Green-Cloud

Economics-inspired Scheduling Energy and Resource Management in Cloud Infrastructures

Rodrigo Fernandes
rodrigo.fernandes@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2015

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 (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

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. 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.

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 Dynamic Voltage Frequency Scaling (DVFS) for improved energy efficiency. Our scheduling algorithm proposes to efficiently assign proper resources to jobs according to their requirements.

2 Related Work

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.

2.1 Scheduling Aspects

Hardware keeps evolving [13] 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 [17], characterised by how they want to reduce energy. Power-aware [14, 17, 2, 15] and Thermal-aware [9] 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.

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 waste in a physical machine.

2.2 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 [14].

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.

2.3 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)[14, 7, 9, 16]. DVFS dynamically scales the processor frequency according to the global CPU load and regardless of the Virtual machine (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.

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. [17] 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. [1] 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. [2] detect overutilisation and underutilisation peaks to migrate VMs between hosts and minimise the power consumption in the datacenter.

Von et al. [14] 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) [15] 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.

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.

3 Solution

The algorithms presented in Section 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 [5], 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.

3.1 Use case

The architecture can be divided into three layers: client layer, hierarchical layer, and physical layer, as depicted in Figure 1.

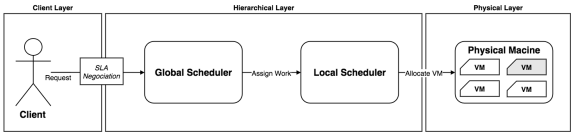

Figure 1: Use case scenario

Figure 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 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

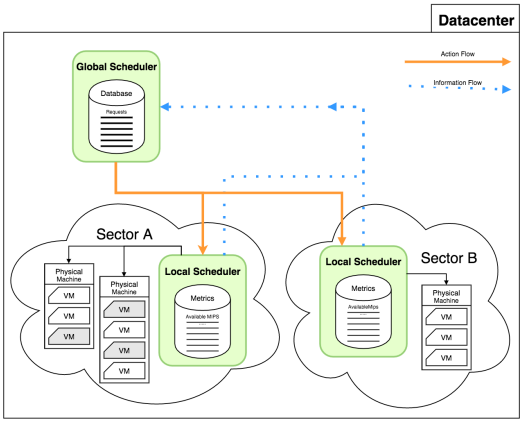

Figure 2: High-level architecture

A high-level description of the proposed solution's architecture for *EcoScheduler* is depicted in Figure 2.

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

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 [3, 8, 10]. Hence, in this work we focus on managing its power consumption and efficient usage. Recent studies, such as [7, 16, 10], 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. 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 [5].

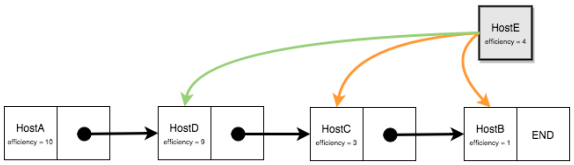

Figure 3: Ordered linked list insert

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

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 [5] 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.

$$CPUPower = HostPower * \frac{CurrentCPUMIPS}{HostTotalMIPS} \quad (1)$$

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 (3), we obtain the

percentage of the power used by the $Host$, $HostPower$ corresponding to the mips being used the CPU we are calculating.

$$EEL = \frac{\sum CPUPower * \frac{VMsTotalMaxMIPS}{HostTotalMaxMIPS}}{CPUsNumber} \quad (2)$$

Then, in Equation (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 ← sectors.first
3:     sector ← sectors
4:     do
5:         if FitsCriteria(sector, vm) then
6:             selectedSector ← sector
7:             break
8:         end if
9:         sector ← sector.next()
10:    while (sector.hasNext())
11:    if selectedSector = null then
12:        selectedSector ← sectors.first()
13:    end if
14:    UpdateSectorsState(selectedSector, vm)
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.

If all the sectors fail this check, we will try to allocate on the first sector since 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 *FitsCriteria*, 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.

The generic algorithm for the local scheduling phase, is very similar to Algorithm 1, in fact the only difference it that it acts upon hosts instead of sectors.

When an allocation is performed successfully, method **UpdateHostsState** is invoked. This method is responsible for triggering updates in the parent sectors, 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 2 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 [11, 12] algorithm on the other VMs.

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.

