ENERGY ADAPTIVE INFRASTRUCTURE FOR SUSTAINABLE CLOUD DATA CENTRES

Corentin Dupont

International Doctorate School
in Information and Communication Technologies
University of Trento
April 2016

New technologies provide benefit only if you drop the rules that were designed to overcome the shortcomings of the old technology.
– Dr Eliyahu Goldratt

Wings are a constraint that makes it possible to fly.
– Robert Bringhurst

To my parents, Sylvie and Jean-Luc,
to my brother Nicolas and my sister Marie,
and to my beloved Anna.

Acknowledgements

Making a PHD is a once in a life time experience. It's a long journey, however the journey often is more important than the destination. I am very grateful to have been given the opportunity to make it, and I feel that I have learnt a lot on the way. I thank Raffaele Giaffreda that fought for allowing me to do this thesis within the research centre Create-Net. I thank Fabien Hermenier for his support all along this process. His advices were invaluable. I thank Imrich Chlamtac and Create-Net for the support and for providing me with a good work environment. I thank my family for their continuous love and support.

I would also like to thank all the consortium members of the EU FP7 projects FIT4Green and DC4Cities. A special thanks goes to the trial teams of these projects. I thank the research teams RIOT and Smarti of Create-Net. Scaling experiments presented in this thesis were carried out using the Grid'5000 experimental testbed [1], being developed by INRIA with support from CNRS, RENATER and several universities as well as other funding bodies. My last thanks goes to my life partner Anna, which supported me during those long and tense months of writing!

[1] https://www.grid5000.fr

Abstract

With the raising concerns about the environment, the ICT equipments have been pointed out as a major and ever rising source of energy consumption and pollution. Among those ICT equipments, data centres play obviously a major role with the rise of the Cloud computing paradigm. In the recent years, researchers have focused on reducing the energy consumption of data centres. Furthermore, future environmentally friendly data centres are also expected to prioritize the usage of renewable energies over brown energies. However, managing the energy consumption within a data centre is challenging because data centres are complex facilities which supports a huge variety of hardware, computing styles and SLAs. Those may evolve through time as user requirements can change rapidly. Furthermore, differently from non-renewable energy sources, the availability of renewable energies is very volatile and time dependent: *e.g.* solar power is obtainable only during the day, and is subject to variations due to the meteorological conditions. The goal in this case is to shift the workload of running applications, according to the forecasted availability of the renewable energy. In this thesis we propose a flexible framework called Plug4Green able to reduce the energy consumption of a Cloud data centre. Plug4Green is based on the Constraint Programming paradigm, allowing it to take into account a great number of constraints regarding energy, hardware and SLAs in data centres. We also propose the concept of an energy adaptive software controller (EASC), able to augment the usage of renewable energies in data centres. The EASC supports two kind of applications: service-oriented and task-oriented applications; and two kind of computing environments: Infrastructure as a Service and Platform as a Service. We evaluated our solutions in several trials executed in the testbeds of Milan and Trento, Italy. Results show that Plug4Green was able to reduce the power consumption by 27% in the Milan trial, while the EASC was able to augment the renewable energy percentage by 7.07pp in the Trento trial.

Key words: Data Centre, Renewable Energy, VM consolidation, Job Scheduling, Constraint Programming, Platform as a Service, Infrastructure as a Service

Publications

We list here the publications by the author that are related to this thesis:

- Sonja Klingert, Florian Niedermeier, Corentin Dupont, Giovanni Giuliani, Thomas Schulze, and Hermann de Meer. *Introducing Flexibility into Data Centers for Smart Cities*. Communications in Computer and Information Science, 2016
- Corentin Dupont, Mehdi Sheikhalishahi, Federico M. Facca, and Fabien Hermenier. An energy aware application controller for optimizing renewable energy consumption in cloud computing data centres. In *8th IEEE/ACM International Conference on Utility and Cloud Computing*, 2015
- Corentin Dupont and Fabien Hermenier. DC4Cities: Better usage of the renewable energies in data centres. In *ICT4S 2015*, 2015
- Corentin Dupont, Mehdi Sheikhalishahi, Federico M. Facca, and Silvio Cretti. Energy efficient data centres within smart cities: Iaas and paas optimizations. In *2015 EAI International Conference on Smart Grids for Smart Cities*, Toronto, Canada, 2015
- Sonja Klingert, Florian Niedermeier, Corentin Dupont, Giovanni Giuliani, Thomas Schulze, and Hermann de Meer. Renewable energy-aware data centre operations for smart cities - the DC4Cities approach. In *SMARTGREENS 2015*. ACM, 2015
- Corentin Dupont, Fabien Hermenier, Thomas Schulze, Robert Basmadjian, Andrey Somov, and Giovanni Giuliani. Plug4green: A flexible energy-aware vm manager to fit data centre particularities. *Ad Hoc Networks*, pages 505–519, 2014
- Corentin Dupont. Building application profiles to allow a better usage of the renewable energies in data centres. In *Energy-Efficient Data Centers*, Lecture Notes in Computer

Science, 2014

- Corentin Dupont. Energy aware infrastructure for green cloud data centres. University of Trento Doctoral School, 2014
- Corentin Dupont. Renewable energy aware data centres: The problem of controlling the applications workload. In Sonja Klingert, Xavier Hesselbach-Serra, MariaPerez Ortega, and Giovanni Giuliani, editors, *Energy-Efficient Data Centers*, volume 8343 of *Lecture Notes in Computer Science*, pages 16–24. Springer Berlin Heidelberg, 2013
- Corentin Dupont, Thomas Schulze, Giovanni Giuliani, Andrey Somov, and Fabien Hermenier. An energy aware framework for virtual machine placement in cloud federated data centres. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, pages 4:1–4:10. ACM, 2012
- Dang Minh Quan, Robert Basmadjian, Hermann de Meer, Ricardo Lent, Toktam Mahmoodi, Domenico Sannelli, Federico Mezza, Luigi Telesca, and Corenten Dupont. Energy efficient resource allocation strategy for cloud data centres. In Erol Gelenbe, Ricardo Lent, and Georgia Sakellari, editors, *Computer and Information Sciences II*, pages 133–141. Springer London, 2012
- Dang Minh Quan, Andrey Somov, and Corentin Dupont. Energy usage and carbon emission optimization mechanism for federated data centers. In *Proceedings of the First International Conference on Energy Efficient Data Centers*, E2DC'12, pages 129–140, 2012

Submitted articles:

- Corentin Dupont, Mehdi Sheikhalishahi, and Michele Santuari. Improving renewable energy consumption in iaas/paas hybrid data centres. Submitted to Futur Generation Computer Systems, 2016

Contents

List of Figures

List of Tables

1 Introduction

Data centres are large facilities which purpose is to host information processing and telecommunication services for scientific and/or business applications. The energy consumed by them in 2010 accounted for between 1.1% and 1.5% of the total electricity consumed worldwide [14], making of data centre energy management an important challenge for researchers. A large amount of research on data centres has been focused on improving metrics like performance, reliability, and availability. However, due to the rise in service demands together with energy costs, the energy efficiency has now been added as a new key metric for data centres. Indeed, the prices for electricity are constantly getting higher and carbon emissions to the environment are increasing every year. Energy-aware strategies are beginning to be integrated inside the data centre resource manager. As an example, Virtual Machine (VM) placement algorithms consider the data centre and the workload characteristics to place the VMs among the servers in the most efficient way, considering performance and energy consumption. This placement must be done respecting the requirements of the Service Level Agreement (SLA) existing between the data centre and its clients.

Furthermore, with the recent adoption of renewable energies to power data centres [15], the research community enlarges its vision to associate with purely *quantitative* energy consumption reduction, the notion of *quality* of the energy consumed, *i.e.* the capacity to rely as much as possible on sustainable power sources. Differently from non-renewable energy sources, the availability of renewable energies is very volatile and time dependent: *e.g.* solar power is obtainable only during the day, and is subject to variations due to the meteorological conditions. The goal in this case is to shift the workload of running applications, according to the forecasted availability of the renewable energy (see Figure 4.7).

This thesis presents the usage of heuristics and meta-heuristics for the management of energy

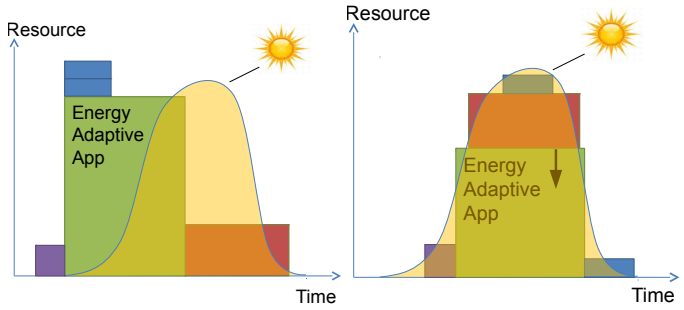


Figure 1.1: Adapting applications for a better usage of renewable energies

in data centres. It aims at achieving two goals:

- save energy in data centres,
- increase the usage of renewable energies in data centres.

Those two objectives were pursued within two typical layers of modern Cloud computing data centres: the Infrastructure as a Service (IaaS) service model and the Platform as a Service (PaaS) service model.

1.1 IaaS optimization

Infrastructure as a Service (IaaS) is a service model that delivers computer infrastructure on an outsourced basis to support enterprise operations. In order to save energy in data centres, numerous studies showed that it is most efficient to use the IaaS layer infrastructure in order to consolidate virtual machines on the most efficient servers, and then switch off the unused servers. This technique is allowed due to the paradigm of virtualization. Virtualization is an old concept, which allows to simulated the behavior of some system, within another system. Such a system can be an operating system, a physical server, a storage device or network resources. The initial success of the Cloud paradigm is due to the possibility to embed practically any legacy applications within VMs, which are managed by an external stakeholder. This permits to relieve the application owner from managing physical infrastructures. The virtualization had the side effect that easy workload consolidation is now possible: several VMs can be hosted on a single physical server. This consolidation is solving a long-standing problem in traditional data centres: the under-utilization of servers. Additional consolidation is possible in environments that permits VM migrations. This consolidation also allows to save energy, by emptying and switching off under-utilized servers. We present in this thesis several algorithms and a framework called *Plug4Green* able to perform an energy-aware consolidation under a

great number of constraints.

1.2 PaaS optimization

Platform as a Service (PaaS) is a category of cloud computing service that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and deploying an application. We found out that the PaaS paradigm offers additional opportunities for energy management. Specifically, at PaaS level the framework has additional knowledge about the applications being run than at IaaS level. Typically, a PaaS framework will compile an application from its source code, and then deploy it inside light weight virtual machines, or containers. This compilation and deployment is done with the help of a manifest file, which describes the configuration and resource needs for the application. Furthermore, PaaS environments usually offers an interface to scale up or down applications, or to schedule various tasks within the applications. This uniform interface offers us a great opportunity with respect to the energy management. In this thesis, we propose to instrument the PaaS framework in order to increase the usage of renewable energies, by using this additional flexibility offered by the framework.

1.3 Application optimization

In order to further increase the usage of renewable energies, we can directly control some key applications and processes of a data centre. This includes applications such as virus scanning, database cleaning and video processing. Those applications are captured in two different models: task-oriented and service-oriented applications. In the first case, the workload of the application is performed within different tasks that are scheduled, such as off-line video transcoding or web crawling. In some cases, those tasks can be post- or pre-poned. We take advantage of this possibility to schedule tasks at the best moment (obviously compliant with the underlying contract). Our prototype then monitors the progress of the task and its KPIs and modifies the schedule if necessary. Service-oriented applications, on the other hand, have a continuous service to perform, such as serving web pages. In this case, we tune the performance of the application within an identified range while respecting the SLA. For example, in the case of a web server, we increase/decrease the number of client threads within the boundaries of the SLA.

1.4 Thesis organization

After this introduction, the chapter 2 will be stating our research problem and objectives. This thesis is then articulated in four technical chapters divided in two parts. The first part is aiming at finding techniques for reducing the energy consumption of data centres, while the second part is aiming at increasing the usage of the renewable energies. The first chapter in the first part, entitled "Heuristics for virtual machine consolidation", presents heuristics able to migrate and consolidate VMs in federated data centres. The objective being to minimize both energy consumption and CO_2 emissions. The second chapter, "Plug4Green, an energy-aware VM placement framework", goes beyond those techniques and proposes an energy aware framework called *Plug4Green*, able to cope with the complex and changing nature of data centres. Plug4Green is based on a meta-heuristic called Constraint Programming, allowing it to take into account a great number of constraints regarding Energy, hardware and SLAs in data centres. In the second part, the chapter "The EASC, an energy adaptive software controller" acknowledges the transition of the current objectives to use better energies. We present in this chapter a framework and algorithms able to increase the use of the renewable energies in Cloud data centres, for two kinds of applications: service-oriented and task-oriented. The last chapter, "Energy optimizations within the PaaS and IaaS paradigms", shows the opportunities that the modern Cloud data centres can provide regarding the usage of renewable energies. Specifically, we demonstrate how the Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) paradigms can interact to provide a better control of the energy consumption. Finally the chapter 7 will conclude this thesis, summing up the research results and suggesting future research directions.

The study of the related works is located within each of the chapters (instead of having one related work section for the full thesis). This allows to present related works that are more focused on the specific topics addressed by each of the chapters, namely related works on heuristics for VM positioning (in Section 3.2), energy aware extensible frameworks (in Section 4.2), server power models (in Section 4.2), renewable energy scheduling systems (in Section 5.2) and finally PaaS/IaaS optimizations and energy models (in Section 6.2).

2 Research problem and objectives

2.1 Research problem

This thesis tackles the research challenges in relation with the green management of energy in Cloud data centres. In particular, the research problems tackled are:

- How to optimize the placement of virtual machines in order to reduce the energy consumption of the DC,
- How to cope with the great number of constraints that exists in data centres, including infrastructure, SLA and energy constraints,
- How to cope with the extensibility needed by the optimization engine in order to include new constraints that may arise,
- How to optimize the workload of applications in order to maximize the usage of renewable energies in DCs,
- How to use the specific opportunities offered by PaaS and IaaS data centres for the usage of renewable energies,
- How to evaluate the energy consumption of applications in a shared infrastructure.

A great challenge of efficiently using the renewable energies in a data centre is to be able to schedule correctly the workload of the applications. Indeed, the availability of renewable energy can have a great variation in time, with comparison to brown energies. To increase the use of renewable energies with respect to brown energies, it is necessary to shift in time the workload of some applications in the data centre. This shows the importance of being able to know the workload that an application will have to run at a certain point of time, in order

to understand under what conditions it can be shifted or delayed, and *in fine* to schedule it correctly.

A current trend is to make the applications running in data centres more and more aware of their self workload. Those applications should be able to predict how much computing power they will require and when. A recent terminology for such applications is Cloud-native[1]. However in data centres, the knowledge of the requirements of an application in terms of resources is still "meta-knowledge", i.e. the knowledge of the data centre operator/application owner. It is the role of the data centre operator to provision sufficient resources for an application, and this provision is often done in a static way. For example, in data centres, database indexing maintenance operations are usually performed at night, to minimize the impact on the overall performance. However, in a data centre using primarily solar power, it would be interesting to shift this task during the lunch break, when the sun is shining. The knowledge that this particular task, "database indexing", can cope with a 12 hour shift, and that it takes approximately half an hour, belongs to the operator's knowledge. It is a very coarse grained and subjective knowledge. This advocates the need for:

- a standardized format and protocol for applications to advertise in real-time their own needs in term of resources, including possible performance trade-offs and uncertainty ranges,
- a data centre management framework and algorithms able to read the application profiles and use them to consolidate and schedule the applications on the servers in the appropriate way, in order to minimize a given utility function,
- a library and programming methodology to allow an external process to control the application load to some extent.

