which increases the frequency to the maximum when work arrives, and then reduces it if for an established amount of time is being under used. Our approach uses a similar idea but we do not scale to the maximum, instead we scale to the step that can handle the work. This allows for less jumps and a more consistent execution.

- **Live allocation** When allocating, we can consider a several different metrics, from the CPU, to the storage, or even outside values such as the price of electricity in the at a specific moment. Most algorithms collect and process this data when they are doing the allocation; this is a good idea if one requires live data and wants the values to be precise, but also do not mind to be slower. Our solution aims for performance and scalability and, for that reason, we process the information when it changes and try to keep up to date values that allow faster decisions.

In our solution, we organise the system as a structured hierarchical network headed by the Global Scheduler (GS)[1] and where the datacenter is partitioned into sectors that aggregate several physical machines.

At the datacenter level, we have the main sector containing the GS that carries out a first level arbitration among the sub-sectors. In each sector, there is a Local Scheduler (LS) that is responsible for all scheduling operations regarding the comprised physical machines. Each LS will implement our energy-utility-based scheduling algorithm.

In Section 3.1, we describe the three layers of our application with a use case of the proposal, followed by a description of the complete architecture of our solution in Section 3.2 and the data structures used are explained on Section 3.3. The relevant metrics to evaluate our solution are reviewed in depth in Section 3.4, followed by the pseudocode of the algorithms explained in Section 3.5.

3.1 Use case

The architecture can be divided into three layers: client layer, hierarchical layer, and physical layer, as depicted in Figure 3.1.

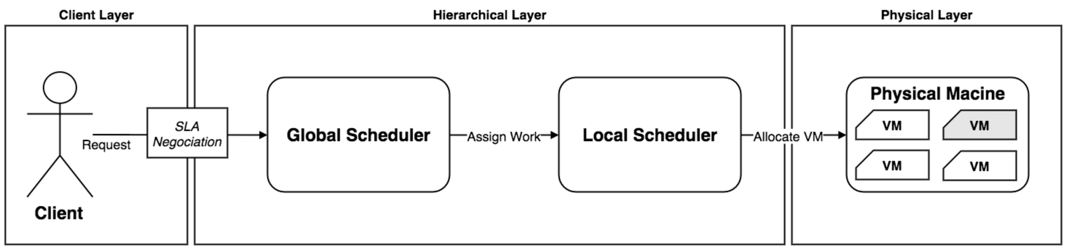


Figure 3.1: Use case scenario

Figure 3.1 describes all the steps in the high-level process for reserving VMs. The client layer, which includes all the clients willing to reserve VMs in our system, communicates with the hierarchical layer via the GS. In the second layer, the request will be processed and a LS assigned having in consideration of the established SLAs and also the energy and resource usage objectives of the system. This is accomplished

[1]The Global Scheduler could be replicated for availability purposes, but we left that out of the design for simplicity.

by passing the request to the selected LS, which will then allocate a VM in the infrastructure layer. After this workflow is completed, the LSs will keep monitoring the physical machines to assure Quality of Service (QoS).

3.2 Distributed Architecture

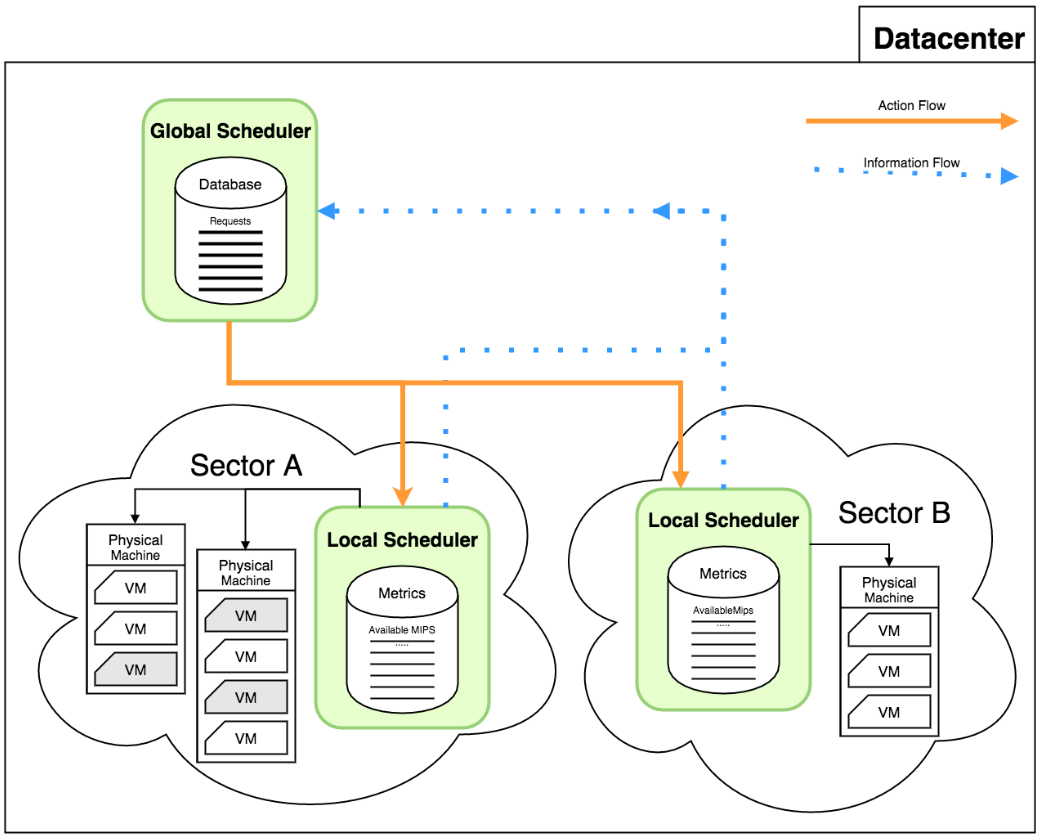


Figure 3.2: High-level architecture

A high-level description of the proposed solution's architecture for *EcoScheduler* is depicted in Figure 3.2. The architecture is composed by two main entities, the GS and the LS. For moderate size clusters (e.g., 1000 to 5000 hosts), the architecture can be composed of only two levels of hierarchy. The first one is composed by the GS followed by the LSs that manage their sectors of physical machines. If the system needs more partitioning, we can achieve it simply by creating sub-levels of GS that delegate scheduling to the next level of the hierarchy.

Each level in the architecture communicates with the upper level to provide information about their state changes. This information can be classified in two categories: static and dynamic.

Static information does not change over time. There are several examples of static information such as operating system, processor (range of frequencies and respective voltages), number of cores, disk space, and RAM.

Dynamic information is all the remaining data that changes over time. Some examples of dynamic information are: current CPU frequency (per core), CPU occupation (per core), number of allocated VMs, free disk space, free RAM.

The information about the machines is requested by the LS after a change in the machine, either an allocation or a deallocation. Then, LSs send an information update about their state (a summary of the machines state) to the GS. The GS processes all the information and creates a summary. With summaries of the metrics necessary to decide on the allocation, we can do faster decisions, scale our allocation and achieve faster allocation times, as opposed to real time metrics. The summary includes average energy efficiency of the sector, maximum CPU available, and maximum available memory (disk and RAM).

The power consumed by computing nodes in a datacenter consists of the consumption by the CPU, storage, and network communication. In comparison to other system resources, CPU consumes the larger amount of energy [36, 37, 38]. Hence, in this work we focus on managing its power consumption and efficient usage. Recent studies, such as [32, 33, 38], show that an idle server consumes approximately 70% of the power consumed by a server when running at full CPU speed, justifying that servers should be turned off to reduce total power consumption as soon as possible.