Algorithm 2 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**
16: $selectedHost \leftarrow decreaseVMMips(vm)$
17: **end if**
18: UpdateSectorsState($selectedSector, vm$)
19: Allocate($selectedSector, vm$)
20: **return** *true*
21: **end function**

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.

4 Implementation

Our solution was implemented in a state of the art cloud simulator, Cloudsim [4]. 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 $Cloud^2Sim$ [6].

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 [5], 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. [5], *PowerVmAllocationPolicySimpleWattPerMipsMetric* and *PowerVmAllocationPolicyDVFSMinimumUsedHost*.

4.1 Overall implementation approach

4.1.1 Cloudsim architecture

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

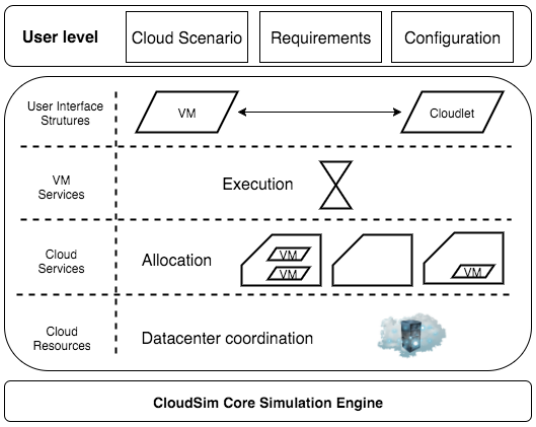

Figure 4: Cloudsim organization layers

Figure 4 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 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 5 highlights the main component changes: *DatacenterBroker*, *VM*, *Host*, *AllocationPolicy*, and *Governor*.

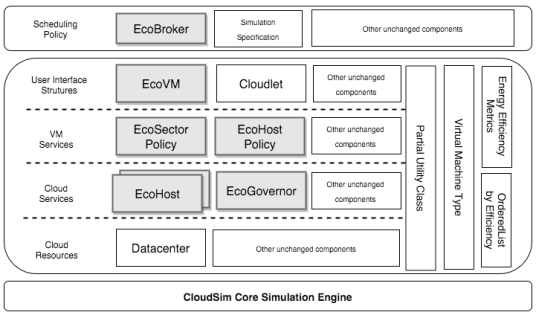

Figure 5: Highlighted extensions to the CloudSim simulation environment

Datacenter broker To capture revenue metrics, we created the *EcoBroker* which extends the *PowerDatacenterBroker*. 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 [12].

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 6, 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.

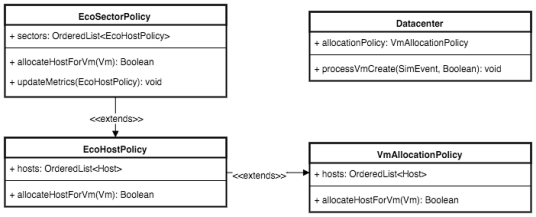

Figure 6: Allocation policy hierarchical extension to CloudSim

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 *OrderedList*, 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 2.

5 Evaluation

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

Table 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 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

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.

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 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 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 3. With the first type, we offered less CPU power with higher 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

Table 3: Host sizes

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

Table 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

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 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 2, since it is becoming a common practice in the literature [1].

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. [5].

Figure 7 and Figure 8 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).

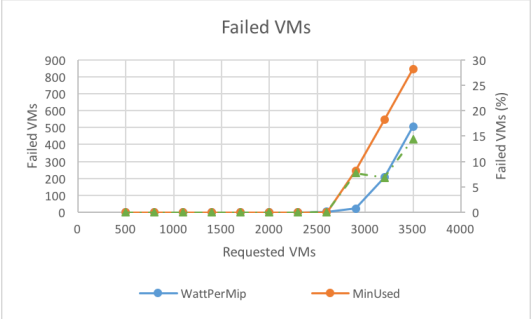

Figure 7: Size 1 - Failed allocations

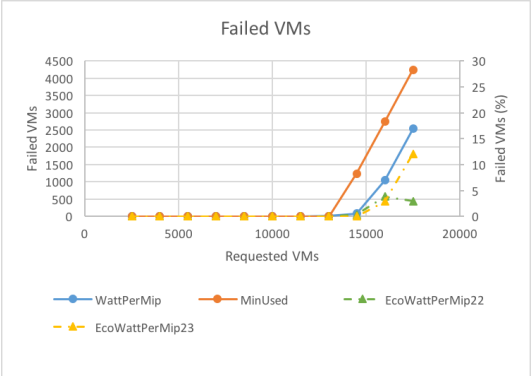

Figure 8: Size 2 - Failed allocations

In the largest test case, depicted in Figure 9, 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.