2.2 Objectives

The objective of this thesis is to tackles the research challenges in relation to energy efficient and energy adaptiveness in Cloud data centres. We define two main research objectives with regard to energy management in data centres:

1. save energy,
2. increase the usage of renewable energies.

[1] https://cncf.io/

2.2.1 Objective 1: save energy

The goal of saving energy in data centres have been tackled in a great number of research works [16][17]. In this thesis, we will try to place this goal within the complexity of current data centres. The following research problems are investigated:

- How to design scalable VM consolidation algorithms, based on a high number of constraints.
- How to capture the complexity of the data centre within a specific language, as input of the consolidation algorithm.
- How to find VM placement solutions in a short amount of time.

2.2.2 Objective 2: increase the usage of renewable energies

Relative to the second objective, the following research problems are investigated:

- How to make applications workload more flexible via an extended SLA.
- How to control the performance level of the applications.
- How to tune the PaaS and IaaS infrastructures to make sure that a performance change at application level translates into an energy consumption change.
- How to modelize the energy consumption of PaaS infrastructures.
- How to design workload scheduling algorithms, that increase the usage of renewables.

The crucial aspect of this objective is to identify the flexibility offered by an application. This flexibility need to be encoded in an extended SLA. For example, some workload have the possibility to be post-poned to match the availability of solar energy. Modern data centres uses advanced management infrastructure such as IaaS and PaaS. These layers should also be investigated in order to make sure that the workload management will effectively translate into energy savings.

2.3 Methodology

The research took place within the FP7 European projects FIT4Green[2] and DC4Cities [3]. The research presented in this thesis was performed following the steps below:

[2] http://www.fit4green.eu/
[3] http://www.dc4cities.eu/en/

- Study of the literature and technology evaluation.
- Elicitation of the requirements.
- Creation of the architecture of the prototypes.
- Implementation of the prototypes in Java.
- Evaluation of the prototypes in real-life trials and simulations.

2.3.1 Research models and algorithms

After a study of the literature, open problems are identified and described. Various candidate technologies are evaluated (such as simulated annealing, genetic algorithms for optimization problems). Following this study, several solutions are proposed and the corresponding algorithms are designed.

2.3.2 Requirement elicitation

The requirement elicitation is a process were we list what exactly the prototype should do. There is a strict formalism to respect. Specifically, the requirements should be consistent, non redundant, complete, unambiguous, testable, clear, correct, understandable, feasible, independent, atomic, necessary and finally implementation-free [4]. Furthermore, the requirements are separated in functional and non-functional requirements. A functional requirement is feature oriented, whereas a non-functional requirement is quality and performance oriented. Requirements are also grouped by the prototype component that they address.

2.3.3 Architecture

The architecture is created after the requirements elicitation process has terminated, and should fulfil every of them. We first define the actors that interact with the prototype. Those actors are either end-user humans, or external components. Then a list of detailed use cases shows examples or usage of the prototype by the various actors. Class diagrams shows the internal components and their interactions. Finally detailed APIs are produced to define the format of the data exchanges between the internal components and also with the external components.

[4] http://www.ibmpressbooks.com/articles/article.asp?p=1152528&seqNum=4

2.3.4 Implementation

The implementation was done with Java as a primary language for development. Maven[5] is used to control the development life-cycle. It's an open source tool that allows to formalise the development process and software life-cycle. Git is used as a source configuration manager and GitHub/Gitlab[6] to manage and share the Git repositories. To avoid developing a massive monolithic repository, it was decided to rely on multiple ones. In practice, each software component that has its own life-cycle will have its dedicated repository. Jenkins[7] and Travis[8] are used as continuous integration servers. Continuous integration is an important task in modern development processes, consisting in regularly running test suites on the source code, based on a controlled and reproducible environment. Sonar[9] is used to analyse the source code quality against common development rules for Java projects.

2.3.5 Evaluation

Evaluation of the prototype was performed in two complementary environments: trial and simulation. The prototypes were evaluated in the trial setup by the projects FIT4Green and DC4Cities. Those two projects defined and built several trial labs located in different sites. For example, DC4Cities trials were performed in Barcelona, Milan and Trento. The algorithms presented in this thesis were evaluated within each of the trials, and then integrated inside the project full prototypes. The simulations, on the other hand, were used to validate the prototypes with a larger scale that what could be achieved in trials: large number of servers, VMs and applications. As the target system is a Cloud computing environment that is intended to create a view of infinite computing resources to the users, it is essential to evaluate the proposed algorithms on a large-scale virtualized data centre infrastructure. However, conducting repeatable large-scale experiments on a real infrastructure is extremely difficult. Therefore, to ensure the repeatability and reproducibility of experiments, as well as carry out large-scale experiments, simulation is used as the initial way to evaluate the performance of the proposed algorithms.

[5] http://maven.apache.org
[6] http://maven.apache.org
[7] http://jenkins.org
[8] https://travis-ci.org/
[9] http://www.sonarqube.org/

2.4 Contributions

The contributions of this thesis are articulated around the two objectives above. Two Open Source software are contributed: Plug4Green[10] and EASC[11]. Plug4Green aims at reducing the power used in data centres while the EASC aims at optimizing the renewable energy usage. Both of them are available on Github.

2.4.1 Plug4Green contributions

Plug4Green is an energy-aware VM placement algorithm and platform that can be easily specialized and extended to fit the data centres specificities. Plug4Green computes the placement of the VMs and state of the servers depending on a large number of constraints, extracted automatically from SLAs. The main contribution is an energetic model based on Constraint Programming, and a list of typical constraints. The main advantage of Plug4Green is its flexibility. It is achieved by allowing the constraints to be formulated independently from each other but also from the power models. This flexibility is validated through the implementation of 23 SLA constraints and 2 objectives to reduce either power consumption or greenhouse gas emission.

2.4.2 EASC contributions

The EASC role is to control one application, so as to make it "energy adaptive", and responsive to external energetical requests. The EASC will plan the activity and control the performance levels of a specific application so as to follow energy budgets. To enable this, the EASC also has to monitor and to predict the energy consumption and activity levels of the application. We contribute the algorithms for two instances of the EASC: EASC for batch processing, and EASC for web services. The batch processing EASC manages applications that have typically a fixed amount of work to do during a certain time, such as generating a certain number of reports or applying a virus scan every day. On the contrary, the web services EASC manages applications that must deliver constantly a service, with various levels of performance. Furthermore, we contribute a Cloud architecture composed of IaaS and PaaS cloud services able to respond to external factors like applications SLAs and renewable energy availability. We present a power prediction model able to predict the power consumption of applications running within a PaaS-IaaS infrastructure.

[10] https://github.com/Plug4Green/Plug4Green
[11] https://github.com/dc4cities/easc

Use less energy Part I

The first two chapters of this thesis address our first objective: reducing the energy consumption in data centres. The first chapter work was performed from 2010 to 2012 as a preliminary study on the heuristics for VM placement. We show that the heuristics selected can be very efficient and fast for the problem of the VM placement. We propose two new algorithms able to allocate VMs and to migrate them in order to lower the data centre energy consumption on the one hand, and to lower the carbon emissions on the other hand. However, this work showed us that one drawback of such algorithms is that they are very specific to a particular use case. The hardware refreshing, the workload characteristics but also the wide variety of SLAs make each data centre unique. Data centres are indeed environments that evolves quickly, in order to accommodate new user requirements. As a result, the original algorithm may not be appropriate anymore, while its *ad-hoc* nature may prevent it from being upgraded according to the new data centre properties.

This work on the heuristics triggered the research on meta-heuristics, presented in the second chapter. We present Plug4Green, an energy-aware VM placement framework that can be easily specialized and extended to fit the new specificities of the data centres. Plug4Green computes the placement of the VMs and state of the servers depending on a large number of constraints, extracted automatically from SLAs. The flexibility of Plug4Green is achieved by allowing the constraints to be formulated independently from each other but also from the power models. The main advantage of heuristic based methods is that they are fast and easy to configure. However, in many situations they cannot lead to the optimal solution if the data centre is heterogeneous. Furthermore they will be hard to extend if new uses cases appear in the data centre. We propose a larger framework that can cope with an arbitrary number of constraints user-defined which ensure the flexibility of the framework and its extensibility regarding new constraints that may come in the future.

3 Heuristics for virtual machine consolidation

This work addresses the problem of high energy consumption and carbon emissions induced by data centres. To address this problem, workload consolidation has been proposed to lower the overall energy consumption. The consolidation of VMs on the most efficient servers allows to switch off the less efficient servers. We consider two allocation scenarios: single allocation (SA) and global optimization (GO) of available resources and propose the corresponding algorithms. The optimization algorithms use the Power Usage Effectiveness (PUE) and Carbon Usage Effectiveness (CUE) in order to evaluate the site efficiency. The evaluation of the algorithms have been performed in a simulation with a federation of data centres with several different configurations. The simulation results shows that the proposed algorithms enable the saving in energy consumption from 10% to 31% and in carbon emission from 10% to 87%.

This chapter is adapted from the papers:

- Dang Minh Quan, Robert Basmadjian, Hermann de Meer, Ricardo Lent, Toktam Mahmoodi, Domenico Sannelli, Federico Mezza, Luigi Telesca, and Corenten Dupont. Energy efficient resource allocation strategy for cloud data centres. In Erol Gelenbe, Ricardo Lent, and Georgia Sakellari, editors, *Computer and Information Sciences II*, pages 133–141. Springer London, 2012

- Dang Minh Quan, Andrey Somov, and Corentin Dupont. Energy usage and carbon emission optimization mechanism for federated data centers. In *Proceedings of the First International Conference on Energy Efficient Data Centers*, E2DC'12, pages 129–140, 2012

3.1 Introduction

Until recently, the key performance indicator of a data centre was its performance. However, the growing number of IT services, resulting in higher power consumption and carbon emission, have forced the ICT community to consider the energy efficiency of data centres carefully. Several energy-aware approaches and resource management techniques have been introduced to tackle power consumption problems in the data centre domain from different points of view.

The objective of this research work is twofold: to reduce both the power consumption and carbon emission of a federation of data centres. To achieve this objective we propose two optimization algorithms for the single allocation of virtual machines and for the global resources optimization.

The chapter is organized as follows: Section 3.2 discusses the related works. Section 3.3 describes the problem formulation. Sections 3.4 and 3.5 present the algorithms for single allocation request and global optimization respectively. The simulation results based on different scenarios and data centres configurations are shown in Section 3.6. Finally, we conclude the chapter in Section 3.7.

3.2 Related works

In order to reduce the energy consumption in data centres, many approaches focus on workload consolidation. The goal is to decrease the number of servers by switching them off or putting them into sleep mode and therefore reduce the power consumption [18, 19]. The biologically-inspired algorithm in [20] determines more power efficient servers within a data centre facility and moves workload on them.

Some research, in contrast, put efforts in minimizing the cooling systems energy consumption with optimized workload [21]. For example [22] uses energy-aware scheduling of workload. [23] tries to find the optimal temperature point of cold air. In [24], the authors study the impact of load placement policies on cooling and maximum data centre temperatures throughout geographically distributed data centres. They propose dynamic load distribution policies that consider all electricity-related costs as well as transient cooling effects.

Several other works use algorithms similar to the First Fit Decreasing algorithm which has been used in previous works [25, 26, 27], with the addition of power-awareness for choosing

the server. In [28], the authors proposed the Modified Best Fit Decreasing, which will allocate a new VM to an active physical machine that would take the minimum increase of power consumption. [29] also proposes algorithms for VM reconfiguration and (re)allocation. A technique for dynamic consolidation of VMs based on adaptive utilization thresholds is proposed in [30]. Those VM utilization thresholds are updated dynamically, which ensures a high level of meeting of the Service Level Agreements. Compared to these works, our work proposes to add explicitly power-awareness in order to choose the appropriate server for VM allocation.

On the other hand, [31] and [32] offers some critical view on VM consolidation techniques. [31] questions how the resource utilization and performance aggregates when VMs are co-hosted, trying to identify bottlenecks that might have adverse impacts on consolidation. Their main insight is that consolidation of cache-sensitive and storage-intensive VMs is likely to lead to severely degraded performance. Yet, they point that heuristics that have simplistic methods of treating multi-dimensional resource requirements are reasonably effective in many common practical situations. [32] analyzed large enterprise workloads with the goal of understanding how effective are the VM consolidation variants in real world. They found out that highly bursty and predictable workloads with high CPU contention can benefit from dynamic consolidation. However, there are many workloads with high memory contention and they recommend semi-static consolidation for such workloads. Semi-static consolidation avoids live migration and associated performance issues making it suitable for critical applications.

3.3 Problem formulation

We assume that we have a set of servers S. Each server $s_i \in S$ is characterised with number of cores, amount of memory and amount of storage ($s_i.nrCore$, $s_i.nrRAM$, $s_i.nrStor$). Each server s_i has a set of running virtual machine V_i including k_i virtual machines. Each virtual machine $v_j \in V_i$ is characterised with required number of virtual CPU, amount of memory, amount of storage ($v_j.rVCpu$, $v_j.rRam$, $v_j.rStor$) and the average CPU usage rate computed in percentage, amount of memory, amount of storage ($v_j.aURate$, $v_j.aRam$, $v_j.aStor$). Because the load in each VM is quite stable , we assume that those values do not change through time.

With each server s_i the constraints 3.1, 3.2, 3.3, 3.4 have to be met. The total usage rate of a certain number of CPUs on a certain number of VMs can not exceed the safe performance

factor for this certain number of cores:

$$\sum_{j=1}^{k_i} v_j.rVCpu * v_j.aURate \leq k * s_i.nrCore \tag{3.1}$$

where k is the safe performance factor, k <1. The average total memory used by the VMs on one server cannot exceed the amount of total available memory of the server:

$$\sum_{j=1} v_j.aRam \leq s_i.nrRam \tag{3.2}$$

The total number of VMs with required number of virtual CPUs is less or equal to number of cores with predefined maximum number of virtual CPUs on board:

$$\sum v_j.rVCpu \leq s_i.nrCore * maxVCpuPCore \tag{3.3}$$

where $maxVCpuPCore$ is maximum number of virtual CPUs per Core. The number of the server's VMs can not exceed the maximum number of VMs, $maxVmPServer$, allowed for the server:

$$k_i \leq maxVmPServer \tag{3.4}$$

When there is a new resource allocation request, the new VM must be allocated to a server in a way that total usage of VMs does not exceed the capacity of the server and the added energy usage is minimum. For the global optimisation request, the VMs must be arranged in a way that total usage of VMs does not exceed the capacity of the server and the energy usage is minimum.

3.4 Single Allocation algorithm

When a new VM is requested by a client, the presented algorithm will check all computing nodes in order to find the suitable node using the least amount of energy. The proposed algorithm will go through each server of the data centre and evaluate its power consumption,

taking into account the impact of the new VM. Based on this evaluation, we select the server having the smallest energy consumption. The algorithm for single allocation request (see Figure 3.1) is presented in Listing 3.1. The power consumption of each server is evaluated by an external component using the power model described in section 4.4.4.

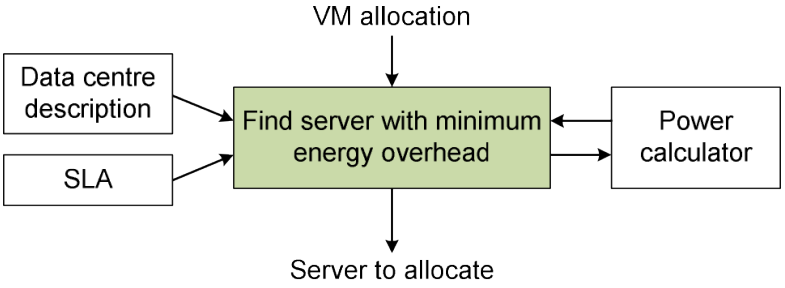


Figure 3.1: Single Allocation diagram

Listing 3.1: Single Allocation algorithm

```
1
2   Input: the model of all servers in the data centres, constraints,
      characteristics of the incoming VM
3   Ouput: the server where to allocate the VM
4   Step 0: Determine all servers meeting the constraints and store them in array A
5   Step 1: If the array A is empty, stop the algorithm and return no solution
6   Step 2: Set index i at the beginning of the array A
7   Step 3: Calculate energy consumption Ei of the data centre if the VM is
      deployed on the server at array index i
8   Step 4: Store (i, Ei) in a list L
9   Step 5: Increase i to the next index of A
10  Step 6: Repeat from Step 3 to Step 5 until i goes out of the scope of A
11  Step 7: Determine min Ei in the list L
12  Step 8: Assign the VM to the server at the corresponding index
13  Step 9: If server at index imin is OFF, put action turn ON to the action list
```

It is noted that Ei is calculated for servers having workload. The server without workload will be shutdown.