As mentioned before, our system has two types of scheduling: global scheduling and local scheduling. Local scheduling happens in all the nodes that are leaves in the hierarchy. At this level the algorithm will allocate the VM in a host. The upper nodes in the hierarchy (global schedulers) are considered global schedulers since they work over summaries of the information.

3.3 Data Structures

The scheduling is based on the information collected by the schedulers, about the sectors in the GSs, or the hosts in the LSs. In our solution, we take into consideration several characteristics of the hosts, such as CPU usage, number of CPU cores, RAM, and available bandwidth.

Our main data structure is a linked list of the sectors (or hosts) sorted by the average energy efficiency metric, as depicted in Figure 3.3. This structure is easier to maintain, as opposed to a list, and helps finding the most efficient sector, with the minimum resources needed to fulfil the SLAs, faster than other data structures used in works such as [35].

We use the linked list to find the best sector (or host) where to allocate a VM. The lookup needs to be fast and we also want the maintainability of that list to be

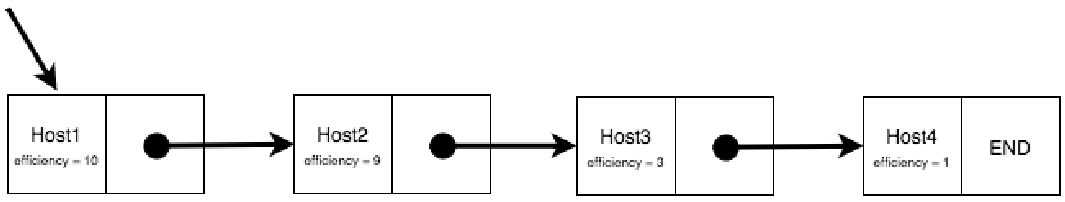


Figure 3.3: Ordered linked list

minimal. Since the list is ordered by efficiency we have a higher chance of finding possible sector (or host) of the beginning.

Each time there is an allocation of deallocation, the information summary is updated, and we have to move the changed sector (or host) to its new position in the list.

If efficiency decreased we will start by the end of the list, but if it increases we will start with the beginning. This decision helps to find the position where to insert faster, since the more efficient elements are in the beginning and the less efficient in the end.

As shown in Figure 3.4, when efficiency decreases we start by the end and will check the elements until we find one that is more efficient than the changed sector.

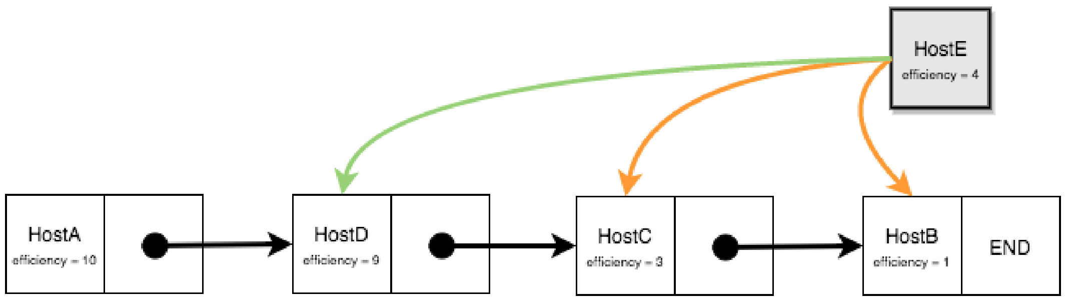


Figure 3.4: Ordered linked list insert

When we find that element, the sector being ordered is inserted after it, as depicted in Figure 3.5.

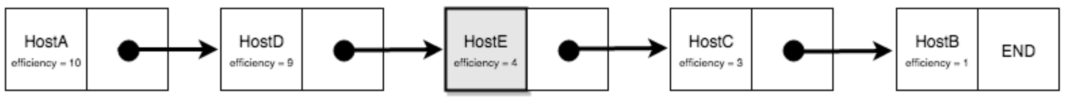


Figure 3.5: Ordered linked list after insert

Considering a list with N elements, on average, our algorithm tends to find a possible sector (or host) in N/2 operations; this may be improved by using a tree with sorted elements, approaching $O(log_2(N))$. Compared to solutions such as PowerVmAllocationPolicySimpleWattPerMipsMetric and PowerVmAllocationPolicyDVFSMinimumUsedHost described in [35] which need to search all the N elements to make a decision.

Only searching a subset of the elements, using summarised information, will lead to less accurate decisions. Our work aims to compensate the accuracy, by distributing the tasks to the servers in a round robin similar way. When other algorithms try to fill a machine before going to the other, we try to spread the work on the machines we have.

3.4 Metrics

Our algorithm uses two main metric types: efficiency and allocation metrics.

Efficiency metrics, more precisely, energy efficiency metrics are expressed by the best ratio Watt/MIP weighted by the theoretical CPU capacity of the host to allocate a new VM.

$$CPUPower = HostPower * \frac{CurrentCPUMIPS}{HostTotalMIPS} \tag{3.1}$$

Since we cannot measure the used power in each processing unit (CPU), we have to approximate it from the power used by the host. In Equation (5.1), we obtain the percentage of the power used by the $Host$, $HostPower$ corresponding to the mips being used the CPU we are calculating.

$$EEL = \frac{\sum CPUPower * \frac{VMsTotalMaxMIPS}{HostTotalMaxMIPS}}{CPUsNumber} \tag{3.2}$$

Then, in Equation (3.2), we retrieve the average $CPUPower$ used in the $Host$. We are measuring efficiency, so we use this metric inverted, to have higher values for more efficient $Host$s, this final value is what we call EEL, the energy efficiency level.

Metrics such as total and maximum CPU capacity, RAM, storage, and available bandwidth are used to decide on the best sector (or host) to allocate.

Regarding allocation metrics, we mostly focused our policy on the CPU, because that is the piece whose efficiency we are trying to increase. When deciding where to perform an allocation, we start by checking the CPU percentage available, and only then we check RAM, storage and available bandwidth.

3.5 Algorithms

Our scheduling algorithm takes two main properties into consideration, the Energy Efficiency Level (EEL) and the CPU Available (measured in MIPS) (CPUA). The EEL is calculated based on the power consumed by each processing unit (CPU) of the host and the available MIPS. Algorithm 1 presents the pseudocode for the global scheduling. This scheduling phase acts upon sectors in the datacenter.

Algorithm 1 Global scheduling: Approximate Best-Fit

```
Require: $sectors$ available sectors                                ▷ sorted by EEL
Require: $vm$ VM to be allocated
 1: function GLOBALSCHEDULING($sectors, vm$)
 2:     $selectedSector \leftarrow sectors.first$
 3:     $sector \leftarrow sectors$
 4:     do
 5:         if FitsCriteria($sector, vm$) then
 6:             $selectedSector \leftarrow sector$
 7:             break
 8:         end if
 9:         $sector \leftarrow sector.next()$
10:     while ($sector.hasNext()$)
11:     if $selectedSector = null$ then                       ▷ Fallback to the first sector
12:         $selectedSector \leftarrow sectors.first()$
13:     end if
14:     UpdateSectorsState($selectedSector, vm$)                     ▷ asynchronous call
15:     Allocate($selectedSector, vm$)
16: end function
```

Algorithm 1 does the first level of arbitration between the sectors based on *sectors* (list of sectors sorted by EEL). Since the sectors are already sorted, it picks the first possible sector with available resources and better efficiency level.

While iterating the list of sectors, Algorithm 2 is used to verify the availability of resources in the *host*. If all the sectors fail this check, we will try to allocate on the first sector, since it is the most efficient at the time.

