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

Rodrigo Tavares Fernandes
rodrigo.fernandes@tecnico.ulisboa.pt

Instituto Superior Técnico
Avenida Rovisco Pais 1 1049-001 Lisboa
INESC-ID

Abstract. Cloud computing gained tremendous importance in the past decade, emerging as a new computing paradigm and aiming to provide reliable, scalable and customizable 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 optimize an objective function. Utility is a measure of the client satisfaction that can be seen as a objective function maximized by schedulers based on the agreed service level agreement (SLA). Our *EcoScheduler* aims at saving energy by using 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 an hierarchical architecture we intend to create a scalable solution for the fast growing heterogeneous cloud.

Keywords: cloud, utility scheduling, dvfs, energy efficiency, partial utility

1 Introduction

Cloud computing is the paradigm that changed the way we see and use information technology services. This services aim to provide reliable, scalable and customizable dynamic computing environments for end-users. With the dynamical provision of resources, the cloud can give better management to resources by optimizing their usage and providing a pay-as-you-go pricing model.

The cloud is built over datacenters spread all over the world, that usually contain large groups of servers connected to the Internet. This infrastructure has to be maintained by the providers that take care of all the working structure and have to keep in mind the environmental footprint it will leave, specially because such structures have huge footprints of energy waste and CO2 emissions. To achieve better results the providers rely on efficient scheduling algorithms to manage the datacenters and take the best out of the resources.

A scheduling algorithm tries to find a resource for a job by fulfilling it's requirements and at the same time optimize an objective function that takes into consideration the user satisfaction and the providers profits. Utility is a measure of a user's satisfaction that can be seen as an objective function that a scheduler tries to maximize based on the SLA.

The performance issues of the scheduling algorithm include not only execution times but also resource utilization. A better scheduler can use fewer resources and run jobs faster. Use fewer resources is very important, specially because it helps consume

less energy, and energy consumption is one of the major issues for building large-scale clouds.

In this paper, we propose *EcoScheduler*, a scheduling algorithm for allocating jobs in the cloud with resource-awareness, user satisfaction and 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 efficiently assigns proper resources to jobs according to their requirements.

The rest of the document is organized as follows: in Section 2 we present what we pretend to achieve, the study and analysis of the related is work is done in Section 3, in Section 4 we present our solution to address the shortcomings mentioned before, the evaluation methodology and metrics are described in Section 5, in Section 6 we present some concluding remarks. There is also an appendix section. In Appendix A we present a table recapitulating the algorithms described in Section 3.2.3 and in Appendix B we present the planning that we are going to follow during the implementation of our solution.

2 Objectives

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. The notions of partial utility are to be enriched by incorporating DVFS (dynamic voltage frequency scaling of CPUs) for improved energy efficiency, and auctioning (that can be simulated) allowing several clients and providers to negotiate for VM allocation taking into account performance, cost and energy profile.

3 Related Work

This section describes the most relevant research work for the definition of *EcoScheduler*, our eco-friendly scheduling algorithm, organized according to a top-down approach. In Section 3.1 we present a background analysis of cloud computing with an overview of the concepts behind it, such as virtualization, virtual machines and hypervisors. Next, on Section 3.2 we describe various aspects about scheduling, such as types of virtual machine scheduling algorithms and how they differentiate from each other. Finally, in Section 3.3 we describe energy and environmental aware algorithms. Then we conclude with some analysis and discussion.

3.1 Cloud Computing and Virtualization

Today most IT companies face challenges related to fast changing environments with very specific requirements. This conditions happen for both modern and legacy applications which need reliability, security and some times strict assured quality of service (QoS). To mitigate this 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 virtualization, which is a technique that splits physical infrastructures of resources in isolated computing parts.

Figure 1 describes some very important events in the history of virtualization that lead to its massive usages and the expansion of the cloud.

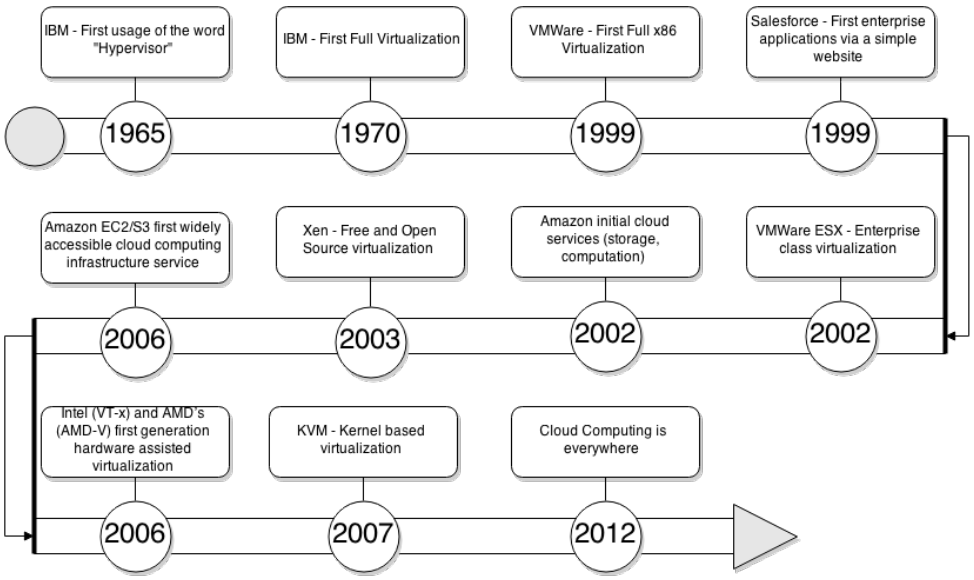

Fig. 1. History of Virtualization

3.1.1 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 [34], occurred due to its reliability, scalability and customization that created a new market where all the big companies fight to provide the best services. Examples of services can go from a complete virtual machines to simple email clients or file hosting services.

The cloud computing services stack can be classified and organized into three major offerings [30]: (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 with just a couple of clicks.