3.5 Global Optimisation algorithm

The Global Optimization algorithm has two main phases, as shown in Figure 3.2. In the first phase, we will move the VMs from low load servers to higher load servers if possible in order to empty the low load server. The emptied server can be turned off. The algorithm for phase 1 is given in Listing 3.2.

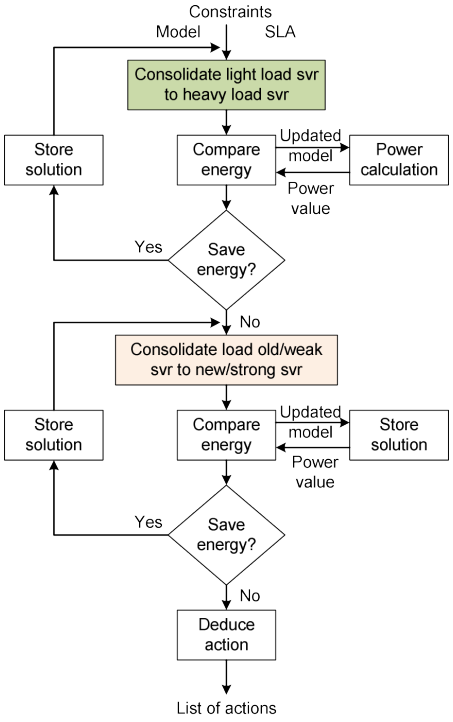


Figure 3.2: Global Optimization diagram

In the second phase, we will move the VMs from the old servers to the modern servers. The emptied server can be turned off as well. As one modern server can handle the workload of many old servers, the energy consumed by the modern server is smaller than the total energy consumed by those many old servers. The algorithm is given in Listing 3.3.

Listing 3.2: Global Optimisation algorithm phase 1

```
1 Step 0: Forming the list LS of running servers
2 Step 1: Find the server having the smallest load rate.The load rate of a server
    is defined as the maximum (CPU load rate, Memory load rate, Storage load rate)
3 Step 2: Sort the VMs in the low load server according to the load level in
    descending order list LW
4 Step 3: Remove the low load server out of running server list
5 Step 4: Use Single Allocation algorithm to find the suitable server in LS for the
     first VM in the list LW
6 Step 5: If Singe Allocation finds a suitable server, update the load of that
    server, remove the VM out of LW
7 Step 6: Repeat from Step 4 to Step 5 until LW is empty of F4G-CS cannot find out
    the suitable server
8 Step 7: If Single Allocation cannot find a suitable server, reset the state of
    found servers and mark Stop=true
9 Step 8: Repeat from Step 1 to Step 7 until Stop=true
```

Listing 3.3: Global Optimisation algorithm phase 2

```
1  Step 0: Sort free servers into a descending order list LFS according to resource
     level.
2  The resource level of a server is defined as the minimum (si.nr_Core/max_Core,si.
     nr_RAM/max_RAM, si.nr_Stor/max_Stor) with max_Core, max_RAM, max_Stor are the
     maximum number of Cores, memories, storages of the server in the pool.
3  Step 1: Sort running servers into an ascending order list LRW according to load
     rate.
4  The load rate of a server is defined as the maximum(CPU load rate, Memory load
     rate, Storage load rate)
5  Step 2: Set m_count=0
6  Step 3: Remove the first server s out of LFS
7  Step 4: Remove the first server w out of LRW
8  Step 5: If we can move workload from w to s m_count+=1
9  Step 6: Repeat from Step 4 to Step 5 until we cannot move workload from w to s
10 Step 7: If m_count <= 1, reset the state of moved w,s and mark Stop=true
11 Step 8: Repeat from Step 2 to Step 7 until Stop=true
```

3.6 Evaluation

In this section, we study in simulation the saving rates of the resource allocation mechanisms presented.

3.6.1 Simulation scenario

We created different resource configuration scenarios in order to evaluate our algorithms in different conditions. We use 4 server classes with single core, dual cores, quad cores and six cores, respectively. The main parameter for each server class are presented in Table 3.1. Furthermore, the algorithms are evaluated in various settings with a federation of data centres.

Table 3.1: Server configuration

Server type (i)	**Nr. Cores**	**Pidle CPU (W)**	**CPU freq. (GHz)**	**RAM (GB)**	**Disk (MB)**	**Pmax (W)**
1	1	7.57	2.0	1	400	102.22
2	2	9.88	2.0	2	500	103.38
3	4	20.14	2.2	4	800	171.70
4	6	22	2.4	6	1000	229

We generated 3 different kinds of data centres: modern data centre, normal data centre and old data centre as shown in Table 3.2. In the normal data centre the percentage of different server classes is balanced. In the old data centre, the percentage of server classes with less cores is predominant. In the modern data centre, most servers have high performance servers.

Table 3.2: DC resource scenarios

Scenario	Nr. servers type 1	Nr. servers type 2	Nr. servers type 3	Nr. servers type 4
1 - Modern DC	50	100	150	200
2 - Medium DC	100	100	100	100
3 - Old DC	200	150	100	50

With each resource configuration, we generated a raw set of jobs. Each simulation is divided in 1000 timeslots, each timeslot simulating 5 minutes of real time. The jobs are submitted randomly to the system during the simulation run. The parameters for each job are determined by random selection. The duration of each job can be for 5 minutes to 500 minutes. We generated 3 kinds of data centres federations: federation with many old data centres, federation with balanced types of data centre, federation with many modern data centres. The detail of each federated data centres configuration is presented in Table 3.3.

Table 3.3: Federation resource scenarios

Federated ID	Nr. old DC	Nr. normal DC	Nr. modern DC
1 - Old DCs	6	3	1
2 - Balanced DCs	3	4	3
3 - Modern DCs	1	3	6

We use 3 PUE/CUE configurations as presented in Table 3.4.

Table 3.4: PUE/CUE configurations

Type	Energy source	PUE	ESC	CUE
Low	Oil 20%, Hydro 40%, Nuclear 40%	1.3	0.13	0.16
Average	Coal 50%, Nuclear 30%, Hydro 20%	1.5	0.46	0.67
High	Coal 80%, Oil 20%	1.8	0.85	1.53

To assign PUE/CUE to each data centres, we use 3 configurations as presented below:

- Energy mix 1: Old data centre high PUE/CUE, average data centre normal PUE/CUE, modern data centre low PUE/CUE
- Energy mix 2: Old data centre normal PUE/CUE, average data centre normal PUE/CUE, modern data centre normal PUE/CUE
- Energy mix 3: Old data centre low PUE/CUE, average data centre normal PUE/CUE, modern data centre high PUE/CUE

We run the simulation for each federated data centres configuration using each of the energy

mix. Several scenarios are run: first we perform the allocation of VMs only within its own preselected data centre. We then run the simulation again, allowing VMs to be allocated within the full federation of data centres. Finally we run again the simulation allocating VMs in the federation, this time running also the global optimization algorithm every 5 timeslot. For each scenario, we calculate the energy consumption and CO_2 emissions for the 1000 timeslots.

3.6.2 Numerical results

The simulation results in terms of energy consumption (in MW*timeslot) are presented in Table 3.5.

Table 3.5: Simulation results for energy consumption

Federated ID-energy mix ID	SA Energy	SA federated		SA + GO federated		
		Energy	Saving	Energy	Transfer Energy	Saving
1-1: Old Fed., mostly high PUE	385.79	277.05	28%	265.66	0.003442	31%
1-2: Old Fed., average PUE	354.12	329.68	7%	313.75	0.003736	11%
1-3: Old Fed., mostly low PUE	338.59	289.33	15%	275.35	0.003646	19%
2-1: Aver. Fed., mostly high PUE	395.18	314.60	2%	301.75	0.049073	24%
2-2: Aver. Fed., average PUE	396.13	369.25	7%	354.16	0.033214	11%
2-3: Aver. Fed., mostly low PUE	413.55	326.03	21%	312.71	0.004585	24%
3-1: Mod. Fed., mostly high PUE	412.09	357.49	13%	343.61	0.068402	17%
3-2: Mod. Fed., average PUE	445.99	418.37	6%	402.13	0.003558	10%
3-3: Mod. Fed., mostly low PUE	502.17	367.96	27%	353.68	0.004252	30%

The simulation result presented in figure 3.3 shows the efficiency of the energy aware algorithms for the federated data centres. Depending on the configuration, the energy saving compared to the single allocation case spans from 10% to 31%. We can see that the federated optimization algorithm is most effective when the values of PUE of each data centre in the federation greatly differ from each other. A federation gathering data centres with different PUE will allow for more opportunities for allocating VMs in the most effective place. On the opposite, with the simulation configurations 1-2, 2-2 and 3-2, where the PUE values are the same for each data centre, the saving rate is much smaller. Furthermore, we can see that the global optimization, performed regularly, offers an additional 3.6% of energy saving. It allows to re-consolidate the VMs when the DC situation is changing (i.e. when old VMs are terminating).

The simulation result in terms of CO_2 emission (in Ton*timeslot/h) is presented in Table 3.6

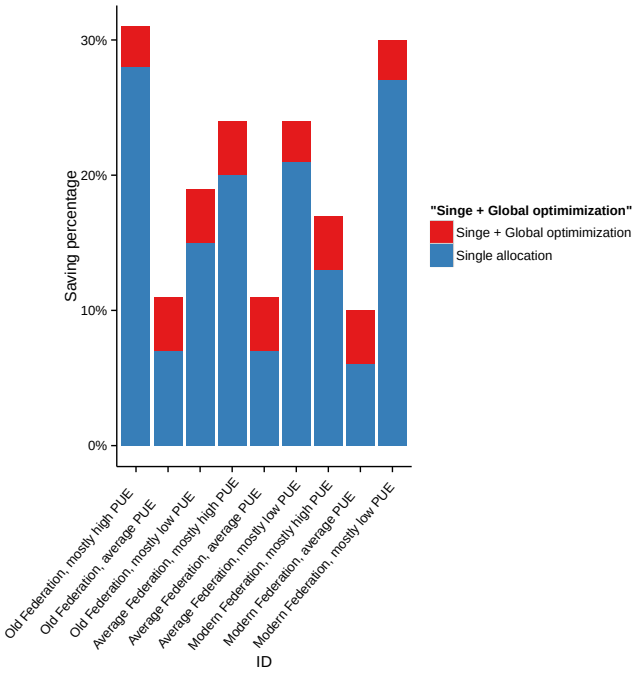


Figure 3.3: Energy savings for a federated data centre

and displayed in Figure 3.4.

Table 3.6: Simulation results for CO_2

Federated ID-energy mix ID	SA CO_2	SA federated		SA + GO federated		
		CO_2	Saving	CO_2	Transfer CO_2	Saving
1-1: Old Fed., mostly high PUE	252.81	34.36	86%	32.95	0.00279	87%
1-2: Old Fed., average PUE	162.83	151.59	7%	144.27	0.005537	11%
1-3: Old Fed., mostly low PUE	124.12	40.41	67%	38.46	0.008822	69%
2-1: Aver. Fed., mostly high PUE	183.24	37.32	8%	35.80	0.005822	80%
2-2: Aver. Fed., average PUE	182.15	169.79	7%	162.85	0.003441	11%
2-3: Aver. Fed., mostly low PUE	233.33	45.11	81%	43.27	0.00734	81%
3-1: Mod. Fed., mostly high PUE	117.63	41.04	65%	39.44	0.007853	66%
3-2: Mod. Fed., average PUE	205.08	192.38	6%	184.91	0.00496	10%
3-3: Mod. Fed., mostly low PUE	363.22	49.27	86%	47.36	0.006843	87%

The saving rate spans from 10% to 87% in CO_2 emissions. The saving in term of CO_2 is potentially very high. This is because the CUE differences between two data centres can be huge, depending on the local energy source mix (see Table 3.4. Similar to the energy consumption case, we can see that the federated optimization algorithm is most effective when the values of CUE of each data centre in the federation greatly differ from each other. The global optimization, performed regularly, offers an additional 1.9% of emissions saving.

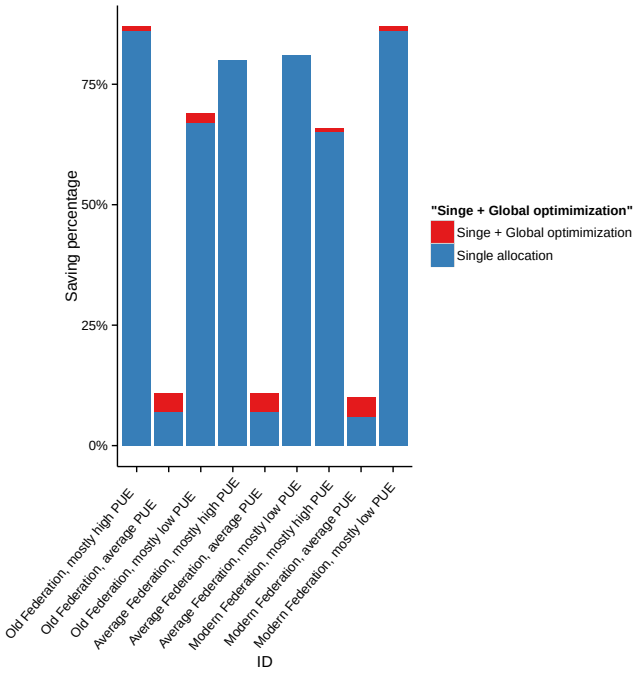


Figure 3.4: CO_2 savings for a federated data centre

3.6.3 Performance comparison

In this section we evaluate our algorithm Single Allocation (SA) and Global Optimization (GO) in comparison with other algorithms. The legacy algorithms round robin, load balance and greedy have been selected for this study. We perform again the scenario presented in the previous section, within the modern, balanced and old DC. We evaluated the energy consumption of each data centre for each scenario, during the full simulation. The Table 3.7 presents the average power consumed over each scenario trial.

Table 3.7: Performance comparison simulation result

	Modern DC	Normal DC	Old DC
Round robin (KW)	61.23	48.45	41.89
Round robin + GO (KW)	47.64	40.83	37.21
Load balance (KW)	60.46	47.87	41.15
Load balance + GO (KW)	46.56	39.67	36.85
Greedy (KW)	50.74	40.35	35.78
Greedy + GO (KW)	45.67	37.78	33.47
SA (KW)	46.58	36.13	32.18
SA + GO (KW)	44.76	34.47	30.7

Our energy aware VM allocation algorithm performed better than the Greedy algorithm, which is the most energy efficient between the legacy algorithms. On average, it consumed 9.44% less.

This improvement is mainly due to the selection of the most energy efficient server performed by our algorithm before allocating the VMs. It consumed 24.20% less energy compared to a non energy efficient legacy algorithm such as round robin, which is still widely used in the data centres. Adding the global optimization algorithm on top of each legacy algorithm allows to save an additional amount of energy: 11.73% on average. This additional energy saving is mainly to the consolidation affect applied by GO in the data centre when VMs are finishing. This shows that existing frameworks using legacy VM allocation algorithm can still be optimized using the global optimization in order to consolidate the VMs.