As mentioned in Section 3.4, we check several metrics before selecting the sector where to process the allocation. In Algorithm 2, we compare the available resources in *host* with the resources requested by the *vm*. First, we check CPU availability and then we perform checks on the rest of the resources, such as RAM, bandwidth, and storage.

Algorithm 1 and Algorithm 2 are the algorithms used at the sector level, and are only used to select which sector will do the final allocation of the *vm* in a *host*.

Algorithm 3 is the generic algorithm for the local scheduling phase. In this phase, we are choosing the host where we will allocate the *vm*. This is very similar to the global scheduling phase. It also tries to find the first match in the ordered list, but, in this case, from the available hosts. Similarly to Algorithm 1, we check if it fits the criteria and we use exactly the same metrics as in the sectors, except that in this phase we have live data from the hosts and not a summary of the needed resources.

When an allocation is performed successfully, method **UpdateHostsState** is invoked. This method is responsible for triggering updates in the parent sectors,

Algorithm 2 Sector fits criteria for allocation of VM

```
Require: $sector$ possible sector for allocation
Require: $vm$ VM to be allocated
1:  function FitsCriteria($sector, vm$)
2:      if $sector.maxTotalAvailableMips >= vm.requestedTotalMips \wedge$
3:
4:          $sector.maxAvailableRAM >= vm.requestedRam \wedge$
5:          $sector.maxAvailableBW >= vm.requestedBw \wedge$
6:          $sector.maxAvailableStorage >= vm.requestedStorage$ then
7:           return $true$
8:      end if
9:      return $false$
10: end function
```

that will then re-calculate all the data summaries and re-order the lists.

To reduce the amount of data stored in each sector and make the updates faster, we keep the sum of the EELs: each time one sub-sector is updated, we just need to subtract the past value of that sector, add the new and average the value.

For a sector EEL change, we need to remove one element from a list and insert it ordered. Since both removal and insertion can be done in the same iteration of the list, the operation has $O(N)$ complexity, with N being the size of the sector.

If we can find a suitable vm, **Allocate** will perform a direct request to the hypervisor that will then allocate the VM with the defined resources.

Algorithm 3 Local scheduling: Energy-Efficiency-Driven Approximate Best-Fit

```
Require: $hosts$ available hosts                          ▷ sorted by EEL
Require: $vm$ VM to be allocated
1:  function GenericLocalScheduling($hosts, vm$)
2:      $selectedHost \leftarrow null$
3:      $host \leftarrow hosts$
4:      do
5:          if FitsCriteria($host, vm$) then
6:              $selectedHost \leftarrow host$
7:              break
8:          end if
9:          $host \leftarrow host.next()$
10:     while ($host.hasNext()$)
11:     if $selectedHost \neq null$ then
12:         UpdateHostsState($selectedHost, vm$)           ▷ asynchronous call
13:         Allocate($selectedHost, vm$)
14:         return $true$
15:     end if
16:     return $false$
17: end function
```

Algorithm 4 is the additional algorithm used when no host can fulfil the VM's requirements. This algorithm finds the host which will have the better EEL, when increased the CPU frequency, and that can then allocate the VM. **FitsIncrease** is very similar to **FitsCriteria**, but instead of comparing CPUA it checks if the host can still increase CPU frequency and if, after the increase, it will be able to host the *vm*. **IncreaseDVFS** sends a request to the hypervisor for increasing the DVFS level of a host.

If after CPU frequency increase, we still cannot allocate the *vm* in the *host*, we apply a CPU decrease in all the VMs of the most efficient host. The method **decreaseVMMipsToHostNewVm** will return a host prepared for allocation after applying the partial utility [39, 26] algorithm on the other VMs.

Algorithm 4 Local scheduling: Efficiency-Driven Increasing Best-Fit

```
Require: hosts available hosts                                  ▷ sorted by EEL
Require: vm VM to be allocated
 1: function IncreasingLocalScheduling($hosts, vm$)
 2:     if GenericLocalScheduling($hosts, vm$) = $true$ then
 3:         return $true$
 4:     end if
 5:     $selectedHost \leftarrow null$
 6:     $host \leftarrow hosts$
 7:     do
 8:         if FitsIncrease($host, vm$) then
 9:             IncreaseDVFS($selectedSector, vm$)
10:             $selectedHost \leftarrow host$
11:             break
12:         end if
13:         $host \leftarrow hosts.next()$
14:     while ($host.hasNext()$)
15:     if $selectedHost == null$ then   ▷ fallback to energy oblivious partial utility
    scheduling
16:         $selectedHost \leftarrow decreaseVMMipsToHostNewVm(vm)$
17:     end if
18:     UpdateSectorsState($selectedSector, vm$)                ▷ asynchronous call
19:     Allocate($selectedSector, vm$)
20:     return $true$
21: end function
```

Algorithm 5 is the last attempt we make to allocate a VM. When we decide to reduce resources in the VMs of a host, we start by setting the frequency of the CPU to the maximum, so that we have to decrease less resources on the VMs already running on the host. After the host is at maximum frequency, we calculate the percentage that will need to be decreased in each VM to be able to allocate the new VM.

Algorithm 5 Decrease MIPS on host

```
Require: hosts available hosts                                ▷ sorted by EEL
Require: vm VM to be allocated
 1: function decreaseVMMipsToHostNewVm(vm)
 2:     host ← hosts.first
 3:     if decreaseIsEnabled then
 4:         host.setMaxFrequency()
 5:         percentage ← host.getPercetageToDecrease(vm)
 6:         host.DecreaseAllVMs()
 7:         return host
 8:     end if
 9:     return null
10: end function
```

Our solution is divided into the three phases explained previously, each of them has a very well defined objective to fulfil. The first, intends to allocate all the first VMs, just by looking up a host, while the second starts the increase in frequency, trying to level the power, taking into consideration the requests. The last phase is intended to balance the system, and reduce the effects of the fragmentation created by the hierarchy, by decreasing VM MIPS and allocate in new requests.

While the first two phases will be happening all the time, we expect the last phase to occur only in approximately 10% of the requests.

Chapter 4

Implementation

Our solution was implemented in a state of the art cloud simulator, Cloudsim [1]. We chose this simulator because it is widely used by many authors, has several of the needed functions, is easily extensible and made distributed by *Cloud^{2}Sim* [40].

As of the time this work was done, Cloudsim did not support DVFS natively in the main code base. To be able to test our algorithm we used the research done by [35], creators of Cloudsim, which implements all the necessary features to simulate DVFS in the cloud. All the results of the simulation compare our hierarchical algorithm with two of the algorithms from Guerout et al. [35], *PowerVmAllocationPolicySimpleWattPerMipsMetric* and *PowerVmAllocationPolicyDVFSMinimumUsedHost*.

4.1 Overall implementation approach

4.1.1 Cloudsim architecture

We have chosen CloudSim for its wide usage and maturity in IaaS simulations. It allowed us to create the hierarchical architecture of the datacenter and implement our scheduling policy for the allocation of virtual machines.

Figure 4.1 depicts Cloudsim layered organisation. The first layer, User level, represents the configuration that the user of CloudSim has to perform in order to prepare the simulation. At this level, the user must specify the relation between Cloudlets (tasks on Cloudsim) and VMs.

The representation of cloudlets is defined by the number of processing elements (PEs), memory, storage requested, and number of millions of instructions (MI) they represent.