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.

Software as a service is the most common and provides some piece of software hosted in the provider's machine that is accessible in 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 infrastructure and is accessible everywhere with minimal setup.

Advantages

Cloud computing distinguishes it self from other services for various advantages [34,14].

- **On-demand provisioning and elasticity:** Computing clouds provide on-demand resources and services. The client can create and customization the services for his needs, by configuring all the parameters 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, specially 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

With flexibility and simplicity come some disadvantages.

- **Overhead:** Virtualization evolved a lot during the past years, specially in terms of performance when compared to native physical execution. The developments in software and hardware assisted virtualization made them very close, although in critical high performance applications virtualization still has some overhead.
- **Resource interference:** One of the most common issues with virtualization is to have many virtual servers within the same physical machine, because 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) [1] is a web service that provides re-sizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. It provides a very simple web interface that allows the

[1] Amazon Elastic Compute Cloud (EC2): `https://aws.amazon.com/ec2/`

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 tools to build failure resilient applications and to isolate themselves from common failure scenarios without minimal maintenance for the client.

Microsoft Azure [2], 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 big enterprise clients, offering easier integration with in-premises installations and more technical support.

Google Cloud Platform Compute [3] is yet another on-line platform for large-scale workloads on virtual machines and is hosted on Google infrastructure. It is the response from Google, that already provided SaaS and PaaS, to the other IaaS big players in the market, that became available in the late 2013 for the general public. It offers the same kind of infrastructure services and tries to challenge the other solutions with fine grained prices and better security measures.

OpenStack [4] 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 data center, can be managed in a web-based dashboard, command-line tools or a RESTful API. It works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructures. Hundreds of big brands around the world rely on OpenStack to manage their businesses every day, reducing costs and helping them move faster. One of its strong points is the big ecosystem which helps it grow and keep running with other big players.

3.1.2 Virtualization

The concept of virtualization had its origins in the late 1960s, when IBM was investigating a way of developing robust machine-sharing solutions [18] and developed M44/44X, the first close enough to a virtual machine system proving that virtualization is not necessarily less efficient than any other approach.

MULTICS [9] 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 [16] released by IBM in 1967, was one of the first operating system implementations of virtualization. The system could share a common physical memory space, it would only run a single kernel, one process launched by one processor could cause an interruption in other, 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.

Virtualization was the solution for organizations or individuals to optimize their infrastructure resource utilization, and at the same time simplify data center management.

[2] Microsoft Azure: `https://azure.microsoft.com/en-us/`
[3] Google Cloud Platform - Compute Engine: `https://cloud.google.com/compute/`
[4] OpenStack Software: `https://www.openstack.org/`

Today, every cloud company uses virtualization in their data centers 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.

The years passed but virtualization has the same meaning it had fifty years ago, allow users to have an environment where they can run multiple independent systems at the same time sharing the physical hardware.

Virtual Machine Concept

A virtual machine (VM) is a specialized software implementation that, like a physical computer, can run an operating system or execute programs as if they were running on their native system [28]. 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 purposes of having multiple OSs running in the same machine to have easier setup, minimize 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 VMs enable programs in the same language to be executed in the same way on different systems. Very common examples are Java Virtual Machine or the Common Language Runtime for .NET.

Challenges in Virtualization

As per Popek and Goldberg, there are three required properties for a virtualizable architecture [28].

- **Efficiency Property:** Provide the ability to execute innocuous instructions directly on hardware bypassing VMM.
- **Resource Control Property:** The Virtual machine monitors should be able to completely control the system. When the guest operating systems try to access resources, the access should be routed through the virtual machine monitor to secure any inconsistencies.
- **Equivalence Property:** Any program running on top of the virtual machine monitor should perform in a way it is indistinguishable from the case when the virtual machine monitor does not exist.

Hypervisor – core of system VMs

The hypervisor, also called Virtual Machine Monitor (VMM), is the host machine and the software that creates and controls the virtualization, 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 [12] 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 hypervisor which is executed in the physical machine (b) Type 2, hosted hypervisor that is executed from within the host OS as an application on an unmodified operating system (OS). Type 1 is the original model developed by IBM in the 60's and some implementation examples are Xen, KVM, Oracle VM, Microsoft Hyper-V or VMware ESX, while for Type 2 we have solutions like VMware Workstation and VirtualBox. The hypervisor plays a very important role in cloud environments since it is the key piece to manage the whole service infrastructure.

Optimizations for performance and efficiency

To perform better virtualization several optimizations were developed in the hypervisors. For handling sensitive and privileged instructions to virtualize the CPU, there are currently three alternative techniques [32].

Full virtualization or virtualization using binary translation and direct execution is the approach used by VMWare. This approach, doesn't 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 optimized 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 virtualization the guest OS is not aware that is being virtualized. Full virtualization offers the best security and isolation for virtual machines, and simplifies migration and portability as the same guest OS instance can run virtualized or on native hardware.

Paravirtualization or operating system assisted virtualization refers to virtualization 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 used by many big companies.

As opposite to full virtualization the non-virtualizable instructions are replaced with with hypercalls that communicate directly with the virtualization 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 virtualization is hard, modifying the guest OS to enable paravirtualization is the easier solution.

Hardware assisted virtualization was created to solve the problems in previous solutions, hardware vendors are rapidly embracing virtualization and starting to develop new features to simplify virtualization techniques. Both Intel and AMD started working in 2006, on generations of CPUs, 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 ring 0. This allows privileged and sensitive calls to trap automatically to the hypervisor, removing the need for either binary translation or paravirtualization.

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-bits 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 [33] 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(n2) 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.

3.2 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 he defines all the process behind the service. This is described in an algorithm called the scheduling algorithm.

3.2.1 Scheduling Algorithms

The scheduling algorithm is a program expressed as set of well defined rules for determining the most adequate choices on 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, utilization 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 big variety of factors, such as CPU usage, available memory, energy consumption, etc. Various issues arise from scheduling multiple heterogeneous systems, the predictability is usually very low and the algorithm has a hard job managing allocations.