3.7 Conclusion

This chapter presented a method that potentially reduces the energy consumption of the IaaS data centre. To save energy, we allocate the resources on the most efficient servers. In the global optimization algorithm, we take advantage of the fact that new generation computer components have higher performance and consume less energy than the old generation. Thus, we move the heavy load applications to the new servers with larger number of cores while moving light load applications to the old servers with smaller number of cores, and finally switch off as many old servers as possible. The evaluation shows that our algorithms can enhance the performance further when the data centre consists of a larger number of old servers, and also in the case when many old servers are working with heavy load rate and many modern servers are working with light load rate.

4 Plug4Green, an energy-aware VM placement framework

To maintain an energy footprint as low as possible, data centres manage their VMs according to conventional and established rules. Each data centre is however made unique due to its hardware and workload specificities. This prevents the ad-hoc design of current VM managers from taking these particularities into account to provide additional energy savings. In this chapter, we present Plug4Green, an energy-aware VM placement algorithm that can be easily specialized and extended to fit the specificities of the data centres. Plug4Green computes the placement of the VMs and state of the servers depending on a large number of constraints, extracted automatically from SLAs. The flexibility of Plug4Green is achieved by allowing the constraints to be formulated independently from each other but also from the power models. This flexibility is validated through the implementation of 23 SLA constraints and 2 objectives aiming at reducing either the power consumption or the greenhouse gas emissions. On a heterogeneous test bed, Plug4Green specialization to fit the hardware and the workload specificities allowed to reduce the energy consumption and the gas emission by up to 33% and 34%, respectively. Finally, simulations showed that Plug4Green is capable of computing an improved placement for 7,500 VMs running on 1,500 servers within a minute.

This chapter is derived from the papers:

- Corentin Dupont, Fabien Hermenier, Thomas Schulze, Robert Basmadjian, Andrey Somov, and Giovanni Giuliani. Plug4green: A flexible energy-aware vm manager to fit data centre particularities. *Ad Hoc Networks*, pages 505–519, 2014
- Corentin Dupont, Thomas Schulze, Giovanni Giuliani, Andrey Somov, and Fabien Hermenier. An energy aware framework for virtual machine placement in cloud federated data centres. In *Proceedings of the 3rd International Conference on Future Energy Systems:*

Where Energy, Computing and Communication Meet, e-Energy '12, pages 4:1–4:10. ACM, 2012

4.1 Introduction

Cloud data centres provide powerful ICT facilities to host a large spectrum of applications. Originally, data centre operation management has been focused on improving metrics like performance, reliability, and service availability. Furthermore, due to the rise of service demands, data centres have to constantly evolve in size and complexity. This and the continuous increase of energy cost have prompted the ICT community to add energy efficiency as a new key metric for improving data centres facilities.

VM consolidation is the norm to improve energy efficiency. In practice, an *ad-hoc* VM placement algorithm considers the data centre and the workload properties to allocate the VMs among the servers according to energy objectives [33, 34, 27, 35, 28]. However, the hardware refreshing, the workload characteristics but also the wide variety of SLAs make each data centre unique. As a result, the original algorithm may not be appropriate anymore, while its *ad-hoc* nature may prevent it from being upgraded according to the new data centre properties.

These evolutions are calling for a VM allocation approach that is flexible enough to address data centres complex and changing nature. In other words, a VM placement algorithm has to take into account a large spectrum of tuning possibilities and constraints associated with data centre specificities. As an example, an energy-aware algorithm should be able to provide additional energy savings by being fine-tuned according to multiple particular hardware and SLAs. We define the following requirements on flexibility that have to be met by a VM placement algorithm: i) be extensible with users and operator's new constraints and requirements, especially in the case of new SLA definitions associated with new services. ii) be adaptable to any data centre and its particularities, and be adaptable to new hardware installed.

To satisfy the requirements, we propose to use the Constraint Programming (CP) [36] paradigm. This paradigm and its associated algorithms have already been applied to address common users requirements such as performance and fault tolerance [29, 37]. However, the lack of energy models prevented it to explicitly address the energy related concerns which become of vital importance in upgraded and consolidated data centres with improved capacity.

In this chapter we present Plug4Green, an energy aware VM manager based on CP, with a

special focus on extensibility. We show how the flexibility realized in our framework can address new requirements arriving in a data centre. Furthermore, we show that an increased flexibility, by allowing to fine-tune the algorithms, allows better energy savings. We validate our approach by implementing a use case on energy efficient VM management in data centres while meeting the requirements on performance. Our main contribution, in terms of its practical value, is threefold:

- *Flexibility:* We propose and implement 23 VM placement constraints to address common concerns such as hardware compatibilities, performance, security issue, and workload instability. We also propose 2 objectives: the first one reduces the overall energy consumption while the second one reduces the greenhouse gas emission. The usage of CP makes placement constraints, objectives, and algorithms independent from each other, which is crucial for extensibility: new concerns can be added in the VM manager without changing the existing implementation.
- *Efficiency:* We show that using our framework in a realistic cloud data centre environment allows to reduce the overall energy consumption up to 33% and the gas emission up to 34%. These savings are achieved by considering the servers' hardware heterogeneity, their different energy-efficiency and different compositions of SLAs.
- *Scalability:* We show by simulation how such an approach can be scalable. In particular, we were able to compute the improved placement of 7,500 VMs on 1,500 servers, while respecting their SLA.

This chapter is organized as follows: Section 4.2 introduces related works Section 4.3 presents the design of Plug4Green. Its implementation is discussed in Section 4.4. Section 4.5 evaluates its practical benefits, and Section 4.6 concludes the chapter.

4.2 Related Work

This section discusses recent advances in the area of energy-aware frameworks for data centres. We survey the literature related to existing flexible and extensible frameworks, and the power consumption prediction models.

4.2.1 Extensible and flexible frameworks

A few flexible and extensible frameworks for VM allocation have been proposed recently. For example, BtrPlace [37] is a CP-based flexible consolidation manager. As detailed in Sec-

tion 4.4.4, Plug4Green leverages on Btrplace [29, 38]. BtrPlace does not take into consideration energy related problems and does not provide an operator with the opportunity of setting optimization objectives. In contrast to BtrPlace, Plug4Green directly addresses energy consumption problem. In this work, Plug4Green proved the practical benefits of flexibility to address energy related problems. This required numerous extensions: the development of a power model and different model extensions, two objectives with their associated heuristics, 7 energy-related constraints, and a domain-specific language to directly exhibit energy concerns and metrics such as PUE, CUE and Watts, to the end-users.

Similar modular consolidation manager adopting CP paradigm is presented in [39]. The authors ensure high availability for VM placement by guaranteeing at any time a certain number of vacant servers to allocate VMs with regards to placement constraints. The authors ensure high availability for VM placement by guaranteeing at any time a certain number of vacant servers to allocate VMs with regards to placement constraints. The scalability is demonstrated with 32 servers and 128 VMs only.

In [40], the authors propose an hybrid approach based on a Business Rules Management System (BRMS) and CP to manage VMs. The BRMS monitors and analyses the servers' state at a period of time to detect overloaded servers and bottlenecks. Once a problem is identified the BRMS models its instance and sends it to the CP solver. A user can express constraints through the BRMS but the resulting specialization cannot be deterministic contrary to Plug4Green. In contrast to our manager, both the systems presented in [39] and [40] are not addressing energy-efficiency problems.

Some preliminary theoretical and practical aspects of Plug4Green were investigated in [10]. Energy-aware VM allocation was the primary goal while this work focuses on flexibility. For this purpose, we created seven new SLA constraints, notably energy-oriented, and a new power objective model has been included. Three new heuristics has been developed, allowing finding good solutions quickly. A complete experimentation has been carried out with new prototype, evaluating the impact of several popular SLA constraints on the energy saving. In this work, we demonstrate an energy saving of 33% while it was 18% in federated cloud data centre experiment in [10], due to new energy-aware constraints and heuristics. The scalability of the framework has been also greatly improved. Plug4Green is about 30 to 40 times faster which makes it capable of managing larger data-centres.

Nefeli [41] is a cloud gateway that places VMs with regard to user preferences called "hints". Nefeli expects that the users are aware of the role each VM plays in the infrastructure and

communicate this information to the cloud as a hint. The VM placement is computed using simulated annealing. A hint is then implemented as a scoring function that evaluates the quality of the placement with respect to its concern. This approach makes Nefeli flexible: Nefely can be extended by programming new hints. As a difference with Plug4Green, the approach does not separate the model from its resolution method. The specialization made by the hints is also not composable as each score is, by nature, relative to the others. Despite the authors discuss some energy-related hints, their system as a whole does not make a special emphasize on energy efficiency. Finally, Nefely has not been evaluated in terms of scalability.

4.2.2 Server power models

In [42], the authors propose a model to predict the average power consumption of a server regardless of its utilisation. The two main benefits are the followings: (i) it is simple to compute and no dynamic information is required, and (ii) it is similar to the method of estimating a system's power consumption based on the manufacturer's specifications. However, it provides very rough predictions especially for heterogeneous software and hardware environments. In [35], a linear model estimates the power consumption according to the server's CPU utilisation. This approach is not suitable for not CPU-intensive workloads. The model in [43] follows a similar approach while taking into account the utilisation of the hard disk. Authors in [44] extend the CPU and disk utilisation model by looking at performance counters of the system such as the amount of instruction-level parallelism, the activity of the cache, or the utilisation of the floating-point unit. However, performance counters are accessed differently on each processor type. As a matter of fact, this model is not usable across heterogeneous systems. In contrast to the above-mentioned models, which provide one linear model for the whole server, our approach aggregates different models for different components based on their behaviour. In addition, our approach does not need any calibration phase. Consequently, our models are suitable not only for homogeneous, but also for heterogeneous environments like cloud data centres.

4.3 Design

Plug4Green is extensible. The architecture (Section 4.3.1) allows to extend the engine by adding new concerns, without modifying the underlying algorithms. In particular, new constraints (Section 4.3.2) can be added easily.

4.3.1 Architecture

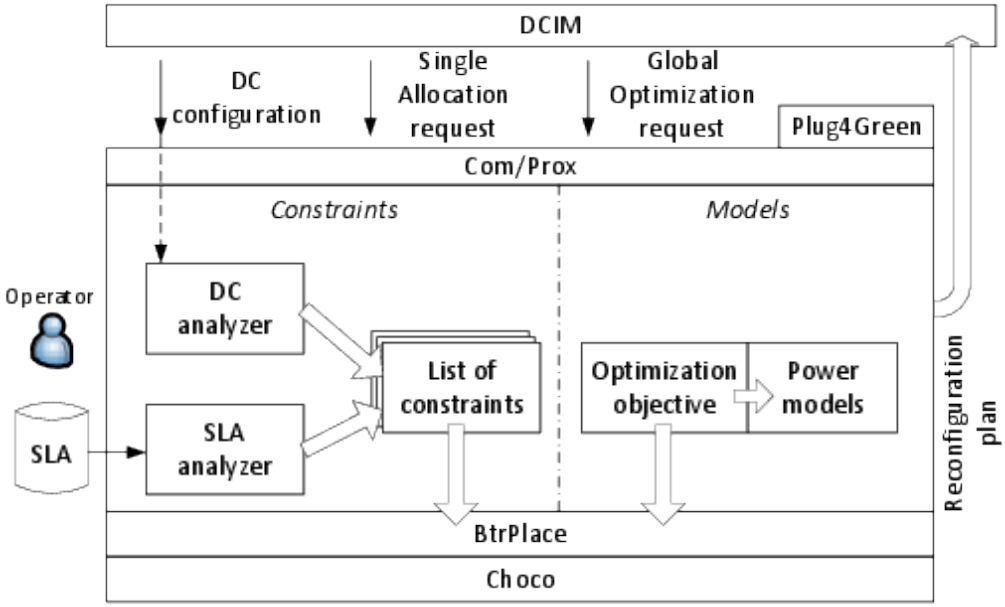


Figure 4.1: Plug4Green architecture

Figure 4.1 depicts the architecture of Plug4Green. Plug4Green considers a set of *SLA* constraints along with the *data centre configuration* to compute a *reconfiguration plan* as an output. The *data centre configuration* captures all the relevant ICT resources of a data centre with their energy-related attributes and interconnections in an XML format. The reconfiguration plan is a set of actions (powering on or off a server, migrating a VM, ...) that satisfies all the constraints and minimizes the current objective. The objective can be to minimize either the power consumption of a federation of data centres, or the CO_2 emissions. The diagram shows the clear separation between the *Constraints* part ("what" we want to do) and the *Models* part ("how" to solve the problem), which is fundamental for extensibility.

Plug4Green is called by the Data Centre Infrastructure Management (DCIM) for two different events: *Single Allocation* or *Global Optimisation*. The *Single Allocation* event is triggered when a new VM have to be allocated. Plug4Green will compute and return the best server to allocate the VM on, taking into account the characteristics of the VM, the current state of the data centre, the SLAs and the current objective. The *Global Optimisation* event is itself triggered regularly and Plug4Green will return a reconfiguration plan. In manual mode, the data centre operator validate or decline this reconfiguration plan, while in automatic mode, it is enacted automatically. Plug4Green will then execute the plan to reduce the overall data centre power consumption or gas emission while also respecting the SLAs. The *Com/Prox* layer ensures

that Plug4Green can be plugged to different existing DCIM. Its the only part that must be updated to connect to a new DCIM. Currently, Plug4Green can be integrated into VMWare[1], Eucalyptus[2], and HP Matrix Operating Environment[3] infrastructures.

4.3.2 Constraints

Numerous SLAs exists in a data centre. Furthermore those are more and more extended with energy concerns Our framework provides a language to express SLAs based on CP, that also takes into account energy constraints. To show the flexibility of our approach, we prepared an extensive number of SLA and energy constraints using this language, as showed in Table 4.1.

SLAs are usually provided as part of an English-written contract between a client and an IT service provider. Upon receiving this contract, the Capacity Planning Team (CPT) of a data centre have to translate it into our SLA schema. The SLA schema is a format allowing the CPT to use the pre-defined constraints detailed in Table 4.1. Once the SLA file is ready, it can be submitted to Plug4Green. The SLA constraints will then be translated automatically to lower level CP constraints and processed by a CP engine, with the process shown in Figure 4.2.

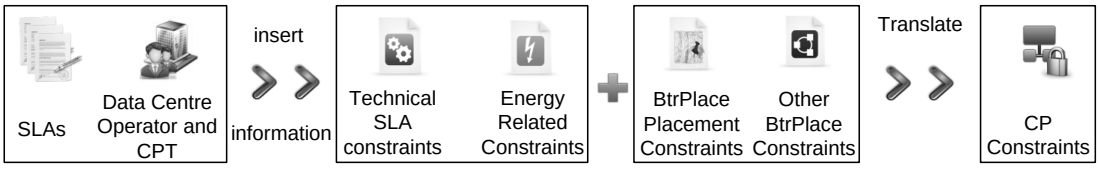


Figure 4.2: Translation of the SLA contract into technical SLAs and then to constraints

Depending on the topology of the data centre, a different SLA contract can be applied to different groups of servers in the data centre: in this way it is possible to have several SLA contracts active within the same data centre.