The CloudSim core is divided into four layers. The first, User Interface Structures, contains the artefacts composed by the user to interact with the simulation, namely, virtual machines and cloudlets. Next, the VM Services layer, determines

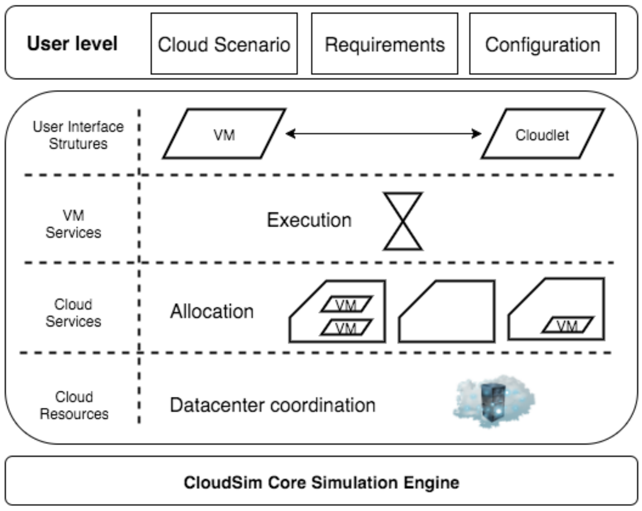


Figure 4.1: Cloudsim organization layers

how progress is made on the Cloudlets, based on the available resources. Allocation and all the resource management is done in the Cloud Services layer. In the final layer of the core, Cloud Resources, we can find the datacenters. To connect the core, an event engine, which keeps track of simulation time and is used for the communication between simulation entities, such as the datacenter and the broker between the clients and the datacenter. These are the main entities that receive and create events and delegate the work to other entities such as the Hosts.

Scheduling decisions are done at two main points: a) when selecting the hosts to allocate VMs, b) when determining the cores assigned to each VM. In both points, there are default policies that can be customised by the user, in order to achieve different objectives.

Our algorithm is only concerned with the VM-Host level of the allocation. For that reason, we customised the scheduling decision at the level of the allocation policy in Cloudsim.

4.2 Cloudsim extensions

Our work extends Cloudsim in two aspects: architecture of the datacenter and allocation policy.

4.2.1 Hierarchy

Figure 4.2 highlights the main component changes: *DatacenterBroker*, *VM*, *Host*, *AllocationPolicy*, and *Governor*.

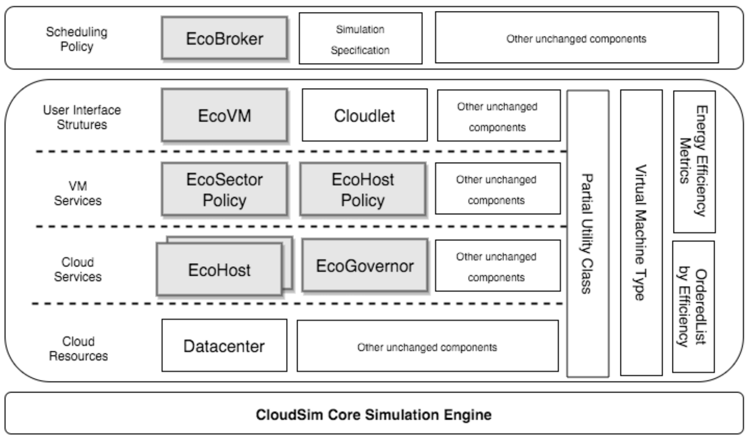


Figure 4.2: Highlighted extensions to the CloudSim simulation environment

Datacenter broker

To capture revenue metrics, we created the *EcoBroker* which extends the *Power-DatacenterBroker*. By extending the broker we were able to register the allocations being performed, calculate infrastructure expenses, the revenue, and were to find the profits of the system in the simulation.

To model revenue, we defined several categories of virtual machines: *micro*, *small*, *regular*, and *extra*, from the less to the more powerful. With each category, we associated an utility class, representing the allowance of the client for reductions on his VMs.

The result was a price matrix used to estimate revenue and the infrastructure cost of the datacenter as seen in [26].

Host and VM

In order to collect revenue data, VMs were extended with classification regarding their type and utility. Each VM was assigned a type regarding its size and an utility, to account for how much the client is willing to reduce the initial requirements, in return for a discount in the final price.

Hosts were extended, to collect information about types of failures in the allocations, mostly so we could focus our concerns in the CPU. When a VM failed to allocate we registered what was the cause and this allowed us to see when the datacenter was getting full, and what was the average number of VMs to Host ratio in the allocations.

Allocation policy

Cloudsim's architecture is flat, meaning that each datacenter contains a collection of hosts that are indirectly managed through the allocation policy. Our algorithm targets a hierarchical arrangement of the hosts, partitioned in smaller sectors.

Hierarchisation of the hosts was achieved by abstracting the current concept of allocation policy and creating two types of policies: sector policies and host policies. Sector policies abstract the allocation on the sectors while host policies, similar to the existing flat policies, allocate VMs directly in the hosts.

In Figure 4.3, we present a summary of the algorithm class hierarchy and the main methods implemented. When performing an allocation, each sector finds the best sub-sector, based on the energy efficiency, and delegates the allocation until it reaches the host. As soon as the VM is allocated, the update of the metrics is triggered and all the chain re-calculates the values to match the changes done in the allocation.

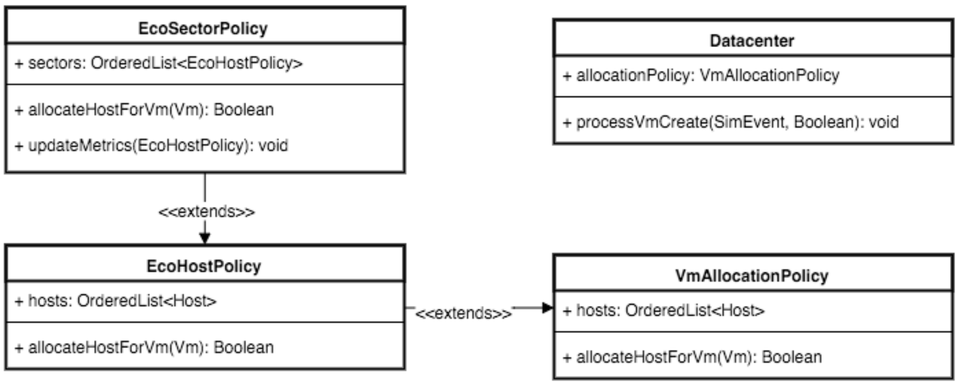

EcoSectorPolicy
+ sectors: OrderedList<EcoHostPolicy>
+ allocateHostForVm(Vm): Boolean
+ updateMetrics(EcoHostPolicy): void
<<extends>>
EcoHostPolicy
+ hosts: OrderedList<Host>
+ allocateHostForVm(Vm): Boolean
<<extends>>
Datacenter
+ allocationPolicy: VmAllocationPolicy
+ processVmCreate(SimEvent, Boolean): void
VmAllocationPolicy
+ hosts: OrderedList<Host>
+ allocateHostForVm(Vm): Boolean

Figure 4.3: Allocation policy hierarchical extension to CloudSim

4.2.2 Algorithm routines

The two allocation policies used to compare our algorithm, found in Cloudsim, PowerVmAllocationPolicySimpleWattPerMipsMetric and PowerVmAllocationPolicyDVFSMinimumUsedHost, use an extensive search for the perfect Host based on a specific metric. In the first, case it uses a Watt per MIPS metric while on the second it looks for the host with lower CPU load. This type of search does not scale, since it needs to check the whole list every time it needs to perform an allocation.

Our policy introduces a partial search algorithm with pre-calculated metrics to achieve faster decisions. The solution is composed of two parts: data structure, the ordered list of sectors (or hosts), and the decision routine which finds the first suitable match in the list for the tasks.

The data structure described in Section 3.3, is implemented by class *Ordered-*

List, which extends the Java implementation of *LinkedList* with two methods named orderedAdd, one for elements other for collections of elements. In this particular implementation, we were not concerned with insert performance details, since Cloudsim only measures virtual time for the tasks and does not have any metrics on the allocation itself. Additionally, in a real deployment, this is to be carried out asynchronously.