The scheduling algorithms are characterized for 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 optimize it. Scheduling in cloud computing environment can be performed at various levels such as workflow, VM level or task level.

Figure 2 describes the intervenients in the scheduling process and the flow of the requests.

Scheduling Phases

The scheduling process can be divided in three major phases [24], resource discovery, system selection and allocation.

Resource Discovery is the first phase and consists in 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.

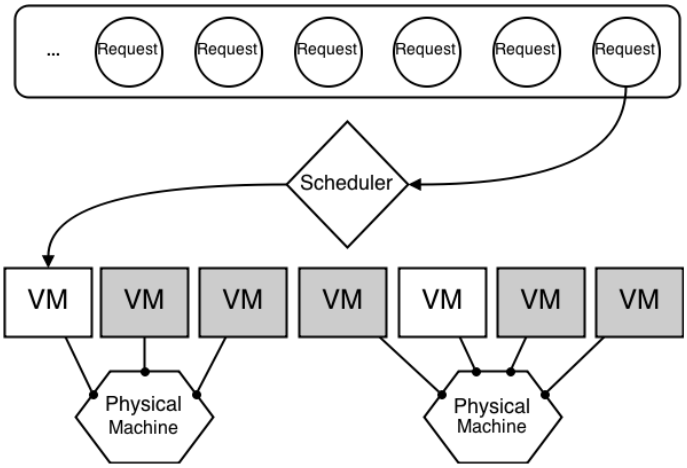

Fig. 2. Virtual machine scheduling overview

The next step is the application requirement definition which consists in 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 to select 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, specially in heterogeneous and rapidly changing environments. The final selection consists in choosing a resource with the gathered information that best fits the clients needs and creates more profit.

VM Allocation is the last phase and consists in submitting the VM to allocation. To assure service quality, some precautions must be taken, such as prepare 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 re-schedule it. The next step is to notify the user and cleanup any temporary files used during scheduling.

3.2.2 Algorithm Classes

The algorithms can be classified by several major parameters, in Casavant et al. [8] 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. Global scheduling allocates VMs to multiple physical machines being able to optimize a system-wide performance goal. Considering what was said before it is obvious to conclude that cloud scheduling for IaaS is mostly global.

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

The decentralized solution consists having no central master, by creating a communication network between the lower level schedulers to make decisions. A very common approach [29] is to use a Peer-to-peer network that communicates using the famous Chord algorithm.

Hierarchical is very similar to centralized solutions but with several lower levels of delegation. In the hierarchical approach schedulers are organized in an hierarchical way, providing a more scalable and fault-tolerant solution, 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 may be possible to find a good enough candidate. In big 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 as approximate, search through big solution sets, but in this case it doesn't have the same guarantee of success. It is usually 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 same performance on all nodes. This solution is more effective in systems where the nodes are homogeneous since it allows to make decisions with more precision. The 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 exists the possibility of resources join and leave the setup and changes of local resource usage policies.

ii) Adaptive vs Non-Adaptive: Adaptive approaches take into consideration the history of the systems. Measures such as previous and current behavior of the system are used to better assign the VM. This kind of scheduler usually has some intelligence that makes him take 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 to adaptive schedulers, non-adaptive schedulers don't modify its behaviour based on the history. The same parameter will always weight the same regardless of the history.

3.2.3 Classical Algorithms

Over the years lots of algorithms have been developed and described in the literature. We classified them by their goal in five categories, capacity driven, deadline/goals, interference free, and economics/utility economics. Next we overview 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 straight forward. As a matter of fact, it was the first kind of scheduling policy used, mainly because of its 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. One main advantage is the simplicity, it is both simple to implement and allocate VMs. As opposite one big disadvantage is the low utilization and distribution of the available resources.

Round Robin is also a very simple scheduling policy, but in contrast with the Greedy 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 utilizes the resources in a uniform way helping to balance the system load. However, the algorithm can waste more power that needed if all the machines and under low load.

Weighted Round Robin is an optimized 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 the 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 to share data.

The main problem in this type of scheduling is the number of hosts to allocated since you usually don't know the total number of tasks, creating a lot of difficulties for optimization of cost or performance. In Silva et al. [26] they use adaptive heuristics to optimize this process and to predict the amount of tasks. The approach is based on adapting the parameters at each task completion allowing the system to maximize execution time and reduce idle time.

Deadlines/SLAs:

Service Level Agreements (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, loss, signal-to-noise ratio, cross-talk, echo, etc. In Cloud environments there are some different QoS aspects to consider such as deadline, execution time and overhead. This type of algorithm tries to maximize the parameters to meet the QoS defined previously.

Abrishami et al. [2] proposes an evolution of Partial Critical Paths (PCP) [1] which aims to minimize 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, only launches another as last resort.

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

Particle Swarm Optimization(PSO) [23] bases its evaluation in 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.

Interference free:

Resource interference on OSs is one of the hardest problems with virtualization. Either because we are underprovisioning the VMs 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 utilization, but this ends up resulting in performance interference between the VMs. The most common practice to reduce this effect is overprovisioning resources to help avoid performance interference. This practice is not very interesting because it goes against the main objective, optimize resource usage, simply because we will be giving more resources than VMs need and they will be underutilized.

Recent work still relies on static approaches that suffer from a lot of limitations due to assumptions about the application behaviour that are not certain *a priory*.

DeepDive [21], 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 deep analysis and finally it reassesses the distribution of the VMs. It transparently deals with interference using low-level metrics, including hardware performance counters and readily available hypervisor statistics about each VM.

Stay-Away [22] is a novel approach to this problem that addresses these limitations, by providing a generic and adaptive mechanism to mitigate the performance interference on responsiveness-sensitive applications. This solution continuously learns about the states of execution and tries to predict and prevent any transition that may cause interference by proactively throttling their 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. This kind of 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 the services based on their usage and quality of the service provided [6].