[1] http://www.vmware.com
[2] http://eucalyptus.com
[3] http://h18004.www1.hp.com/products/solutions/insightdynamics/overview.html

Cat	Constraint	Restriction
Hardware	HDDCapacity	minimum amount of hard disk space available for a VM
	CPUCores	minimum number of CPU cores available for a VM
	CPUFreq	minimum CPU frequency available for a VM
	MemorySpace	minimum amount of memory space available for a VM
	GPUCores	minimum number of GPU cores available for a VM
	GPUFrequency	minimum GPU frequency available for a VM
	RAIDLevel	minimum Raid level available for a VM
QoS	MaxVMperServer	maximum number of VMs per server
	MaxCPULoad	maximum load of CPUs for a server
	MaxVLoadperCore	maximum virtual load associated to a CPU core
	MaxVCPUperCore	maximum number of virtual CPU associated to a CPU core
	MaxVRAMperPhyRAM	maximum amount of virtual RAM per physical RAM
	MaxServerAvgVCPUperCore	Same as MaxVCPUperCore but averaged for all cores of a server (not Core per Core)
	MaxServerAvgVRAMperPRAM	Same as MaxVRAMperPRAM but on a server basis
	Bandwidth	minimum network bandwidth available for a VM
Security	DedicatedServer	a VM will be hosted on a server with no other VMs
	Access	a certain secure access possibility for a VM (e.g. VPN)
Energy	MaxServerPower	maximum power consumption for a server
	DelayBetweenVMMigrations	minimum delay between two successive VM migrations
	DelayBetweenServerOnOffs	minimum delay between two state changes for a server
	VMPaybackTime	allow a VM migration only if the energy spent for the migration is 'paid back' within the given time interval.
	SpareNodes	minimum amount of servers that are kept free (spare capacity) in the data centre
	SpareCPUs	minimum amount of CPUs that are kept free in the data centre

Table 4.1: SLA constraints

4.4 Implementation

In this section we provide details on the model that allowed us to easily build the constraints presented earlier. We then present the power objective model and the heuristics we used to increase the scalability of our framework and the quality of the computed configurations.

4.4.1 The constraint programming family

In practive, Plug4Green extends the flexible consolidation manager BtrPlace [37]. The flexibility of BtrPlace (and consequently of Plug4Green) comes from the usage of CP [36]. CP allows modelling and solving combinatorial problems where the problem is modelled by stating constraints (logical relations) that must be satisfied by its solution. To use CP, a problem is modelled as a Constraint Satisfaction Problem (CSP), comprising a set of variables, a set of domains representing the set of possible values for each variable and a set of constraints that represent the required relations between the values of the variables. A solver computes a solution for a CSP by assigning each variable to a value that simultaneously satisfies all the constraints. A CSP can be augmented to a Constraint Optimisation Problem (COP) by stating an objective that requires to minimize or maximize the value of a given variable. The algorithm used to solve a CSP or a COP is independent of the constraints composing the problem and the order in which they are provided. When no timeout is specified, the CP solver computes and returns the solution that lead to the best solutions according to the objective and the constraints. Otherwise, it returns the best solution computed so far that still satisfies the constraints. By using CP, we achieve the important goal of separating two concerns: the development of a placement objective and the development of constraints that specialise the objective.

As an alternative to CP, we explored briefly Satisfiability Modulo Theories (SMT) [45]. A SMT problem is to determine the satisfiability of ground logical formulas with respect to background theories expressed in classical first-order logic with equality. Modern SMT solvers integrate a Boolean satisfiability (SAT) solver with specialized solvers for a set of literals belonging to each theory. The problem consists in finding an assignment to the variables that satisfy all constraints. We also surveyed the feasibility of using purely functional languages such as Haskell[4] as the base language for the constraint engine of Plug4Green. Programs in Haskell tend to be much less verbose than in Java (in the order of ten time less lines). It is also a declarative language, like Constraint Programming is, so the expression of constraints is

[4] https://www.haskell.org/

more clear and natural. Furthermore, Haskell is pure, which combined with its strong type system allows to reduce drastically the number of bugs. An implementation of the classical VM packing problem is shown in annexA.

4.4.2 The Plug4Green model

In practice, BtrPlace embeds the CP solver Choco[5] and provides a core CSP, called the VM Repacking Scheduling Problem (VRSP) that only models the memory and CPU demands of the VMs, the server's state and the future VM placement. By default, BtrPlace *does not* provide variables, constraints and objectives that are related to energy concerns. These variables are then provided by Plug4Green, as an extension of the VRSP. We use the VRSP as a pivot model to solve energy-efficient VM placement and server management problems.

Energy related variables	
P	Future global power consumption of the data centre federation
$P(s)$	Future power consumption of a server s
E_{reconf}	Energy spent by the reconfiguration plan
$E_{move}(v)$	Energy spent for the migration of VM v
$E_{onoff}(s)$	Energy spent for switching on or off a server s
Variables used from BtrPlace model	
hosters	Association array VM/Server of the resulting configuration
$card(s)$	Number of VMs that a server s will host
$n^{CPU}(s)$	Future CPU load of a server s
$n^{RAM}(s)$	Future RAM usage of a server s
$n^{HDD}(s)$	Future HDD usage of a server s

Table 4.2: Model variables

Table 4.2 is summarizing the energy variables that we created, and the variables that were reused from BtrPlace. As an example, Listing 4.1 presents the definition of the constraint *MaxServerPower* is term of these variables:

Listing 4.1: Definition of constraint *MaxServerPower*

```
1 public void injectMaxServerPower(VRSP model, int maxServerPower) {
2     model.post(eq($P$, plus(mult(card, $\beta$), $\alpha$)));
3     model.post(leq($P$, maxServerPower));
4 }
```

As a reminder, $maxServerPower$ ensures a maximum power consumption for a server. Using the CP operators eq, $plus$ and $mult$ we first define the power of a server P (one of the variables

[5]Choco: http://www.emn.fr/z-info/choco-solver/

in Table 4.2). α and β are defined in the Section 4.4.4. We then create the constraint that this power P must be less or equal than a threshold $maxServerPower$. This constraint is then injected into the *model*. The constraint definition is declarative. It does not state *how* we want to solve it but only *what* we desire. This is a clear separation of concerns: we were not obliged to revise the solving algorithm to define this constraint.

4.4.3 From SLA to constraints

As can be seen in [46], SLAs commonly contain metrics related to the hardware infrastructure the customer is guaranteed to be provided with. They define for example a certain amount of memory, hard disk space, CPU frequency or a certain RAID level. In addition to this first category, the technical SLA contains also QoS-related constraints. Within the third category we capture constraints concerning security issues, such as secure access possibilities (e.g. VPN) or the guarantee that one VM will only be hosted on one server. The fourth category include energy related constraints: constraint on the energy consumption for servers, the time between VM movements, and the amount of free resources that must be kept available in the data centre.

In order to be included in Plug4Green the constraints need to be mapped into rules understandable by the CP engine. The constraint can be either implemented over pre-existing constraints in BtrPlace or using the low-level constraints provided by the Choco library. For instance, the constraints related to hardware metrics are usually implemented using the *Fence* constraint provided by BtrPlace to restrict the VM placement to a given group of servers. In a pre-selection process, a set of servers having a satisfying hardware is computed. A *Fence* constraint is then instantiated with this set, allowing an allocation to be performed only on this set of servers. In practice, 17 of the 23 constraints bundled currently inside Plug4Green were developed without relying on pre-existing constraints in BtrPlace.

The output of Plug4Green is also dependent on energy-related constraints. The simplest of these constraints is *MaxServerPower*: it allows the data centre operator to specify the maximum power that a server or a group of servers can consume. This constraint takes into account the fact that Plug4Green does not work with perfect information. Indeed, despite Plug4Green aims explicitly at reducing the energy of the overall federation of data centres, it may not be aware, for example, that the cooling system of a specific group of servers is not efficient, or that its power feed is weak. This information may not be included in the power model of Plug4Green, showing the usefulness of the constraint. In practice, to satisfy this constraint Plug4Green will limit the number of VMs hosted by the servers, to keep the overall

power under the threshold.

The energetic and performance costs of a VM movement itself are also considered. Plug4Green provides the data centre operator three ways of acknowledging those costs. The first and preselected possibility is to take the VM life-time into account, if available. If we know in advance the remaining time that a VM will be online before being shut down, it's easy to compute whereas it's worthwhile to migrate it. We simply compare the energy saved by the VM if we move it to the energy cost of the move, as shown in Section 4.4.4. However, especially in Cloud computing, the remaining life-time of VM may not be available. The *DelayBetweenMove* provides a second opportunity to control the migrations by stating a minimal delay between migrations of given VMs. This simple solution can be used whenever no information about the remaining life-time is available. Finally, a more advanced version considers the migration as an investment that must be worthy. This management opportunity is granted through the *VMPaybackTime*. For this constraint, the data centre operator defines a mean life time for VMs, depending on the data centre typology. This time will be used to compute if it's worthwhile to move a VM. For example, the operator can define that the average VM life time in the overall data centre is one hour. Using the same algorithm as described above, we can then compute if it's worthwhile migrating the VM.

In addition to the acknowledgement of energy consumed for the movement of VMs, a data centre operator needs to deal with the problem of rapid fluctuations of workload. One solution is to ensure a specific amount of resources is always available to absorb the variations. We implemented the constraints *SpareNodes* and *SpareCPUs* to allow the operator to define the associated values as a function of time. In this way the best trade-off between reliability and energy efficiency is achieved. If, for instance, the historical load pattern of the data centre shows that during night time there is only a low variation of the number of VMs but in the time between eight and nine rapid rise of number of VMs occurs (e.g. due to the start of the working day), the *SpareNodes* or *SpareCPUs* parameter values are kept small during night-time and is increased before office hours.

4.4.4 Optimisation objectives

In this section, we first introduce the power prediction models that estimate the power consumption of a server. We then present two objectives implemented as a proof of concept to optimise a data centre usage: $minEnergy$ to reduce the energy consumption, and $minGasEmission$ to reduce the CO_2 emission.

Power consumption prediction

In order to derive the optimisation objectives mentioned, we need to design appropriate power consumption prediction models. In this section, we provide models for idle and dynamic power consumption estimations. We introduce the corresponding models for servers by breaking down into their constituent components (e.g., processor, memory, hard disk, see Figure 4.3).

When a *Single Allocation* or a *Global Optimisation* is triggered, the framework collects all the necessary information from the data centre management framework, in particular the *dynamic* parameters. This information is passed to Plug4Green using the schema described in [47]. The prediction models described in this section are then used by Plug4Green to build an objective function, adapted for the current configuration and state of the data centre. The built objective function (either *minEnergy* or *minGasEmission*) is thus recomputed and may be different for each call of Plug4Green.

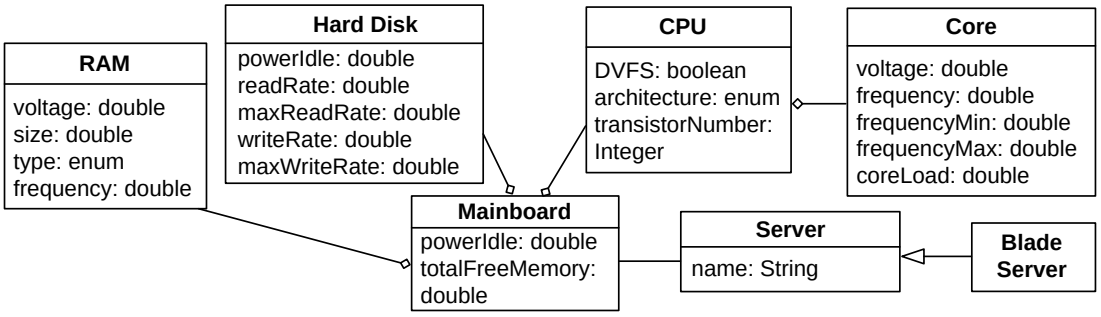


Figure 4.3: Server UML class diagram

Idle power A server is in idle state when all of its constituent components are inactive but still powered. To this end, the major contributors to the idle power consumption of a blade server are the processor, memory, and hard disk. The idle power consumption of a *multi-core* processor i is given by equation (4.1) where t denotes the total number of cores, v_s the total number of transistors of the s^{th} core, I_{us} and V_{us} the current and the voltage of the u^{th} transistor of the s^{th} core, respectively.

$$P_{CPU}(i) = \sum_{s=1}^{t} \sum_{u=1}^{v_s} I_{us} \times V_{us} \tag{4.1}$$

In [48], an approach is presented to model the current I_{us} in terms of the voltage V_{us}, by taking

into account the processor frequency. Hence, the most relevant energy-related parameters are the architecture (e.g., Intel, AMD), the number of transistors (in the order of millions), voltage as well as the corresponding C-states[6] with which the processor is operating. Concerning the *memory* modules, they consume power during the idle state while refreshing the memory ranks holding stored data. The idle power consumption of a memory module j is given by:

$$P_{RAM}(j) = \sum_{v=1}^{r} s_v \times f_v \times c \times V_v^2 \tag{4.2}$$

where s_v denotes the size (GB), f_v indicates the frequency (MHz), and V_v represents the voltage of the v^{th} memory module, whereas c is a constant. In [48], values for c where derived based on the type (e.g., DDR2, DDR3) of the memory modules. Consequently, the most relevant energy-related attributes are the size, frequency, voltage, and type of the memory modules. All the above-mentioned attributes are static ones and can be found within the manufacturers' data sheets. A *hard disk* is in idle state when neither read nor write operations take place. Moreover, during the idle state, most of the mechanical parts of the disk are stopped. Consequently, manufacturers provide in their data sheets the average idle power consumption for hard disks which can be used as a best estimate.

Given a blade server s composed of processors, memory modules and hard disks, its idle power consumption is estimated by the equation (4.3). l, m, and n denote respectively the total number of processors, memory modules as well as hard disk drives. c represents the power consumption of the mainboard which can be estimated from observation data[7].

$$P_{idle}(s) = \sum_{i=1}^{l} P_{CPU}(i) + \sum_{j=1}^{m} P_{RAM}(j) + \sum_{k=1}^{n} P_{HDD}(k) + c \tag{4.3}$$

Dynamic power The dynamic power denotes the power that is consumed by a server to carry out operations of the running VMs such as accessing the memory or the hard disk as well as executing instructions by the processor. As in the idle part, the major contributors to the dynamic power consumption of blade servers are the processor, memory and hard disk. Equation (4.4) models the dynamic power consumption of a multi-core i processor based on the following well known CMOS circuit formula. t denotes the total number of cores, C_{eff} the

[6] Technology enabling for a processor to choose from a set of power related saving modes.
[7] For the blade servers of the evaluated testbed, c takes a value between 70-85 W.

effective capacitance (i.e. activity factor), f the frequency, and V the voltage of the core.

$$P'_{CPU}(i) = \sum_{s=1}^{t} C_{eff}(s) \times f(s) \times V^2(s) \tag{4.4}$$

In [49], the authors showed that the power consumption of a multi-core processor is not the pure summation of the consumption of its constituent cores. Consequently, the authors modelled the power consumption of the processor by dividing it into different levels and identified the contribution of each level to the overall power consumption. For a processor, its frequency, voltage, utilization rate but also its specific energy efficient mechanism such as Intel SpeedStep [50] play a major role in the computation of the dynamic power consumption. DDR3 memory modules have a constant power consumption (in average 9.5 W) regardless read or write operations are carried out. Since in cloud computing data centres, the availability of an application profile of VMs in terms of number of accesses to the memory for read/write is challenging, the most relevant energy-related attribute is the total available free memory space (GB) of the system. This dynamic parameter helps at defining when a memory access is probable: the more free memory space the system has, the less probable will be access to the memory. Finally, the hard disk consumes power when its mechanical as well as electrical parts are used to perform read or write operations. Furthermore, an additional start up power is induced whenever the disk changes its state from idle to accessing modes. As in the case of memory, since it's not possible to have a detailed profile of the application, then the most relevant energy-related attributes for the hard disk are the read and maximum read rates as well as write and maximum write rates all expressed as MB/s. Those parameters help in providing guesses with respect to one of the three states (e.g., idle, start up, accessing) the hard disk could be. Additionally, the power to access the hard disk is in average 1.4 times greater than the one of the idle state. Further details on the probabilistic approaches adopted for memory and hard disk can be found in [51]. As a result, the overall power consumption of a blade server is the summation of the P_{idle} and the dynamic powers of its CPUs, RAM and HDDs.