The algorithms were implemented in two classes: *EcoSectorPowerVmAllocationPolicy* implements Algorithm 1 while *EcoHostPowerVmAllocationPolicy* implements Algorithm 4.

Chapter 5

Evaluation

This chapter describes the detailed evaluation of our energy-utility-driven algorithms, while comparing it with some of state of the art algorithms for placing virtual machines (or servers) in a datacenter aiming for energy efficiency.

Section 5.1 describes the configurations of the different datacenters, hosts, and VMs used in our simulation. Section 5.2 analyses the allocations success. Section 5.3 shows the effects of the revenue when using overcommit of resources. Section 5.4 analyses the energy used along the process. Section 5.5 studies the influence of the algorithms in the execution of the workloads.

The evaluated metrics, relevant to the provider, are the VMs requested but not allocated, resource utilisation, and revenue, while for the owner, they are the total execution time of workloads and the price. After the implementation and validation of our DVFS algorithm model in CloudSim, this section presents all the relevant metrics comparing our agains the two other implementations.

All the tests use PlanetLab workloads[1], to create a close to real experience and obtain more solid results.

The power models used in the simulations experiments are based on real experimental values from REIMS GRID 5000 SITE[2] on CPU Intel(R) Core(TM)2 Quad CPU Q6700 @ 2.66GHz and 4GB RAM running with Ubuntu TLS 10.4 (Linux kernel 2.6.32). All energy consumption measures have been done on this host with Plogg wireless electricity meter that allows live power consumption. The Values were measured by changing frequency of 1 core in a 4 core CPU [35].

Our test simulation was executed on a CPU Intel(R) Core(TM) i7 CPU 870 @ 2.93GHz and 12GB RAM running Ubuntu LTS 14.04.2 (Linux kernel 3.13.0-55).

[1]PlanetLab workloads: https://github.com/beloglazov/planetlab-workload-traces; https://www.planet-lab.org/

[2]REIMS GRID 5000 SITE: https://www.grid5000.fr/mediawiki/index.php/Status

Table 5.1: Datacenter sizes

DC-Size	**Hosts**	**Depth**	**Fan-out**
Size-1	1000	2	2
Size-2	5000	2	2
		2	3
Size-3	10000	2	4
		2	3
		4	2

Table 5.2: VM Types

VM Type	**MIPS**	**CPUs**	**RAM (MB)**
Micro	500	1	870
Small	1000	1	1740
Regular	2000	1	1740
Extra	2500	1	2613

5.1 Methodology and Configurations

To evaluate our energy-efficient proposal we used a configuration composed of different datacenter arrangements, two types of host machines and four different VM types that were available to the client from request.

First we will describe the datacenters, which are characterised by: the number of hosts available in the datacenter; the depth of the hierarchy: number of intermediate sectors levels; hierarchy fan-out: number of children of each node in the hierarchy. All the tested combinations are listed in Table 5.1, and go from a small datacenter of 1000 hosts to higher sizes with 10000 hosts.

Due to the increasing memory requirements for the tests we stopped at 10000 hosts. Our algorithm targets datacenters with sectors of size around 1000 hosts or more, and with the tests done we can extrapolate the scalability of the algorithm.

In order to simulate heterogenous client requests, we defined four types of VMs, as listed in Table 5.2. With a limited number of types, covering tasks from small to large requirements, we were able to obtain more precise results regarding the distribution of the MIPS.

To create a more diverse allocation scenario, we chose two different types of host, catalogued in Table 5.3. With the first type, we offered less CPU power with higher

Table 5.3: Host sizes

Host-Size	**MHz**	**Memory (MB)**	**Storage (TB)**
Size-1	3720	10000	1
Size-2	5320	8192	1

Table 5.4: CPU Frequencies

CPU-Freqs	**Percentage (%)**
Freq-1	100.00
Freq-2	89.89
Freq-3	79.89
Freq-4	69.93
Freq-5	59.925

memory, while in the second case we had more CPU with a little bit less memory than the previous. This setup was projected to handle easily two VMs in the *Size-2* host and not always two VMs in *Size-1* host.

The energy simulation, described previously, offered five levels of CPU consumption, present in Table 5.4 the frequencies range from 59% to full CPU usage. Using Cloudsim the values take into consideration whether the CPU is being used or idle, considering then ten levels of power usage (five idle and five when being used).

Simulating the requests was done using heterogeneous virtual hardware, from the options listed in Table 5.2, since it is becoming a common practice in the literature [34].

From now on, for simplicity purposes, when we refer to the algorithms, we will use shorter versions of their names: *PowerVmAllocationPolicySimpleWattPerMipsMetric* will be *WattPerMip*, *PowerVmAllocationPolicyDVFSMinimumUsedHost* will be *MinUsed*, and our *EcoPowerVmAllocationPolicy* will be *EcoWattPerMip*. In the cases where we simulated different configurations of the datacenter, our algorithm will be suffixed with the depth and fan-out of the hierarchy: *EcoWattPerMip*, for depth two and fan-out; *EcoWattPerMip32*, for depth three and two of fan-out.

For the example *EcoWattPerMip32* we will have $3^2 = 9$ sectors as leaves of the hierarchy.

5.2 Allocation success rate

The provider-side metrics measured considering the allocation success, i.e., number of failed VMs and MIPS, show that our strategy was able to, at least, match the non-hierarchical strategies described in Guerout et al. [35].

Figure 5.1 and Figure 5.2 show that all the algorithms start rejecting VMs at about 86% of the capacity. Since our algorithm uses a hierarchical distribution, and partial data to make decisions, the failure rates should be higher than the other optimal algorithms. The reason why it can handle the allocations better, is related to how the search for the host is done: the non-hierarchical alternatives optimise the search for their objectives, not taking into account the capacity of the machine, and in case the perfect machine they selected cannot handle the VM they will immediately reject it. In our case, we search the ordered list, that already considers our objective in its ordering, but then we choose a host that is capable of handling our VM (if available).