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

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

Cloudpack [10] tries to disrupt static pricing models for resources. A typical data center has different cost based the energy cost, 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 minimize the probability of failure in tasks without loosing revenue by overprovisioning the infrastructure. In Macias et al. [17] they maximize 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.

In Morshedlou et al. [20] two user hidden characteristics are 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 method presented decides which VMs should release their resources, based on each client willingness to pay. This approach is similar to Partial Utility SLAs [27] 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.

3.3 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 clouds became an issue and lots of research has been done in the last years.

3.3.1 Scheduling Aspects

Hardware keeps evolving [31], and with new technologies, such as low–power CPUs, solid state drives and other energy efficient components the energy footprint got smaller. Even though it is 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 in two categories [37], characterized by where they want to reduce energy, in order to achieve lower energy consumption. Power-aware [14,37,4,35] and Thermal-aware [19], 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 maximize the performance without wasting too much power.

Thermal-aware solutions target is to reduce the emissions of heat from the computer components and with it reduce the wasted energy in cooling the machines. Although thermal-aware scheduling does not seem to be directly related to energy consumption, the cooling in these computing facilities consumes huge amounts of energy.

Figure 3 inspired in [37] groups the areas of study for energy aware algorithms and highlights the topics in which we focus our solution.

Power-Aware Scheduling

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

Very common approaches focus specially on CPU and how it is being used, 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 don't keep competing for memory access, but also to reduce communication between them.

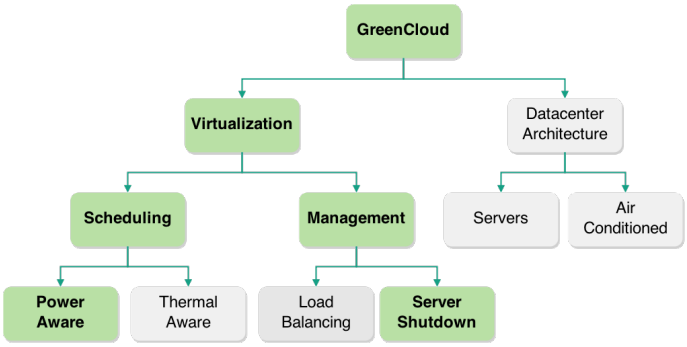

Fig. 3. GreenCloud: Green items represent areas in range for this paper

Virtual machine consolidation is crucial to power management, but it is always a hard problem to solve. Gather all VMs to the same physical machine to avoid wasting energy is not possible because we will overcommit its resources. The best solutions try to gather all the info available, such as SLAs and also previous usage footprints to better allocate the jobs. Every resource matters, but we always have to trade something to better fulfill one objective.

3.3.2 Energy Aspects

The energy used by the machines can be characterized based on its values changing over time or being of constant amount all the time [14].

Static energy consumption is the part of the energy used as a base for the machine. This is all the energy used by the components even when they are in idle mode, not performing any kind of work.

Dynamic energy consumption, as opposite, is calculated in terms of the proportion of resources being utilized 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 have a bad impact in the environment.

When we are managing a datacenter all this parameters need to be taken in consideration. Based on the SLAs defined with the client the provider can improve the 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 night the energy is usually cheaper because the demand is smaller and that can be a good reason to delay a job from a client that doesn't have a deadline.

This kind of approaches are not much common but are definitely a good opportunity to reduce the price for the client and still reduce costs for the provider.

3.3.3 Efficiency Aspects

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

Dynamic Voltage Frequency Scaling

Some of these approaches primary target is dynamic voltage frequency scaling (DVFS)[14,13,19,36]. 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 in half of the needed computing power.

To solve this, some solutions try to distribute workloads by need and are able to have 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, specially when every 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. [37] tries to achieve maximum energy efficiency by combining a greedy algorithm with live migration. It minimizes power consumption within the data center 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. (2010) [3] presents a decentralized architecture of a resource management system for cloud data centers that aims to use continuous optimization policies of VM placement. They look at factors such as CPU, RAM, network bandwidth utilization and physical machines temperature to better reallocate machines and improve overall efficiency.

In Beloglazov et al. (2012) [4] they detect over and under utilization peaks to migrate VMs between hosts and minimize the power consumption in the datacenter.

Von et al. [14] uses a batch scheduling approach that is based on DVFS. The 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) [35] is a solution with three objectives, inter-machine communication and energy redution with load balancing. They group the machines in groups to reduce communication and the allocation is done by finding the machine with the resource availability closer to the request. By controling the information flow they manage to migrate VMs and keep improving the disposition of the VMs.

ThaS (Thermal-aware Scheduler) [19] is different from the previous solutions, it is thermal-aware. It's 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.

All this approaches have tried to reduce energy consumption and improve resource usage, but none that I know, have used the concept of DVFS in conjunction with partial utility. *EcoScheduler* does transparent DVFS scheduling with resource awareness (mainly CPU performance) and tries to achieve maximum request satisfaction using the concept of Partial Utility SLAs [27].

4 Proposed Solution

A high level description of the proposed solution's architecture for *EcoScheduler* is depicted in Figure 5. In our solution, we organize the system as a structured hierarchical network headed by the master scheduler (MS) and where the datacenter is partitioned in sectors that aggregate several physical machines.

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

In Section 4.1 we describe in detail each of the architecture's entities and the interactions between them. In Section 4.2 we describe our energy efficient scheduling algorithm. In Section 4.3 we present the technologies that will support our work.

4.1 Architecture

In this section we present a simple use case, the network topology and the main information exchanged between the entities of our system.