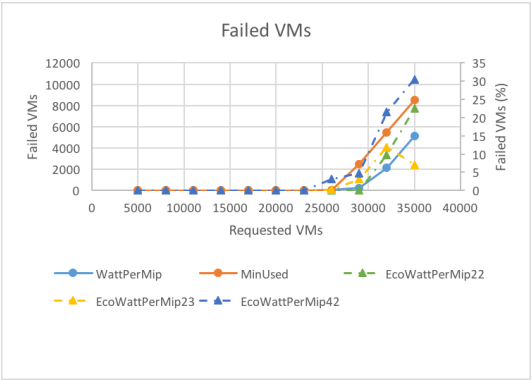

Figure 9: Size 3 - Failed allocations

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).

5.3 Energy efficiency

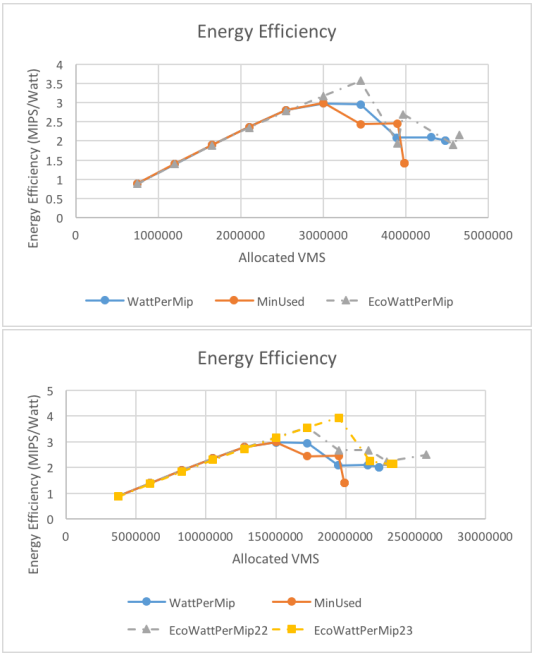

Figure 10: Energy Efficiency

While aiming for energy efficiency, we intended to execute more MIPS using the same Watts. To measure this, we used Equation (3), the relation between the real allocated MIPS in each algorithm, divided by the sum of the used power.

$$Efficiency = \frac{RequestedMips - FailedMIPS}{PowerSum} \quad (3)$$

Depicted in Figure 10, 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.

5.4 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 [2]. 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 11, our policy has an average execution time that matches the execution time of the

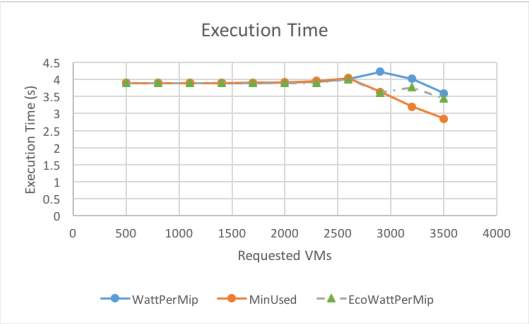

Figure 11: Size 1 - Execution times

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 11, we depict the percentage of the differences between our algorithm and the minimum, maximum, and average of the other algorithms.

6 Conclusion

In this paper, we propose a solution that extends some of the models being worked by the academic researchers and try to help in solving big 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

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.

References

[1] 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, pages 826–831, Washington, DC, USA, 2010. IEEE Computer Society.

[2] 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*, 24(13):1397–1420, 2012.

[3] 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*, pages 6–17. CSREA Press, 2010.

[4] 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*, 41(1):23–50, 2011.

[5] 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*, 39:76–91, 2013.

[6] 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.

[7] 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*, 12(1):1–15, 2009.

[8] Y. C. Lee and A. Y. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.

[9] 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*, pages 101–114. Springer, 2013.

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

[11] 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*, volume 1, pages 274–281. IEEE, 2013.

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

[13] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys (CSUR)*, 37(3):195–237, 2005.

[14] 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, 31 Aug. – Sep. 4 2009. IEEE.

[15] 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*, pages 220–225. IEEE, Feb 2014.

[16] 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*, 37:141–147, 2014.

[17] 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*, pages 357–364. IEEE, 2010.