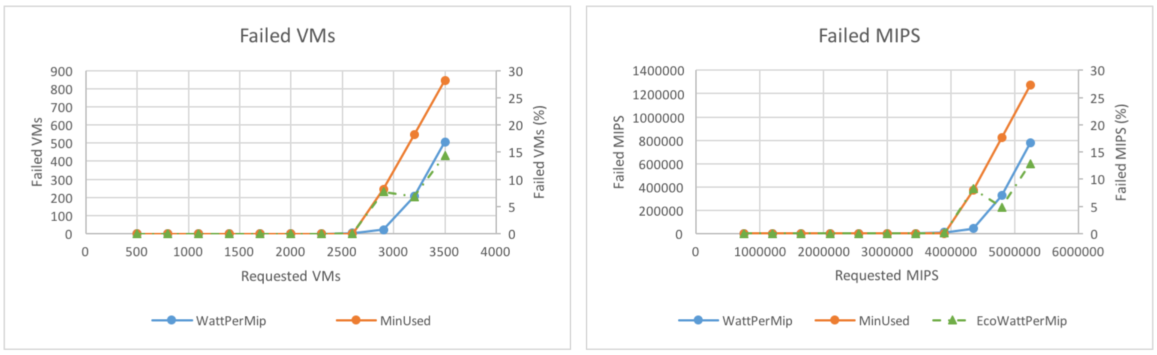


Figure 5.1: Size 1 - Failed allocations

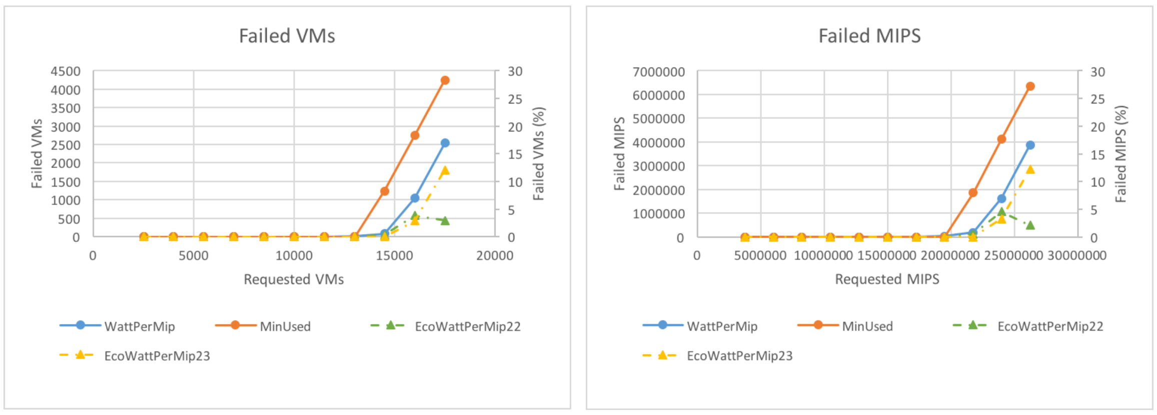


Figure 5.2: Size 2 - Failed allocations

In the largest test case, depicted in Figure 5.3, we can see that our algorithm still behaves well, but already looses the best allocation ratio to the *WattPerMip* policy. This is still a good result, since we increased the size of the datacenter ten times, and we still can compete with the flat algorithms regarding failed allocations. Flat algorithms do not loose any performance in Cloudsim, since it does not measure the allocation calculations or any other values not concerning the tasks runtime, but our algorithm requires significantly fewer calculations due to its hierarchical approach and list ordering.

This kind of algorithms are doing extensive searches of all the hosts to find their match, and this is a serious scalability problem if we are talking about a large size datacenter (e.g. thousands of hosts).

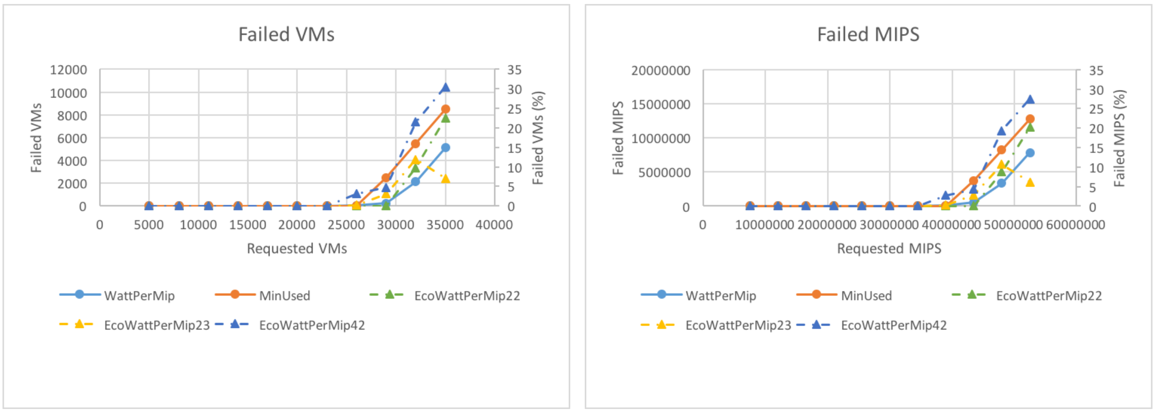


Figure 5.3: Size 3 - Failed allocations

5.3 Overall revenue

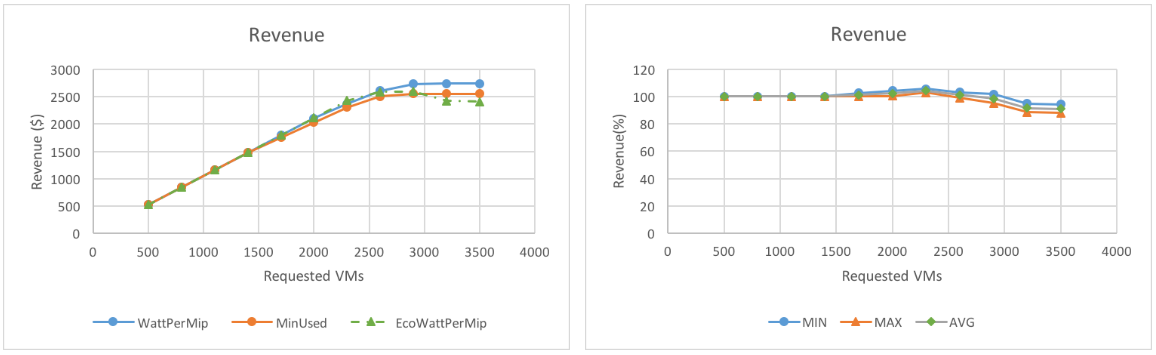


Figure 5.4: Size 1 - Revenue (per hour)

Revenue is a very important metric, especially to understand the efficiency in terms of resources of the algorithms. On average, our algorithm had a smaller revenue than the others, even when running more VMs, as depicted in Figure 5.4. This is related to the fact that we reduced VMs power in order to have better allocations. But, in the end, if we also take into consideration the satisfaction of the client (paying less) and the the price of the energy, our algorithm could be more efficient and profitable.

5.4 Energy efficiency

One of our main goals, was to design an energy-efficient algorithm. This objective was achieved as it can be observed in Figure 5.5. On average, our algorithm consumed less power than the other two. This affirmation is only false, when our algorithm executed a higher number of VMs than the others, and only in those cases it used a slightly higher amount of power.

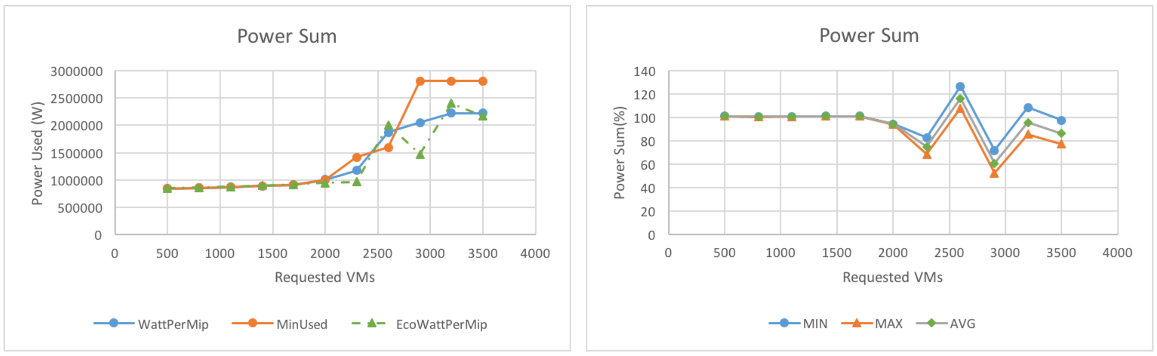
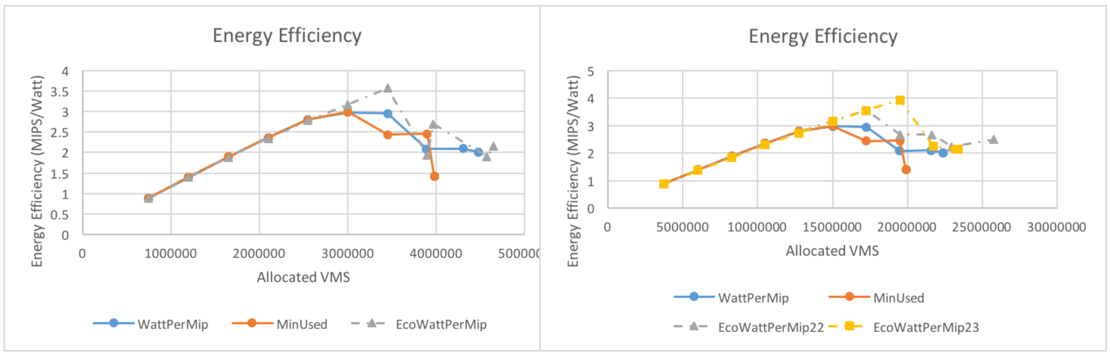