Power objectives elaboration

Having the power consumption prediction models described in the previous section at the disposal of Plug4Green, we are now able to compute the power objectives. However, using directly the power consumption prediction models at each step of the optimisation process would be too costly in terms of computation time and resources. Furthermore, this approach

would not take advantage of the CP, where the objective function must be stated as a *constraint programming variable* that must be minimized, and thus cannot be written as a simple Java function. Our approach to solve this problem is the following: In a first step, Plug4Green groups the servers into families that share similar hardware characteristics (e.g., processor, memory, hard disk), and similarly the VMs are grouped into families that share similar characteristics according to the SLA (e.g., small, medium, large). Note that such an assumption is possible since it is common for a data centre to have families of similar equipment and because VMs often share similar run-time characteristics as well. Plug4Green will then generate for each server its idle and dynamic power consumption patterns under several usage conditions, using the VMs families to simulate the load, and store them in two vectors α and β. This means that the necessary values are retrieved and stored in vectors before and not during the search process which results in a much faster search. We can obtain the pre-computed version of the power consumption for the server i in family k by using the following equation:

$$P(s_i^k) = X_i \times \alpha_k + \sum_{j=1}^{p} h_{ij} \times \beta_{kl} \tag{4.5}$$

where α_k denotes the idle power of the family of servers k, and β_{kl} denotes the power consumption of the VM l if running on a server from family k. $h_{ij} = 1$ if the node s_i^k is hosting the VM v_j^l and 0 otherwise. X_i is a variable with a value of 1 if there is at least one VM in a server s_i^k, and 0 otherwise. We assume that, if a server contains no VMs, it can be switched off by Plug4Green and then consumes no energy. We denote as PUE_d the Power Usage Effectiveness of the data centre d from a federation of D data centres. The global power consumed by this federation is computed by Plug4Green as:

$$P = \sum_{d=1}^{D} (PUE_d \times \sum_{i=1}^{n} P(s_i^k)) \tag{4.6}$$

To reduce energy consumption, consolidation through VM migration is a common solution. This operation has however an energy cost that is integrated in the power objective function of Plug4Green. This way, Plug4Green will not migrate a VM if the cost of the move is too high compared to the expected energy gain. We compute the energy needed for the migration of a VM $E_{move}(i)$ based on the characteristics of the source server, the destination server and the VM itself, as detailed in the power consumption evaluations in [52]. We also include an energy penalty $E_{onoff}(j)$ for switching on and off a server. Indeed, a certain amount of time

is needed to switch on or off a server and during this time, no workload can be carried out despite a certain energy consumption.

As an approximation, we assume that the energetic situation in the data centre is stable between two reconfigurations during a delay T_{reconf} (in seconds). At the next reconfiguration and to take into account changes in the data centre like VM termination, Plug4Green will recompute the power objective. Using equation (4.6), Plug4Green computes P_{bef} and P_{aft}, the power of the federation before and after application of the reconfiguration plan, respectively. The global energy saved by the reconfiguration plan, at federation level is therefore:

$$E_{tot} = (P_{bef} - P_{aft}) \times T_{reconf} - \sum_{i=1}^{p} E_{move}(i) - \sum_{j=1}^{n} E_{onoff}(j) \tag{4.7}$$

Similarly, Plug4Green is computing Q_{total}, which is the total quantity of carbon emissions saved by the reconfiguration plan, by replacing "PUE" by "CUE" in the equations. As stated at the beginning of this section, our objectives *minEnergy* or *minGasEmission* consists of minimizing E_{tot} or Q_{tot}, respectively.

4.4.5 Reducing the solving duration

Computing a configuration according to an objective may be time consuming for large infrastructures as selecting a satisfying server for each running VM is NP-Hard [29]. To solve a COP, the constraints are used by the solver to remove inconsistent variable assignments, while the power objective variable is used to select values that are relevant to save energy. However, this can be a very time-consuming process. To help reduce this duration, so-called *search heuristics* are used to indicate to the solver the variables to focus on in priority, and supposed good values to try first. A *search heuristic* is thus attached to each objective (*minEnergy* or *minGasEmission*). The objective is to find good solutions as soon as possible in the search tree. A search heuristic is tightly coupled to an objective but completely independent from the stated constraints to maintain the composability of Plug4Green. This way, an arbitrary number of constraints can be used with the same search heuristics. In Plug4Green, the heuristics are typically guiding the solver into finding values for the variables related to the position of the VMs on the servers and the state of the servers.

For each objective, its search heuristic suggests to migrate the VMs from the least loaded, or least energy-efficient servers, to highly-loaded and energy-efficient servers. The algorithm

is similar to the well-known Best Fit Decreasing, used for example in [28]. The notion of efficiency depends on the objective: The PUE is used for the *minEnergy* objective and the CUE is used for the *minGasEmission* objective. Once the new placement for each VM is computed, the heuristics makes the solver try to turn off unused servers.

4.5 Framework Evaluation

As stated before, the goal of Plug4Green is to improve data centres energy efficiency through placement algorithms that are easy to specialize. In this section, we first discuss the extensibility of Plug4Green. We then demonstrate the impact of its specializations to reduce the power consumption or the gas emission on a heterogeneous data centre federation running an industrial workload. We finally evaluate its scalability within a simulator for a data centre with up to 2500 servers.

4.5.1 Extensibility of Plug4Green

The design of Plug4Green allows the integration of new concerns. Each constraint and objective is a plug-in composed by a XML Schema and a Java implementation. A developer can then develop a new constraint or objective as a plug-in and integrate it to Plug4Green with an automatic and deterministic specialization process. Contrary to methods derived from Linear Programming, the developer must not provide a linear model for his constraint. This eases tremendously the expertise that is required to express constraints. In practice, it took only a few hours for an engineer to create and test a new constraint.

To demonstrate this flexibility, we developed 23 placement constraints and 2 objectives. They have been developed to match a large range of expectations from the clients and data centre operators in terms of hardware compatibilities, performance level, resource sharing or energy capping. Plug4Green exposes a core set of variables and relations to manipulate VMs and servers. In the case those core variables would not be sufficient to express a user's problem, new variables can be added to express meaningful information, and then be linked with low-level relations to the Plug4Green variables. These low-level relations consist of either basic constraints in VM placement problem such as assignment, scheduling, or counting constraints, but also arithmetic, logic and domain constraints.

As an example, Plug4Green does not currently support the thermal-aware management of VMs. Current approaches [33, 34, 53] propose heuristics derived from thermal models to estimate

the impact of the VMs management on the server temperature. To integrate a thermal model inside Plug4Green, the knowledge-specific information would be defined with new variables, linked to Plug4green variables with arithmetic constraints. Once these links established, the temperature variables would be available to express new concerns: for example, capping the server temperature to disallow hotspot, or performing a thermal load balancing to reduce the cooling costs.

4.5.2 Experiments on Cloud Testbed

To evaluate the practical efficiency of Plug4Green in an environment as realistic as possible, a trial has been set inside a private cloud with a state-of-the-art cloud stack running two workloads derived from industrial traces.

Environment Setup

The cloud simulates an heterogeneous data centre federation. It is composed of two racks (see Table 4.3), each embedding a HP C7000 blade enclosure. The first data centre (DC1) has 4 BL 460c to host VMs using VMWare ESX v4.0 native hypervisor. 3 additional blades are used to manage the cloud, to schedule the workloads using the open-source scheduler *JobScheduler*[8], and to run Plug4Green. The second data centre (DC2) has 3 BL 460c to host VMs also using VMWare ESX. 2 additional blades are used to manage the cloud and to monitor the system and the energy usage of the federation using *Collectd*. The racks are connected to a single LAN and a SAN device stores all the datas, including the VMs images. For the whole duration of the experiments, we monitored the energy consumed on every nodes running an hypervisor

	Enclosure 1	**Enclosure 2**
Processor model	Intel Xeon E5520	Intel Xeon E5540
CPU frequency	2.27GHz	2.53GHz
CPU & Cores	Dual CPU -– Quad core	Dual CPU -– Quad core
RAM	24 GB	24GB

Table 4.3: Characteristics of the racks/enclosures

Plug4Green has been evaluated against 2 synthetic workloads derived from real traces within the private cloud of a corporation in Italy[9]. Figure 4.4 depicts a weekly load pattern. The first workload reproduces this trace, compressed into 24 hours. The second workload focuses on a single work day (depicted by the black frame), compressed into 12 hours. The first workload

[8] http://sourceforge.net/projects/jobscheduler/

[9] due to privacy issue, we cannot disclose the corporation name and the workload details.

considers a week-end with a low load and therefore more energy saving possibilities. The second workload is more challenging since the load is higher on average.

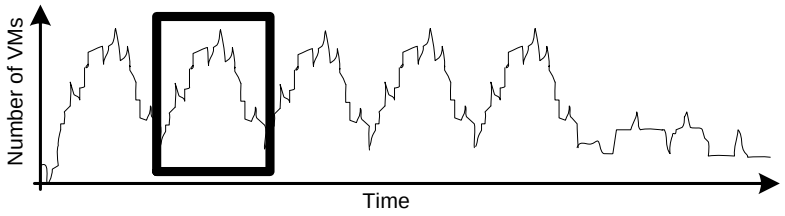


Figure 4.4: Schematic view on the weekly load patterns

Specializing Plug4Green to fit an heterogeneous federation

We evaluate here the practical efficiency of Plug4Green at managing a federation of data centres having different PUE and CUE. The experiments have been run using different data centre configurations.

In a first experiment, we evaluate the effectiveness of the *minEnergy* placement objective using 3 scenarios. In the "No P4G" scenario, Plug4Green is not used. An ad-hoc heuristic deploys the VMs on servers with a load-balancing placement objective. Idle servers are not turned off and VMs are not migrated. In the "P4G same PUE" scenario, Plug4Green is used and all the servers expose the same PUE. This is equivalent to ignoring the PUE parameter. Finally, in the "P4G different PUE" scenario, the servers in DC1 and DC2 have a PUE set to 1.5 and 2.5, respectively. Plug4Green can then benefit from the servers in DC1 that are more energy-efficient.

Figure 4.5 shows the result. The savings in the total federated sites energy increases to over 33% compared to the "No P4G" scenario, with an improvement of over 13% due to the consideration of the different PUE efficiency. In practice, we observed Plug4Green allocated more VMs in the first DC which was more energy-efficient overall with its lower PUE. During the peak of activity, we obtained an allocation of 46 VMs in DC1 and 18 VMs in DC2.

The second experiment evaluates the effectiveness of the *minGasEmission* placement objective using three scenarios similar to those used in the previous experiment. In the "No P4G" scenario, Plug4Green is not used. In the "P4G same CUE" scenario, Plug4Green is used and all the servers have the same CUE. Finally, in the "P4G different CUE" scenario, the servers in DC1 and DC2 have a CUE of 0.400 g/Wh and 0.250g/Wh, respectively. Figure 4.6 shows a reduction of the gas emissions by 34% in the "P4G same CUE" scenario with respect to "No P4G" with an increase of over 9% due to the consideration of the CUE differences. Again, the

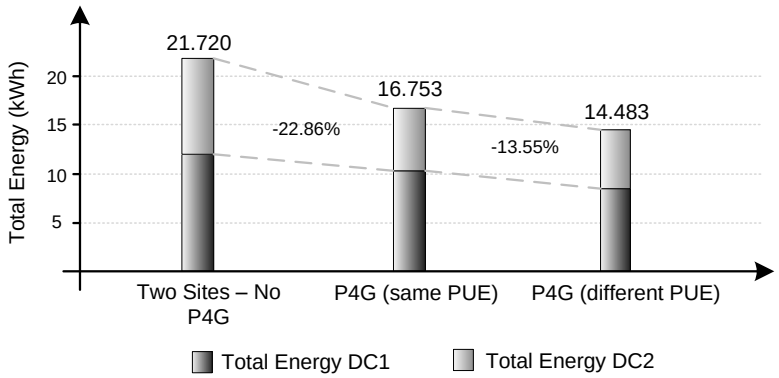


Figure 4.5: Energy consumption of two data centres with different PUE values

behaviour of Plug4Green was to run more VMs on DC2, the most efficient data centre from the emission perspective. During the peak of activity, we obtained an allocation of 26 VMs in DC1 and 32 VMs in DC2.

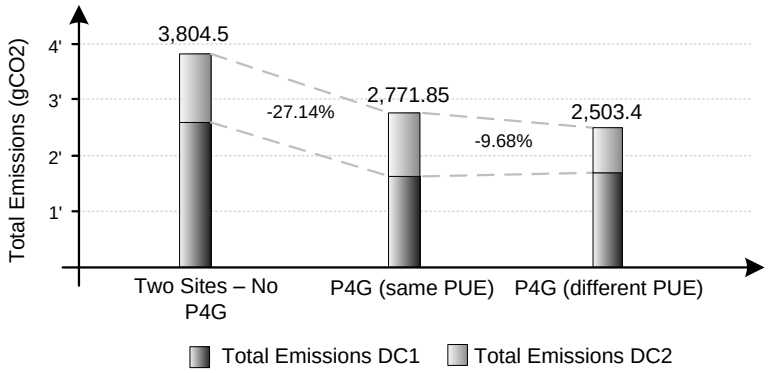


Figure 4.6: Energy consumption of two data centres with different CUE values

Specializing Plug4Green to fit the particularities of the workload

We evaluate here the practical benefits of Plug4Green when it is specialized to fit the workload. In practice, we measure the data centre energy consumption depending on different variations of a set of constraints.

The first experiment evaluates the savings when Plug4Green controls the aggressiveness of the VM consolidation. Frequent arrival of VMs may lead to a scheduling delay as additional servers may need to be booted on emergency to host them. One solution consists in ensuring at all time a certain amount of free resources (such as idle servers) in the data centre to be able to boot the VMs faster. This number should, however, be considered carefully. A too small

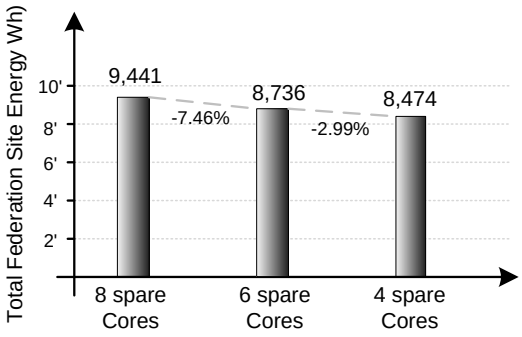
value will be ineffective while a high value would augment the overall power consumption. With Plug4Green, this fine grain tuning is done easily through a *spareCPU* constraint. A CPU is considered "spare" when it is not associated with any VCPU. Figure 4.7a shows the energy savings when the number of spare cores varies from 8 to 4 cores. This confirms that the number of spares resources should be set to a minimum to improve energy efficiency. Maintaining at most 4 cores available instead of 8 allowed an extra 10% saving.

The second experiment evaluates the savings when Plug4Green controls the frequency of the migrations. A VM migration is indeed costly in terms of energy but also in terms of performance. It is then useful to disallow to migrate too frequently the same VMs. With Plug4Green, this parameter is controlled easily through a *DelayBetweenMove* constraint. Plug4Green will then consider as candidate for migration only the VMs that have been last migrated at least the required amount of time ago. Figure 4.7b shows the energy saving depending on the migration time interval. With a 30-minute timeout, Plug4Green saved an extra 20% of energy compared to the 1 minute timeout, because it prevented unnecessary VM migrations.