Use case:

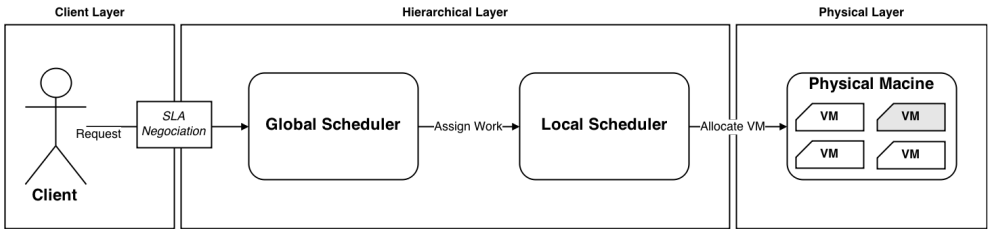

Fig. 4. Use case scenario

The architecture can be divided in three layers, client layer, hierarchical layer and physical layer, as depicted by Figure 4. Figure 4 describes all the steps in the high level process of reservation of the VMs. The client layer, which includes all the clients willing to reserve VMs in our system, communicates with the hierarchical layer via the master node. In the second layer the request will be processed and an allocation will be made having in consideration the established SLAs and also the energy and resource usage objectives of the system. This is accomplished by passing the request to the specified LS selected 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).

Distributed Architecture

To accomplish a scalable scheduling system we chose to use a hierarchical architecture, depicted in greater detail Figure 5. The architecture is composed of two main

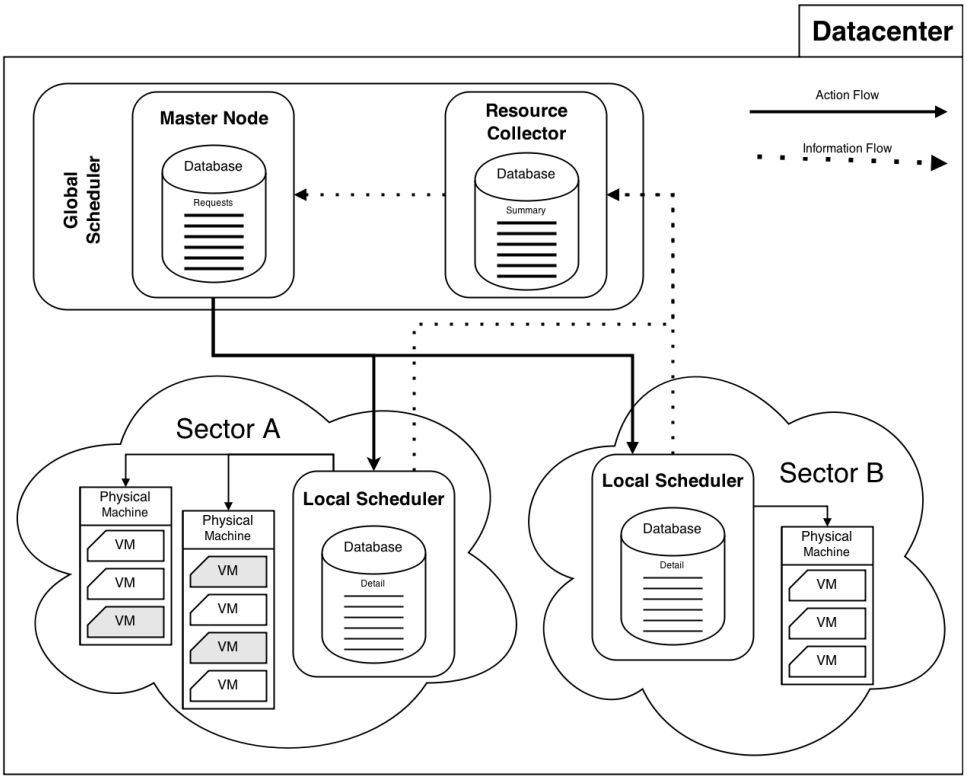

Fig. 5. High level architecture

entities, the global scheduler (GS) and hierarchy the local schedulers (LSs). For small clusters 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.

Each level in the architecture communicates with the upper level by providing information about their current state. 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, RAM, etc.

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 periodically sent to the LS using JSON format. LSs send information about their machines state periodically to the RC. The RC processes all the information and creates a summary that is also periodically sent to the master node. This summary includes average energy efficiency level, maximum CPU available and maximum available memory (disk and RAM).

The power consumed by computing node in a datacenter consists of the consumption by CPU, storage and network communication. In comparison to other system

resources, CPU consumes larger amount of energy [5,15,25], and hence in this work we focus on managing its power consumption and efficient usage. Recent studies, such as [13,36,25], show that an idle server consumes approximately 70% of the power consumed by the server when running at full CPU speed, justifying that servers should be turned off to reduce total power consumption as soon as possible.

4.2 Scheduling Algorithm

As mentioned before, our system has two types of scheduling, global scheduling and local scheduling. Local scheduling happens in all the nodes that are leafs in the hierarchy tree, and basically is the algorithm which really allocates the VM. The upper nodes in the hierarchy are considered global schedulers since they work over summaries of the information.

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 bandwidth available. Our main data structure is a list of the sectors (or hosts) grouped by the average energy efficiency level, that can be implemented as a tree. Each level has the sectors sorted by CPU availability. This structure is easy to maintain and helps finding the most efficient sector with the minimum resources to fulfill the SLAs very fast. In the second part of the local algorithm we use variation of the previous data structure that also groups the hosts by class, used to simplify the second part of the algorithm if increasing DVFS level.

Algorithms

Our scheduling algorithm takes two main properties into consideration, Energy Efficiency Level (EEL) and the CPU Available (measured in MFLOPS) (CPUA). The EEL is partitioned in five levels from A..F (identified as 1..6), A is more efficient and F is the less efficient. This levels are calculated based on the energy (kWh) as a factor of millions of floating point operations (MFLOPS). Each class of machine in the datacenter has a well defined mapping between the DVFS level and the EEL. Algorithm 1 presents the pseudocode for the global scheduling. This scheduling phase acts upon sectors in the datacenter.