Figure 5.5: Size 1 - Power Sum



While aiming for energy efficiency, we intended to execute more MIPS using the same Watts. To measure this, we used Equation (5.1), the relation between the real allocated MIPS in each algorithm, divided by the sum of the used power.

$$EnergyEfficiency = \frac{RequestedMips - FailedMIPS}{PowerSum} \tag{5.1}$$

Depicted in Figure 5.6, we have the full results for all the algorithms in each *Size*: higher values mean more efficient algorithms.

For all sizes, our efficiency is considerably higher than the other two, except for *Size-3* with depth 4, where our values are closer, but this is due to the fact that the times of the other algorithms are considering the amount of time needed to search ten thousand hosts in each VM request.

5.5 Effects on workloads

Regarding user-related metrics we analysed the effects of our algorithm on task execution time. As stated before, the simulation relied on tasks generated by VMs provisioned at PlanetLab [29]. Each of the generated workloads was assigned to a VM in our simulation, to be used as work being required by the VM.

As we can see in Figure 5.7, our policy has an average execution time that

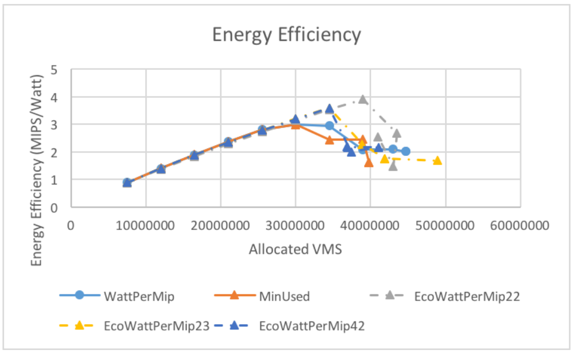
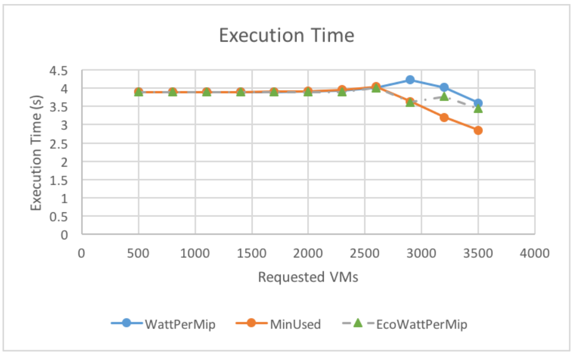


Figure 5.6: Energy Efficiency



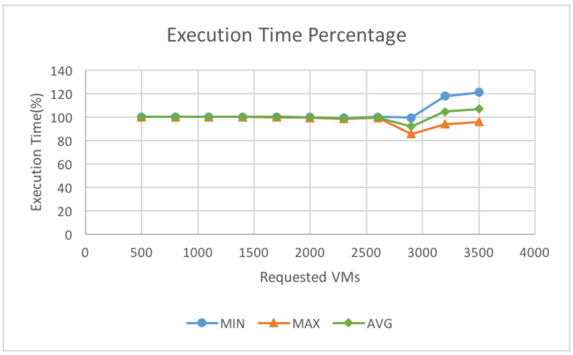


Figure 5.7: Size 1 - Execution times

matches the execution time of the other policies. Still, if we cross-reference this data with the failures in allocation, we will see that the lower execution times of policy *MinUsed* are directly related to that fact it executed less VMs. In Figure 5.7, we depict the percentage of the differences between our algorithm and the minimum, maximum, and average of the other algorithms.

5.6 Overall results

In sum, we can consider our solution a very good alternative to the current flat policies, especially the ones that rely on full data search to make decisions.

Our tests proved that we can delegate work in a hierarchy, and make decisions based on partial data and still be efficient.

Using ordered lists and pre-processing the information about the hosts helps the allocations spare energy and reduce the amount of effort handling an allocation.

After the tests, we can conclude that our solution behaves better in large groups of hosts (e.g. at least a thousand hosts), and there is a noticeable decrease of the efficiency when each sector is less that 10% of the whole system.

Chapter 6

Conclusion

Cloud computing is being embraced as an important part of the future in the technological future. With the growing number of cloud service users, technology giants such as Amazon, Google and Microsoft are exploring this business, and securing new clients. Also in academia, many of the research groups are working in this area or one related, such as grids or clusters.

In this thesis, we proposed a solution that extends some of the models being worked by the academic researchers and try to help in solving important problems such as energy efficiency, while addressing the critical issue of scaling the scheduling of the computational power in the datacenters.

6.1 Concluding remarks

We started this document by presenting the cloud and the importance of virtualisation in the global view about scheduling. We described and classified some cloud solutions, the major classes of algorithms and some classic scheduling approaches. The analysis of all these topics allowed us to have the necessary knowledge to identify some aspects that have not been explored.

Once the shortcomings were identified, we proposed a solution that considers the datacenter as a structured hierarchical network divided into sectors, with local schedulers that interact with the upper levels, by exchanging information about the state of their machines. The solution was implemented in Cloudsim and tested against multiple heterogenous situations.

The obtained results show that our solution efficiently assigns resources to jobs, according to their requirements and helps to maintain an energy-efficient infrastructure.

Our algorithm demonstrated efficiency for setups with at least one thousand hosts per sector, when this value decreases, we start failing more VMs and the fragmentation creates a less efficient environment.

6.2 Future work

There is still work to do, to achieve more efficient infrastructures. Companies such as Amazon, Google and Microsoft are constantly improving their processes, not only to cut costs in maintaining their datacenters, but also to reduce the environmental effects of this large energy wasted in the deployment of machines.

Regarding some specific topics of our work, there are some aspects that should be considered as future work. Some research can be done around the specific decisions of our work, but also related to the energy efficiency topic in general. Next we enumerate and describe some examples:

- **VM Migration** One of the main problems of our solution is the fragmentation of the VMs between the sectors, the more sectors we have, for the same amount of VMs, the less precision we have when making a decision about where to allocate the VM. To reduce this problem, we could use VM migration to balance our sectors and minimise fragmentation in the whole system.

- **Integration with other simulators** Cloudsim is currently one of the state of the art simulators. Still, it only considers that tasks progress based on the CPU that is assigned to them. One of the problems we intend to solve is scalability and that cannot be measured directly with Cloudsim. It would be interesting to simulate the cost of processing allocation decisions and be able to measure the time and number of instructions one algorithm uses in order to process an allocation.

- **Energy price and source** Our algorithm is focused on the efficiency in the hosts, meaning that it would try to keep energy used to perform one task lower, without compromising other metrics such as execution time. Sometimes, if we use more power, we can execute tasks faster. Even better would be to take into consideration the energy price at the moment, to increase the frequencies and do more work when the energy is cheaper. Not only is price important, the source of the energy is also an important factor, regarding the environment, to make decisions and that could be applied in the same way as the price.