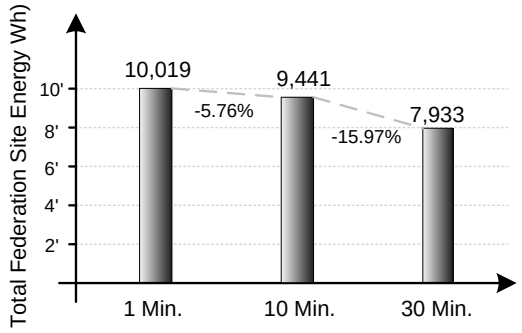
The last experiment evaluates the savings when Plug4Green is used to control the resource sharing. Its objective is to assess to which extent we can improve the achievable energy saving, when an energy relevant SLA parameter constraint is relaxed with respects to its standard value. Based on observations from the testbed, *MaxVCPUperCore* constraint has been identified as the most important one of this category. Figure 4.7c demonstrates the impact of this parameter on the overall energy consumption. If we relax the constraint up to 2.5 VCPU/core, we reduce the energy consumption by up to 45%. This is not a surprise as with a factor of 2.5 it's possible to consolidate twice as much VMs on a server than with factor of 1.2. This means that Plug4Green used only one half of the servers to run the workload and can switch off the other half.

4.5.3 Scalability of Plug4Green

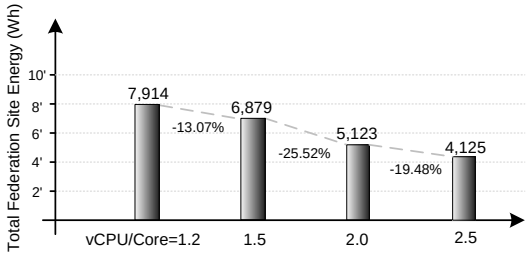
Placing VMs on servers with regard to their resource requirements is a NP-Hard problem. The scalability of Plug4Green is then determined by the size of the configurations (number of VMs and servers) and their associated constraints that Plug4Green is able to solve in a reasonable time. To evaluate this scalability, we generate 5 sets of configurations that are composed of 500 up to 2,500 servers. Each set is composed of 50 configurations, with different VM templates and different initial placements. We run Plug4Green on each configuration with the constraints evaluated in Section 4.5. The solving process stops once the first solution is computed. We then analyse the solving process and the estimated energy savings.



(a) *SpareCPUs* constraint impact



(b) *DelayBetweenMove* constraint impact



(c) *MaxVCPUperCore* constraint impact

Figure 4.7: Impact of constraints on the global energy consumption

To provide a realistic evaluation with simulation data, configurations are generated from the testbed described in Section 4.5.2 and common practices. Servers are identical to those used in the cloud testbed with an equal repartition between the models used in the two enclosures. Each VM instantiates a template randomly selected among the three available in the testbed. These templates, namely *m1.small,m1.large,m1.xlarge* mimics standard EC2 template[10] with regards to their allocated amount of RAM and VCPU. The amount of VMs in each configuration equals five times the number of servers, according to a consolidation ratio observed in industry [54]. Finally, the initial VM placement is computed randomly but ensures that their SLA is initially satisfied.

Figures 4.8, 4.9 and 4.10 depict the results. "P4G" denotes the usage of Plug4Green without any additional constraints. The "+spare" label denotes the addition of one *SpareCPUs* constraint to keep 1% of all the PCPUs directly available. The "+vcpu" label denotes the addition of a *MaxVCPUperCore* constraint to restrict to at most 2, the number of VCPU attached to a single PCPU. Finally, the "+delay" label denotes the addition of a *DelayBetweenMove* constraint to prevent the migration of any VMs migrated less than 30 minutes ago. We consider for the simulation that 5% of the VMs, randomly selected, are in this state.

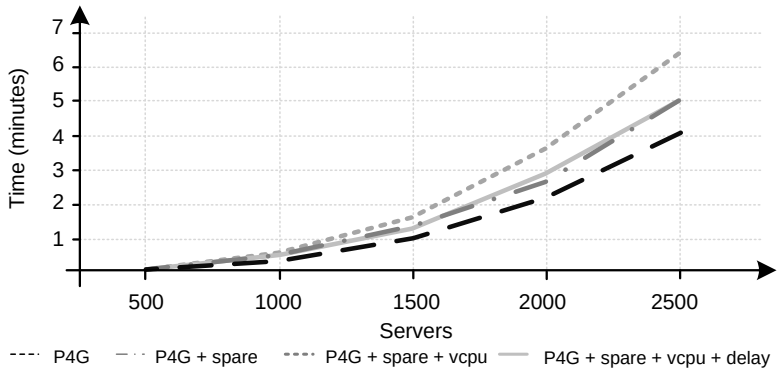


Figure 4.8: Solving duration to compute the improved configurations

Figure 4.8 shows that the solving time increases exponentially with regard to the data centre size. This is expected as the problem addressed by Plug4Green is NP-hard. Without additional constraints, 30 seconds are required to compute an improved configuration for a data centre with 1,000 servers. Doubling the size of the data centre requires 4 times more time. We however observe that Plug4Green is able to compute an improved configuration in one minute in a data centre with up to 1,500 servers running 7,500 VMs. At this scale, we observe that the addition of constraints does not alter significantly the solving process. Above that limit, the solving

[10] https://aws.amazon.com/ec2/instance-types/

time gets more dependent on the constraints. The *DelayBetweenMove* constraint reduces the computation time as it reduces the number of VMs that have to be considered by Plug4Green in each time slot. However, the *SpareCPUs* and the *MaxVCPUperCore* constraints increase the computation time by 25% each. With a 1,500 servers, this adds a 15 seconds overhead. With 2,500 servers, the overhead equals one minute. This overhead is explained by the constraints implementation: each of these constraints extends BtrPlace to expose the mapping of the VCPUs to the PCPUs. This extension injects one *bin packing* [55] constraint into the CP solver that cannot scale linearly with the data centre size. By default extensions of Plug4Green, even identical, are not shareable. This includes a redundancy that alters the performance. Providing a sharing mechanism for the extensions would lower this overhead.

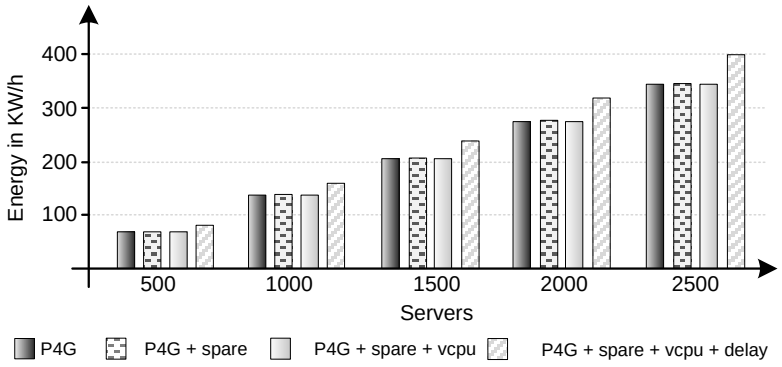


Figure 4.9: Energy consumption of the improved configurations

Figure 4.9 shows the energy consumption of the improved configurations. This value was computed using the power consumption prediction model. We first observe that the *SpareCPUs* and the *MaxVCPUperCore* constraints do not alter the quality of the improved configurations. For the *SpareCPUs* constraint, Plug4Green was able to keep free the requested amount of PCPU capacity without having to turn on additional servers. We also observe that the *DelayBetweenMove* constraint is reducing the saving opportunities by 10%. This is explained by the particular setting of this experiment: the VMs that are not allowed to be migrated due to the constraint have been selected randomly. The selected VMs are spread over numerous servers which prevented Plug4Green to turn them off.

Figure 4.10 shows the number of migrations to execute to reach the improved configurations. We observe the *SpareCPUs* and the *MaxVCPUperCore* constraints did not alter the number of migrations. This shows Plug4Green was able to consider these constraints without having to rearrange additional VMs. The *DelayBetweenMove* constraint reduces the amount of migrations by 5% which is the number of VMs considered by the constraint.

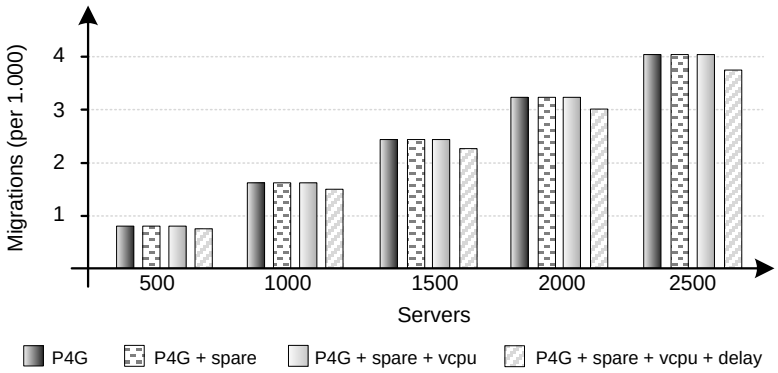


Figure 4.10: Number of migrations to reach the improved configurations

4.6 Conclusion

Trends in application design, workload volatility, but also hardware heterogeneity make each data centre unique. However, the ad-hoc design of current energy-aware VM managers prevent them to take these particularities into account to provide additional savings. In this chapter, we presented a flexible energy-aware VM manager named Plug4Green. Thanks to Constraint Programming, Plug4Green can be easily specialized to support various combinations of SLAs, power models and energy policies. Its flexibility has been verified through the implementation of 23 meaningful SLAs and 2 energy policies. Its practical effectiveness has been evaluated on an industrial testbed. While the default version of Plug4Green reduced the power consumption and the gas emission by 27% and 23% respectively, its specialization to fit the hardware heterogeneity improved the saving by up to 34%. Furthermore, additional specializations to fit the workload particularities reduced the power consumption from 9% up to 50%. Finally, scalability experiments on simulated data have shown that Plug4Green is able to compute an improved placement for 7,500 VMs running on 1,500 servers in a minute, while respecting their SLA.

In future works, we will focus on data centres powered by renewable energies. This requires indeed a new look on the energy efficient management of VMs as the nature and the availability of the energy is varying over time.

Part II Use better energies

This part introduces several techniques aiming at augmenting the renewable energy consumption in data centres. The problem of increasing the usage of renewable energy in data centres is mainly a scheduling problem. The renewable energies being often volatile and variable in time, it thus makes sense to try to schedule the workload during the hours when the maximum renewable energy is available. Furthermore, this renewable energy availability is relatively predictable. Electrical companies provides predictions of their production for the next days[11]. However, to schedule the workload of the applications with respect to the renewable energy availability, we need a way to declare their flexibility. We also need to have an insight into their energy consumption. The first chapter introduces the Energy Adaptive Software Controller (EASC), a generic software controller and interface able to schedule application workload according to energy constraints and SLA. The EASC introduces a methodology and tools allowing developers to make their application adaptive to renewable energy availability. We introduce several scheduling algorithms adapted to several application types, namely task-oriented and service-oriented applications.

The second chapter extends and generalizes the techniques presented to modern Cloud data centre paradigms, and in particular to the PaaS paradigm. At PaaS level the framework has additional knowledge about the application being run. Typically, a PaaS framework will compile an application from its source code, and then deploy it inside light weight virtual machines, or containers. Most modern PaaS frameworks also have the possibility to scale up or down applications, and to schedule various tasks within the applications. We take advantage of this existing interface to perform the renewable energy aware workload scheduling. This enhanced PaaS framework then facilitates the programming and deployment of energy aware applications.

[11] http://clients.rte-france.com/lang/an/clients_producteurs/vie/prod/prevision_production.jsp

5 The EASC, an energy adaptive software controller

Sustainable energy sources such as renewable energies are replacing dirty sources of energy in order to address the environmental challenges of the century. In order to operate data centres with renewable energies we have to mitigate their volatile and variable nature. In this chapter, we present the Energy Adaptive Software Controller (EASC), a generic software controller and interface that developers can use to make their application adaptive to renewable energy availability. Adaptivity is realized through the concept of working modes which allows to run an application under various performance levels. We advocate for a collaborative approach involving the developers of the applications in order to use the renewable energies more efficiently. The notion of EASC allows to abstract away the details of application scheduling, execution, and monitoring. We demonstrate the applicability and genericity of the EASC concept through four different instantiations. These instantiations cover two types of applications: task-oriented and service-oriented; and two kind of computing environments: Infrastructure-as-a-Service, and Platform-as-a-Service. The EASC has been trialled in the data centre of the healthcare agency of Trento, Italy and in the laboratory of HP Milan, Italy, with a mix of energy sources: national grid and local solar panels. The experimental results show how the EASC allowed to increase the renewable energies usage of 14% and 4.73% for Trento and HP Labs trials, respectively.

This chapter is derived from the paper:

- Corentin Dupont, Mehdi Sheikhalishahi, Federico M. Facca, and Fabien Hermenier. An energy aware application controller for optimizing renewable energy consumption in cloud computing data centres. In *8th IEEE/ACM International Conference on Utility and Cloud Computing*, 2015

- Corentin Dupont, Mehdi Sheikhalishahi, Federico M. Facca, and Silvio Cretti. Energy efficient data centres within smart cities: Iaas and paas optimizations. In *2015 EAI International Conference on Smart Grids for Smart Cities*, Toronto, Canada, 2015
- Sonja Klingert, Florian Niedermeier, Corentin Dupont, Giovanni Giuliani, Thomas Schulze, and Hermann de Meer. Renewable energy-aware data centre operations for smart cities - the DC4Cities approach. In *SMARTGREENS 2015*. ACM, 2015

5.1 Introduction

The rise of the energy consumption in data centres, and the high share of their electricity consumption around the world [14] induced data centres owners to take actions. Energy efficiency measures has been introduced in order to reduce the energy consumption of data centres, and now we move towards sustainable energy sources such as renewables [15] [56] [57] in order to address the current environmental challenges. For example, Google's data centres are currently operated with a 35% share of green energy[1], whereas 87% of energy consumption of Apple's data centres comes from renewables[2].

With the recent adoption of renewable energies to power data centres [15], the research community enlarges its vision to associate with purely quantitative energy consumption reduction, the notion of quality of the energy consumed, i.e. the capacity to rely as much as possible on sustainable power sources. Differently from brown energy sources, the availability of renewable energies is very volatile and time dependent: e.g. solar power is obtainable only during the day, and is subject to variations due to the meteorological conditions. The goal is then to schedule the workload of running applications according to the forecasted renewable energy availability.

The problem is, however, that data centres are rather heterogeneous environments with many different kind of applications with different kind of computing styles: for example Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). The applications running on those platforms do not cooperate to provide a scheduling of the workload that matches the times where the renewable energies are available.

In this chapter, we advocate for a generic and a collaborative approach that involves developers of applications thanks to the notion of Energy Aware Software Controller (EASC). The EASC is a generic software controller that application developers/administrators can use to make

[1] http://www.google.com/green/energy/
[2] https://www.apple.com/environment/

their application adaptive to renewable energy availability. A data centre can support a mix of task-oriented and service-oriented applications that are made adaptive to renewable energy availability. The EASC abstracts away the details of application scheduling, execution, and monitoring. The EASC constitutes the southbound subsystem of a prototype implementation of the EU project DC4Cities[3].

We make the following contributions:

- The EASC, a software controller allowing to develop renewable energy adaptive applications of any type. Software developers can integrate it into their applications to make them adaptive to the current energetic situation.
- An API, the *Energy Adaptive Software Control Interface*, allowing applications to receive energy related instructions.
- Four different EASC instantiations to demonstrate the genericity and the applicability of our concept. These instantiations cover two types of applications: task-oriented and service-oriented; and two kind of computing environments: IaaS and PaaS.
- An practical validation of the task-oriented and the service-oriented EASCs inside two data centres in Italy. The task-oriented EASC was trialled in the data centre of the healthcare agency of Trentino that is powered by the national grid, while the service oriented EASC was trialled in the lab of HP Milan that is also powered by local solar panels, in addition to the national grid. These experiments confirmed that the EASC concept allows to increase the renewable energy usage of both kinds of applications. The renewable energy percentage improvement are 14% and 4.73% for Trentino and HP Labs trials, respectively.

The remainder of the chapter is organized as follows. Section 5.2 is a review of prior work. Section 5.3 describes the EASC concept and architecture. Section 5.4 describes the various EASC instances. Section 5.5 presents the experimental results. Finally, Section 5.6 describes next steps and concludes the chapter.