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

Algorithm 2 is the generic algorithm for the local scheduling phase. In this phase we are choosing the host where we will allocate the machine. This is very similar to the global scheduling phase because it just tries to find the best match in the available hosts. **FitsCriteria** is the method that implements the comparison of the available resources with the needed resources. **UpdateHostsState** is responsible for updating the lists, to avoid compromising scalability, is very simple and efficient. In the worse case, it needs to remove one element from a list and insert it ordered in other list which has $O(log(n))$ complexity. **Allocate** is a direct request sent to the hypervisor to allocate the VM with the defined resources.

Algorithm 1 Global scheduling: Best-Fit

Require: $sectorsByEEL$ available sectors ▷ grouped by EEL and sorted by CPUA
Require: vm VM to be allocated
1: **function** GlobalScheduling($sectorsByEEL, vm$)
2: $selectedSector \leftarrow null$
3: $eel \leftarrow sectorsByEEL$
4: $sector \leftarrow eel.getSectors()$
5: **do**
6: **if** FitsCriteria($sector, vm$) **then**
7: $selectedSector \leftarrow sector$
8: break
9: **end if**
10: **if** $\neg sector.hasNext() \wedge eel.hasNext()$ **then**
11: $eel \leftarrow eel.next()$
12: $sector \leftarrow eel.getSectors()$
13: **else**
14: $sector \leftarrow sector.next()$
15: **end if**
16: **while** $(sector.hasNext() \wedge sector.MaxCPUA > vm.neededCPU)$
17: **if** $selectedSector = null$ **then** ▷ Fallback to the first sector
18: $selectedSector \leftarrow sectorsByEEL.next().getSectors$
19: **end if**
20: UpdateSectorsState($selectedSector, vm$) ▷ asynchronous call
21: Allocate($selectedSector, vm$)
22: **end function**

Algorithm 2 Local scheduling: Efficiency-Driven Approximate Best-Fit

Require: $hostsByEEL$ available hosts ▷ grouped by EEL and sorted by CPUA
Require: vm VM to be allocated
1: **function** GenericLocalScheduling($hostsByEEL, vm$)
2: $selectedHost \leftarrow null$
3: $eel \leftarrow hostsByEEL$
4: $host \leftarrow eel.getHosts()$
5: **do**
6: **if** FitsCriteria($host, vm$) **then**
7: $selectedHost \leftarrow host$
8: break
9: **end if**
10: **if** $\neg host.hasNext() \wedge eel.hasNext()$ **then**
11: $eel \leftarrow eel.next()$
12: $host \leftarrow eel.getHosts()$
13: **else**
14: $host \leftarrow host.next()$
15: **end if**
16: **while** $(host.hasNext() \wedge host.CPUA > vm.neededCPU)$
17: **if** $selectedHost \neq null$ **then**
18: UpdateHostsState($selectedHost, vm$) ▷ asynchronous call
19: Allocate($selectedHost, vm$)
20: **return** $true$
21: **end if**
22: **return** $false$
23: **end function**

Algorithm 3 Local scheduling: Efficiency-Driven Increasing Best-Fit

Require: $hostsByEEL$ available hosts ▷ grouped by EEL and class, sorted by CPUA
Require: $machineClassesByEEL$ machine classes ▷ sorted by next EEL
Require: vm VM to be allocated

function IncreasingLocalScheduling($hostsByEEL, vm$)
  if GenericLocalScheduling($hostsByEEL, vm$) = $true$ **then**
    return $true$
  end if
  $selectedHost \leftarrow null$
  $eel \leftarrow hostsByEEL$
  $class \leftarrow machineClassesByEEL.getClasses(eel.id)$
  $host \leftarrow eel.getHosts(class.id)$
  do
    if FitsIncrease($host, vm$) **then**
      IncreaseDVFS($selectedSector, vm$)
      $selectedHost \leftarrow host$
      break
    end if
    if $\neg host.hasNext() \wedge class.hasNext()$ **then**
      $class \leftarrow class.next()$
      $host \leftarrow eel.getHosts(class.id)$
    else if $\neg host.hasNext() \wedge \neg class.hasNext() \wedge eel.hasNext()$ **then**
      $eel \leftarrow eel.next()$
      $class \leftarrow machineClassesByEEL.getClasses(eel.id)$
      $host \leftarrow eel.getHosts()$
    else
      $host \leftarrow host.next()$
    end if
  while $(host.hasNext() \wedge host.CPUA > vm.neededCPU)$
  if $selectedHost == null$ **then** ▷ fallback to energy oblivious partial utility scheduling
    $selectedHost \leftarrow getHostByPartialUtility()$
  end if
  UpdateSectorsState($selectedSector, vm$) ▷ asynchronous call
  Allocate($selectedSector, vm$)
  return $true$
end function

Algorithm 3 is the advanced algorithm used when no host can fulfill the VM requirements. This algorithm finds the host which will have the better EEL when increased the DVFS level and that can then allocate the VM. **FitsIncrease** is very similar to **FitsCriteria**, but instead of comparing CPUA it compares future CPUA if DVFS level is increased. **IncreaseDVFS** sends a request to the hypervisor for increasing the DVFS level of a host and updates the lists, having the same impact as **UpdateHostsState**. If after all this we still could not fit the VM in any host, we will apply the Partial Utility SLAs [27]. The method **getHostByPartialUtility** will return a host prepared for allocation after applying the partial utility algorithm.

4.3 Software Architecture and Implementation Issues

Our solution will be implemented in two phases. The first phase will be to implement our solution using a cloud simulator. The simulator will allow us to have an idea of how the solution will behave in a scenario with a big cluster with different kinds of resources. The chosen simulator was CloudSim [7] because it is widely used by many authors, has a lot of needed functions, is easily extensible and made distributed by $Cloud^2Sim$ [11]. Figure 6 inspired by [27] and based on the documentation page [5] depicts the main changes that need to be done in CloudSim to simulate our solution.