Bibliography

[1] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[2] L. Wang, G. Von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, and C. Fu, "Cloud computing: a perspective study," *New Generation Computing*, vol. 28, no. 2, pp. 137–146, 2010.

[3] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, ISPAN '09, (Washington, DC, USA), pp. 4–16, IEEE Computer Society, 2009.

[4] G. von Laszewski, L. Wang, A. J. Younge, and X. He, "Power-Aware Scheduling of Virtual Machines in DVFS-enabled Clusters," in *Proceedings of the 2009 IEEE International Conference on Cluster Computing (Cluster 2009)*, (New Orleans), IEEE, 31 Aug. – Sep. 4 2009.

[5] R. Meyer and L. Seawright, "A virtual machine time-sharing system," *IBM Systems Journal*, vol. 9, no. 3, pp. 199–218, 1970.

[6] R. C. Daley and J. B. Dennis, "Virtual memory, processes, and sharing in multics," *Commun. ACM*, vol. 11, pp. 306–312, May 1968.

[7] A. S. Lett and W. L. Konigsford, "Tss/360: A time-shared operating system," in *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), (New York, NY, USA), pp. 15–28, ACM, 1968.

[8] J. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[9] H. Katzan, Jr., “Operating systems architecture,” in *Proceedings of the May 5-7, 1970, Spring Joint Computer Conference*, AFIPS ’70 (Spring), (New York, NY, USA), pp. 109–118, ACM, 1970.

[10] C. A. Waldspurger, “Memory resource management in vmware esx server,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 181–194, 2002.

[11] VMware, “Understanding full virtualization, paravirtualization and hardware assist,” 2007.

[12] J. M. Schopf, “Grid resource management,” ch. Ten Actions when Grid Scheduling: The User As a Grid Scheduler, pp. 15–23, Norwell, MA, USA: Kluwer Academic Publishers, 2004.

[13] T. L. Casavant and J. G. Kuhl, “A taxonomy of scheduling in general-purpose distributed computing systems,” *Software Engineering, IEEE Transactions on*, vol. 14, no. 2, pp. 141–154, 1988.

[14] J. L. V. Vasques, “A decentralized utility-based scheduling algorithm for grids,”

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[16] J. N. Silva, L. Veiga, and P. Ferreira, “A2ha—automatic and adaptive host allocation in utility computing for bag-of-tasks,” *Journal of Internet Services and Applications*, vol. 2, no. 2, pp. 171–185, 2011.

[17] S. Abrishami, M. Naghibzadeh, and D. H. Epema, “Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds,” *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[18] S. Abrishami, M. Naghibzadeh, and D. H. Epema, “Cost-driven scheduling of grid workflows using partial critical paths,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1400–1414, 2012.

[19] M. Rodriguez Sossa and R. Buyya, “Deadline based resource provisioning and scheduling algorithmfor scientific workflows on clouds,” 2014.

[20] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini, “Deepdive: transparently identifying and managing performance interference in virtualized environments,” in *Proceedings of the 2013 USENIX conference on Annual Technical Conference*, pp. 219–230, USENIX Association, 2013.

[21] N. Rameshan, L. Navarro, E. Monte, and V. Vlassov, "Stay-away, protecting sensitive applications from performance interference," in *Proceedings of the 15th International Middleware Conference*, pp. 301–312, ACM, 2014.

[22] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[23] V. Ishakian, R. Sweha, A. Bestavros, and J. Appavoo, "Cloudpack," in *Middleware 2012*, pp. 374–393, Springer, 2012.

[24] M. Macias and J. Guitart, "A risk-based model for service level agreement differentiation in cloud market providers," in *Distributed Applications and Interoperable Systems*, pp. 1–15, Springer, 2014.

[25] H. Morshedlou and M. Meybodi, "Decreasing impact of sla violations: A proactive resource allocation approach for cloud computing environments," *IEEE Transactions on Cloud Computing*, p. 1, 2014.

[26] J. Simão and L. Veiga, "Partial utility-driven scheduling for flexible sla and pricing arbitration in clouds," *IEEE Transactions on Cloud Computing*, 2013.

[27] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys (CSUR)*, vol. 37, no. 3, pp. 195–237, 2005.

[28] A. J. Younge, G. Von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers, "Efficient resource management for cloud computing environments," in *Green Computing Conference, 2010 International*, pp. 357–364, IEEE, 2010.

[29] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[30] S.-H. Wang, P.-W. Huang, C.-P. Wen, and L.-C. Wang, "Eqvmp: Energy-efficient and qos-aware virtual machine placement for software defined datacenter networks," in *Information Networking (ICOIN), 2014 International Conference on*, pp. 220–225, IEEE, Feb 2014.

[31] Y. Mhedheb, F. Jrad, J. Tao, J. Zhao, J. Kołodziej, and A. Streit, "Load and thermal-aware vm scheduling on the cloud," in *Algorithms and Architectures for Parallel Processing*, pp. 101–114, Springer, 2013.

[32] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster computing*, vol. 12, no. 1, pp. 1–15, 2009.

[33] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters," *Future Generation Computer Systems*, vol. 37, pp. 141–147, 2014.

[34] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, (Washington, DC, USA), pp. 826–831, IEEE Computer Society, 2010.

[35] T. Guérout, T. Monteil, G. Da Costa, R. N. Calheiros, R. Buyya, and M. Alexandru, "Energy-aware simulation with dvfs," *Simulation Modelling Practice and Theory*, vol. 39, pp. 76–91, 2013.

[36] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges," in *PDPTA 2010: Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 6–17, CSREA Press, 2010.

[37] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[38] L. Sharifi, N. Rameshan, F. Freitag, and L. Veiga, "Energy efficiency dilemma: P2p-cloud vs. datacenter," *IEEE Transactions on Cloud Computing*, 2014.

[39] J. Simão and L. Veiga, "Flexible slas in the cloud with a partial utility-driven scheduling architecture," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1, pp. 274–281, IEEE, 2013.

[40] P. Kathiravelu and L. Veiga, "An elastic middleware platform for concurrent and distributed cloud and map-reduce simulation-as-a-service," *IEEE Transactions on Cloud Computing*, 2014.

Appendix A

Algorithm characteristics

	Local vs Global	**Static vs Dynamic**	**Centralised vs Distributed**	**Adaptive vs Non-Adaptive**	**Load Balancing**	**Immediate vs Batch**	**Approximate vs Heuristic**
A2HA	Global	Dynamic	Centralised	Adaptive	No	Immediate	Heuristic
IC-PCP	Global	Dynamic	Centralised	Non-Adaptive	No	Immediate	-
IC-PCPD2	Global	Dynamic	Centralised	Non-Adaptive	No	Immediate	-
PSO	Global	Dynamic	Centralised	Non-Adaptive	No	Immediate	Approximate
Stay-Away	Global	Dynamic	Centralised	Adaptive	No	Immediate	-
Beloglazov 2010	Global	Dynamic	Distributed	Non-Adaptive	Yes	Immediate	-
Beloglazov 2012	Global	Dynamic	Distributed	Non-Adaptive	Yes	Immediate	-
Von	Global	Dynamic	Centralised	Non-Adaptive	No	Batch	-
EQVMP	Global	Dynamic	Centralised	Non-Adaptive	Yes	Immediate	-
ThaS	Global	Dynamic	Centralised	Non-Adaptive	Yes	Immediate	-
Cloudpack	Global	Dynamic	Centralised	Non-Adaptive	No	Immediate	-
Macias	Global	Dynamic	Centralised	Non-Adaptive	No	Immediate	-
Morshedlou	Global	Dynamic	Centralised	Non-Adaptive	No	Immediate	-
Partial Utility	Global	Dynamic	Centralised	Non-Adaptive	No	Immediate	-

Table A.1: Algorithms characteristics