In the second phase we will implement our algorithm on a real cloud environment. We will use OpenStack referenced previously in Section 3.1.1 because it is open-source and has a very good community supporting it.

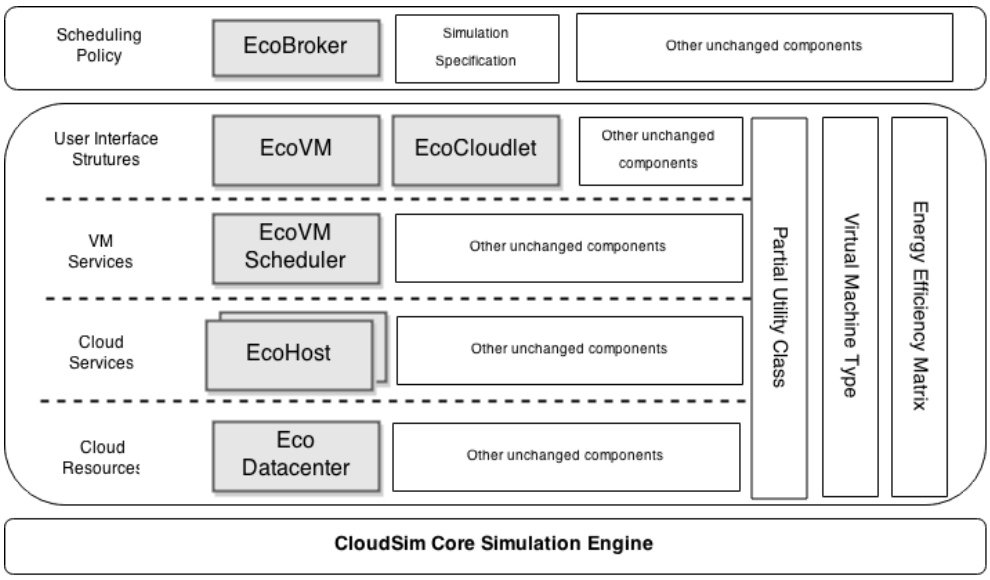

Fig. 6. Highlighted extensions to the CloudSim simulation environment

[5] CloudSim class tree: `http://www.cloudbus.org/cloudsim/doc/api/org/cloudbus/cloudsim/package-tree.html`

5 Evaluation Methodology

In this section we present the metrics and the test environment that we will use to evaluate our solution. We will use the following metrics to evaluate the performance of our algorithm:

- **Makespan (in seconds):** or time since last finished job is used to understand if the scheduler is being performant.
- **Energy consumption (in kWh):** understand the energy profile of the overall system and estimate the costs.
- **Power consumed over time (in kW):** perceive the power supplied to our system and take into consideration its source.
- **Energy per work unit (in kWh/MFLOP):** measure the energy efficiency of the system.
- **Total income (in USD):** overview of the total incoming money payed by the clients.
- **Cost of used resources (in USD):** cost of all the infrastructure hardware and maintenance, used in conjunction with the income to calculate the revenue.
- **Number of direct, delayed and reduced allocations:** used to interpret how the system is allocating the VMs. If it is using Algorithm 2 for direct scheduling, Algorithm 3 for DVFS scaling or using the partial utility reductions.
- **Idle resources:** perceive if the system is wasting energy with idle resources.
- **Average resource usage (in %):** how are the resources in the system being used, specially to understand if we have the system under low usage.
- **Average user satisfaction:** understand how we are fulfilling the SLAs

Our tests will be run on CloudSim simulator to evaluate the scalability and behavior of the solution for large number of sectors and users. CloudSim will be the tool to make a comparison between centralized schedulers approach and our hierarchical solution. We will also implement our solution in a real cloud environment, OpenStack, to assert how the algorithm behaves in a non-simulation scenario.

6 Conclusions

We started this document by presenting the cloud and the importance of virtualization 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 theses topics allowed us to have the necessary knowledge to identify some aspects that have not been explored. Once identified the shortcomings, we proposed a solution that considers the datacenter as a structured hierarchical network divided in sectors with local schedulers that interact with the upper levels by exchanging information about the state of their machines. Our scheduling solution efficiently assigns proper resources to jobs according to their requirements and the energy used. Finally, we presented the metrics that will support the evaluation of our solution.

References

1. Abrishami, S., Naghibzadeh, M., Epema, D.H.: Cost-driven scheduling of grid workflows using partial critical paths. Parallel and Distributed Systems, IEEE Transactions on 23(8), 1400–1414 (2012)

2. Abrishami, S., Naghibzadeh, M., Epema, D.H.: Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Generation Computer Systems 29(1), 158–169 (2013)
3. Beloglazov, A., Buyya, R.: 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. pp. 826–831. CCGRID '10, IEEE Computer Society, Washington, DC, USA (2010), http://dx.doi.org/10.1109/CCGRID.2010.46
4. Beloglazov, A., Buyya, R.: 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 24(13), 1397–1420 (2012)
5. Buyya, R., Beloglazov, A., Abawajy, J.: 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)
6. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems 25(6), 599–616 (2009)
7. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 41(1), 23–50 (2011)
8. Casavant, T.L., Kuhl, J.G.: A taxonomy of scheduling in general-purpose distributed computing systems. Software Engineering, IEEE Transactions on 14(2), 141–154 (1988)
9. Daley, R.C., Dennis, J.B.: Virtual memory, processes, and sharing in multics. Commun. ACM 11(5), 306–312 (May 1968), http://doi.acm.org/10.1145/363095.363139
10. Ishakian, V., Sweha, R., Bestavros, A., Appavoo, J.: Cloudpack. In: Middleware 2012, pp. 374–393. Springer (2012)
11. Kathiravelu, P., Veiga, L.: An elastic middleware platform for concurrent and distributed cloud and map-reduce simulation-as-a-service. IEEE Transactions on Cloud Computing (2014)
12. Katzan, Jr., H.: Operating systems architecture. In: Proceedings of the May 5-7, 1970, Spring Joint Computer Conference. pp. 109–118. AFIPS '70 (Spring), ACM, New York, NY, USA (1970), http://doi.acm.org/10.1145/1476936.1476960
13. Kusic, D., Kephart, J.O., Hanson, J.E., Kandasamy, N., Jiang, G.: Power and performance management of virtualized computing environments via lookahead control. Cluster computing 12(1), 1–15 (2009)
14. von Laszewski, G., Wang, L., Younge, A.J., He, X.: Power-Aware Scheduling of Virtual Machines in DVFS-enabled Clusters. In: Proceedings of the 2009 IEEE International Conference on Cluster Computing (Cluster 2009). IEEE, New Orleans (31 Aug – Sep 4 2009), http://cyberaide.googlecode.com/svn/trunk/papers/09-greenit-cluster09/vonLaszewski-cluster09.pdf
15. Lee, Y.C., Zomaya, A.Y.: Energy efficient utilization of resources in cloud computing systems. The Journal of Supercomputing 60(2), 268–280 (2012)
16. Lett, A.S., Konigsford, W.L.: Tss/360: A time-shared operating system. In: Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I. pp. 15–28. AFIPS '68 (Fall, part I), ACM, New York, NY, USA (1968), http://doi.acm.org/10.1145/1476589.1476593
17. Macias, M., Guitart, J.: A risk-based model for service level agreement differentiation in cloud market providers. In: Distributed Applications and Interoperable Systems. pp. 1–15. Springer (2014)
18. Meyer, R., Seawright, L.: A virtual machine time-sharing system. IBM Systems Journal 9(3), 199–218 (1970)
19. Mhedheb, Y., Jrad, F., Tao, J., Zhao, J., Kołodziej, J., Streit, A.: Load and thermal-aware vm scheduling on the cloud. In: Algorithms and Architectures for Parallel Processing, pp. 101–114. Springer (2013)

20. Morshedlou, H., Meybodi, M.: Decreasing impact of sla violations: A proactive resource allocation approach for cloud computing environments. IEEE Transactions on Cloud Computing p. 1 (2014)
21. Novaković, D., Vasić, N., Novaković, S., Kostić, D., Bianchini, R.: 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)
22. Rameshan, N., Navarro, L., Monte, E., Vlassov, V.: Stay-away, protecting sensitive applications from performance interference. In: Proceedings of the 15th International Middleware Conference. pp. 301–312. ACM (2014)
23. Rodriguez Sossa, M., Buyya, R.: Deadline based resource provisioning and scheduling algorithmfor scientific workflows on clouds (2014)
24. Schopf, J.M.: Grid resource management. chap. Ten Actions when Grid Scheduling: The User As a Grid Scheduler, pp. 15–23. Kluwer Academic Publishers, Norwell, MA, USA (2004), http://dl.acm.org/citation.cfm?id=976113.976116
25. Sharifi, L., Rameshan, N., Freitag, F., Veiga, L.: Energy efficiency dilemma: P2p-cloud vs. datacenter. IEEE Transactions on Cloud Computing (2014)
26. Silva, J.N., Veiga, L., Ferreira, P.: A2ha—automatic and adaptive host allocation in utility computing for bag-of-tasks. Journal of Internet Services and Applications 2(2), 171–185 (2011)
27. Simao, J., Veiga, L.: Partial utility-driven scheduling for flexible sla and pricing arbitration in clouds. IEEE Transactions on Cloud Computing (2013)
28. Smith, J., Nair, R.: Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)
29. Vasques, J.L.V.: A decentralized utility-based scheduling algorithm for grids
30. Vecchiola, C., Pandey, S., Buyya, R.: High-performance cloud computing: A view of scientific applications. In: Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks. pp. 4–16. ISPAN '09, IEEE Computer Society, Washington, DC, USA (2009), http://dx.doi.org/10.1109/I-SPAN.2009.150
31. Venkatachalam, V., Franz, M.: Power reduction techniques for microprocessor systems. ACM Computing Surveys (CSUR) 37(3), 195–237 (2005)
32. VMware: Understanding full virtualization, paravirtualization and hardware assist (2007), http://www.vmware.com/resources/techresources/1008
33. Waldspurger, C.A.: Memory resource management in vmware esx server. ACM SIGOPS Operating Systems Review 36(SI), 181–194 (2002)
34. Wang, L., Von Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J., Fu, C.: Cloud computing: a perspective study. New Generation Computing 28(2), 137–146 (2010)
35. Wang, S.H., Huang, P.W., Wen, C.P., Wang, L.C.: 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)
36. Wu, C.M., Chang, R.S., Chan, H.Y.: A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters. Future Generation Computer Systems 37, 141–147 (2014)
37. Younge, A.J., Von Laszewski, G., Wang, L., Lopez-Alarcon, S., Carithers, W.: Efficient resource management for cloud computing environments. In: Green Computing Conference, 2010 International. pp. 357–364. IEEE (2010)

A Algorithm characteristics

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

Table 1. Algorithms characteristics

B Planning

Planning		
Tasks	**Details**	**Duration**
Introduction to CloudSim	- Code Study - Tutorials and Examples	January (2 weeks) January (1 week)
Implementation on CloudSim	- Set up network topology - Implement algorithm	February (1 week) February (2 weeks)
Implementation on Cloud environment	- Install OpenStack - Set up network topology - OpenStack API's - Implement algorithm	March (1 week) March (1 week) March (1 week) March (1 week) April (4 weeks)
Conclusion of Implementation	- Conclude any unimplemented functionality	May (4 weeks)
Performance Measurements	- Evaluate implemented solution	June (4 weeks) July (2 weeks)
Thesis final report writing	- Write thesis report	July (2 weeks) August (4 weeks)
Review and Submission	- Report review and submission	September (2 weeks)
Documentation	- Document design choices, code and tests	January - September
Bi-weekly meetings	- Analyze the progress of the work	January - September

Table 2. Planning schedule