5.2 Related work

A survey of the literature shows many researches addressing high energy consumption of data centres and energy adaptive approaches in applications, for example [58, 15, 57, 56] and [59]. The emergence of cloud computing provided the users with the ability to scale their services

[3] http://www.dc4cities.eu

horizontally or vertically according to the demands [60]. In addition, the notion of adaptive applications has been introduced in [61].

iSwitch [57] adapts the infrastructure itself. Servers are either powered through the grid or a local source of renewable energy. VMs are deployed by default on servers powered by renewable energy. When this source becomes scarce, the workload is migrated to servers powered by the grid and the idle servers are turned off. In order to adapt to renewables energy availability, in [62] the authors argue for either pausing VM executions or migrating VMs between sites based on local and remote energy availability.

In [15] and [56] the authors adapt the workload to renewables when it is composed by deferrable jobs. According to forecasts, the jobs are delayed to periods where renewable sources are available. The work in [58] makes a step forward by adapting map/reduce applications. The number of workers and the number of servers hosting them is adapted according to the amount of available energy. In [56], Goiri et al. proposed GreenSlot, as a solar power-sensitive scheduling algorithm for data centre workloads. GreenSlot was shown to reduce data centre costs and increase green power consumption. In [59], authors exploited workload shifting in combination with local generation as an adaptation technique for two purposes: to avoid the coincident peak, and to save energy cost. In all these papers, workloads are task-based applications.

In addition, there are similar approaches to minimize energy costs, and carbon emissions. In order to reduce electricity costs in SCs, power-aware resource management without degrading utilization has been proposed in [63, 64]. The novelty of the proposed job scheduling mechanism is its ability to take the variation in electricity price into consideration as a mean to make better decisions about job start times. In [65], authors explored the opportunities for HPC clusters to adapt to dynamic electrical prices, variation in carbon emissions within an electrical grid, and the availability of local renewables. They showed that adaptation to the renewables availability and dynamic pricing lead to significant gains. On the other hand, adaptation to the variation in the electrical grid did not led to significant gains.

By comparison with aforementioned approaches and solutions, the EASC approach allows to bring together different environments with a mix of deferrable (task-oriented) and non-deferrable (service-oriented) applications, running over IaaS or PaaS paradigms. In EASC we advocate for a generic approach that involves the infrastructure and the resource manager as well as the developers of applications.

5.3 Energy Aware Software Controller

In this section, we describe the EASC architecture and its context.

5.3.1 Overview and context

For each application in a data centre, the EASC role is to build a workload scheduling plan according to a power budget, to enact the plan and finally to monitor its activities and energy consumption. Each EASC in the data centre is able to receive a *power budget* for its application and will perform the scheduling accordingly. This is done through the interface called the *Energy Adaptive Software Control Interface*. This interface defines a standardized communication mean for all energy related information toward the applications of a data centre. The power budget transmitted consists in a recommended power consumption for each time of the day.

The power budget for each application is computed by an external component called the CTRL. It is computed using informations from the data centre and from the energy provider, such as the energy availability and source mix forecasts. This information will be used to determine the reference consumption levels for each applications, in order to meet the energy and power goals set at data centre level.

5.3.2 Architecture

This section describes the EASC architecture, its components and APIs. Figure 5.1 illustrates a high-level architecture of the EASC components, including its API interface toward the CTRL. The EASC implements workload scheduling techniques that can reconfigure the applications according to the power budgets. The power budget is expressed as a series of time slots, covering a full day, with a power associated to each slot. Each slot denotes a recommended amount of power that the application should consume. We assume a time slot is a 15 minute period in our experimentation. The EASC selects the possible working modes (WMs) for the application according to the power budgets, the energy profiles of the working modes, and the SLAs.

The EASC allows the application operator to define the KPIs of his application, its SLA and its working modes. The KPIs are measured in term of a number of *business items* per unit of time. For example, the operator can define the business item to be the number of web pages served by his application, a number of files processed or a number of reports generated. The

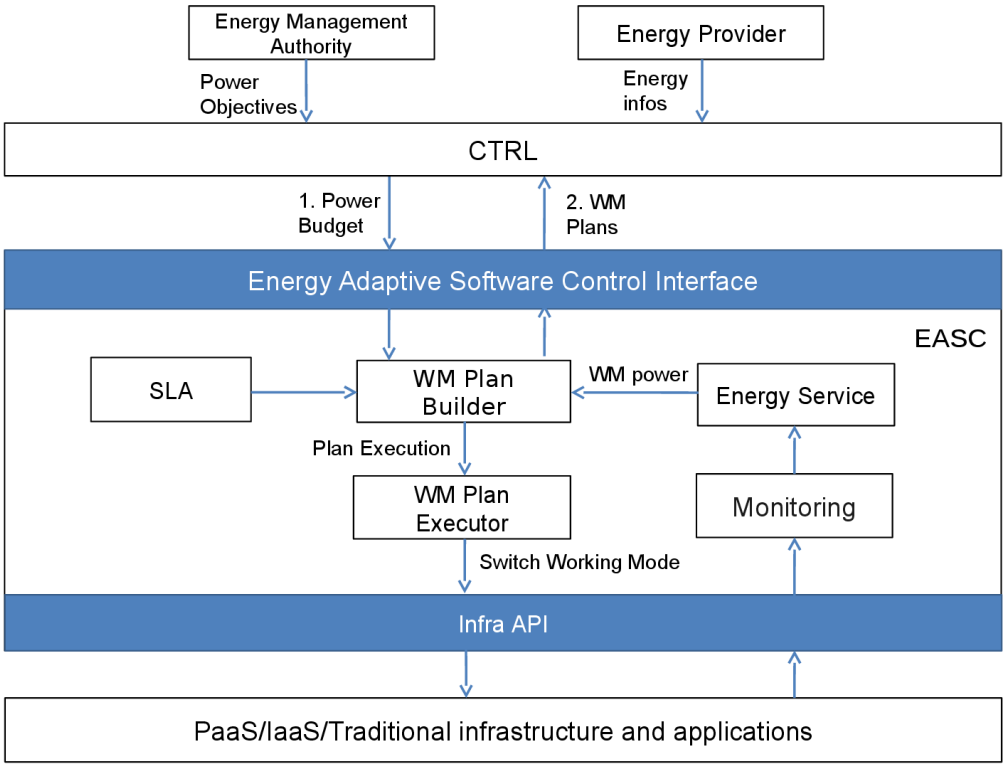


Figure 5.1: The EASC architecture

SLA are defined as a threshold on the KPIs that the EASC must guaranty. Depending on the type of application, the objectives of the SLA can be cumulative (task-oriented application) or instantaneous (service-oriented application). A working mode consists of:

- an actuator able to start and stop the working mode,
- an estimated power consumption,
- a maximum attainable performance for the KPI defined,
- a certain number of resources associated to that working mode.

Based on this information, the EASC computes a scheduling for the application called the working mode plan. A working mode plan is composed of a working mode for each time slot. The EASC shall be able to reconfigure the application according to the working mode plan selected. For example, the EASC will send to the cloud manager reconfiguration actions in order to instantiate or stop VMs, boot or stop nodes. In addition, the EASC monitors the *work done* by its application in term of business items processed. We can then deduce a *work left to do*: it is the difference between the guaranteed number of business items (SLA) and the work done. This work left to do is what we have to schedule within the working mode plans.

In terms of monitoring, the EASC shall be able to measure the KPI of the application at any moment. The EASC calls a so-called *Energy Service* to compute the expected energy profile for each working mode. This service predicts the consumption of the various components of the infrastructure (servers, VMs, containers). For each component of the data centre, a regression analysis is performed by the Energy Service on historical data in order to find a model of its consumption.

5.4 EASC instantiations

This section introduces the implementations of the EASC concept for the two application types.

5.4.1 EASC for task oriented applications

Task oriented applications are characterized by a certain amount of tasks to perform. Those tasks have a minimum start time, a deadline, and a wall-time. Some of those tasks are deferrable: for example an anti-virus scan have to be performed every day, but can be scheduled at various time of the day. Other examples of task oriented applications include video conversion services or report generation applications.

When generating the working mode plan, the EASC can follow the following policies:

- *Proportional*: this policy schedules the tasks in such a way that the expected energy consumption profile is similar to the power budget profile.

- *Aggressive*: this policy schedules all the tasks during the hours where there is the maximum renewable energy availability. In a typical day with solar energy available, the EASC will use the most powerful working modes during the central hours in order to complete the tasks. However, it is relatively risky; if the renewable energy forecast changes through time there is a risk that the SLA will not be met.

- *Eager*: this policy schedules all tasks at the earliest possible. This policy is more appealing when there is uncertainty over the future availability of renewable energy; therefore it is rather conservative.

The algorithm presented in Figure 5.2 schedules the tasks of the application so as to follow the policies presented. It presents two parameters: the aggressiveness and the eagerness. The aggressiveness controls the possibility for the application to consume more or less aggressively

the renewable energies. For example, at a high aggressiveness level the application will run at the highest performance level when renewable energies are available, and at the lowest performance when they are not. On the other hand, the eagerness controls the necessity for an application to complete its tasks more or less early. A low eagerness allows to be more flexible with regard to the scheduling of the applications tasks during the period of availability of the renewable energies.

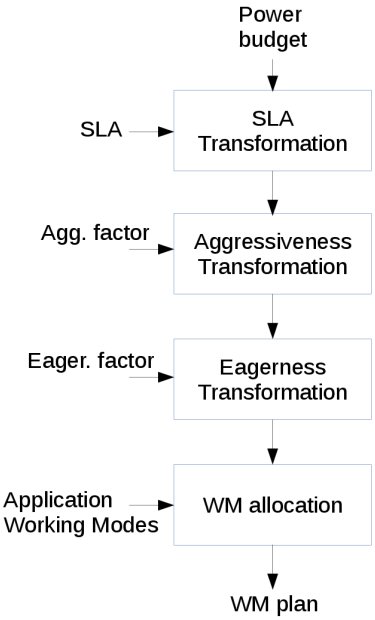


Figure 5.2: EagerAgg algorithm overview

In the algorithm 1, the function $AggTransform$ transforms a power budget by favoring the times of the day where there is more renewable energy. The resulting power consumption plan consumes the same amount of energy that the input power budget, but it presents more power in the maxima of the curve, and conversely less power in the minima of the curve. This allows the function WMAllocation to select more powerful working modes when renewable energy is available, and thus complete more quickly the tasks to perform. A high $aggFactor$ makes the application consumes the renewable energies more aggressively. The function $EagerTransform$, on the other hand, transforms a power budget by favouring the first time slots in the power budget. It uses a vector $defEagerVector$, which have the same size than the power budget. This vector $defEagerVector$ defines a simple descending array of powers, for example [100W, 99W .. -99W, -100W]. It represents the preference for the early time slots in the power budget. The transformation defined in $EagerTransform$ then transforms the power budget so as to allocate more power to the early time slots. The input factor $eagerFactor$ allows to tune this operation according to the preference of the

Algorithm 1 EagerAgg algorithm

```
1: constant workingModes : List of WorkingMode

2: function AggTransform(pb : Listof(TimeSlot, Pow), aggFact : Float)
3:     for all (timeSlot, power) ∈ pb do
4:         power ← power ∗ aggFact
5:         ▷ normFactor is computed so that the total energy of the final power budget remains the same than the initial one
6:         power ← power ∗ normFactor
7:     end for
8:     return pb
9: end function

10: function EagerTransform(pb : Listof(TimeSlot, Power), eagerFactor : Float)
11:     ΔBP ← max(pb) − min(pb)
12:     eagerVector ← defEagerVector ∗ ΔBP ∗ eagerFactor
13:     ▷ sum of two vectors of equal length
14:     eagerBP ← pb + normEagerVector
15:     return eagerBP
16: end function

17: function SLATransform(pb : Listof(TimeSlot, Power), sla : Float)
18:     ▷ normFactor is computed so that the total energy of the final power budget allows to run the correct working modes
    during enough time to cover the full SLA
19:     normFactor ← f(workingModes.maxBizPerf, sla)
20:     for all (timeSlot, power) ∈ pb do
21:         power ← power ∗ normFactor
22:     end for
23:     return pb
24: end function

25: function WMAllocation(pb : Listof(TimeSlot, Power))
26:     ▷ sort BP by power
27:     sortedBP ← sort(pb))
28:     for all (timeSlot, power) ∈ pb do
29:         ▷ select the WM that have the power closest to the current power budget power
30:         plan(timeSlot) ← nearest(workingModes, power)
31:     end for
32:     return plan
33: end function
```

application for running its tasks eagerly or not. Like for $AggTransform$, this transformation allows the function $WMAllocation$ to select more powerful working modes at early time slots, and thus complete earlier the tasks to perform. The $SLATransform$ ensures that the power budget has enough energy to allow the application to complete its full SLA. If it is not the case, a normalization factor is applied to the power budget to augment it or reduce it accordingly. The complete $EagerAgg$ algorithm is the simple composition of these four functions.

5.4.2 EASC for service oriented applications

Service oriented applications, by contrast with the task oriented applications, do not define specific tasks with a start and a stop dates. On the contrary, they have to maintain a certain level of performance all the time, to serve clients. Typical examples of such applications include Web servers, database services and mail servers. The services are characterized by one or several KPIs together with their mandatory levels for each time-slot. For this type of application we created an algorithm called the *MinMaxAgg* presented in algorithm 2 and Figure 5.3. To allow a certain flexibility of the application, a system of reward/penalty is included in the SLA. In accordance with the client, if the application is running slightly under the SLA, a reward/discount is paid by the data centre.

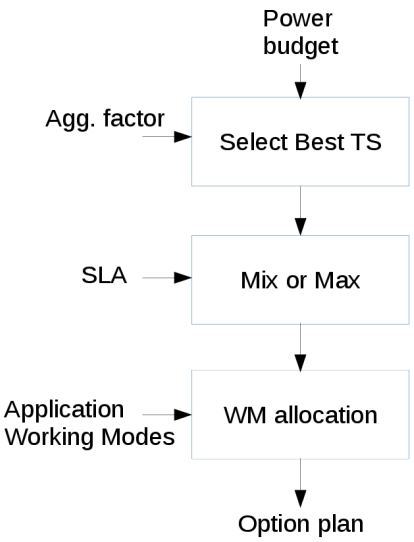


Figure 5.3: MinMaxAgg algorithm overview

In this algorithm, the function $BestTS$ selects the X timeslot of the day where there is more power in the power budget. Those timeslots are generally corresponding to the best hours for renewable energy availability. The factor X is computed relative to the aggressiveness factor.

Algorithm 2 MinMaxAgg algorithm

```
1: constant SLA: List of (TimeSlot, BusinessPerf)
2: constant workingModes : List of WorkingMode

3: function MinMaxAgg(pb : Listof(TimeSlot, Power), numTS : Int)
4:     return MinMax(BestTS(bp, numTS))
5: end function

6: function MinMax(filteredBP : Listof(TimeSlot, Power))
7:     ▷ for the best timeslots
8:     for all (timeSlot, power) ∈ filteredBP do
9:         ▷ select the WM closest to the power budget, without violating the SLA
10:        plan(timeSlot) ← getWM(SLA(timeslot).businessPerf, power)
11:    end for
12:    ▷ for the rest of the timeslots
13:    for all (timeSlot, businessPerf) ∈ (SLA \ filteredBP) do
14:        ▷ select the WM that have the business performance just above to SLA business performance
15:        plan(timeSlot) ← above(workingModes, businessPerf)
16:    end for
17:    return plan
18: end function

19: function BestTS(pb : Listof(TimeSlot, Power), numTS : Int)
20:    ▷ sort BP by power
21:    sortedBP ← sort(pb))
22:    ▷ get the numTS first timeslots
23:    filteredBP ← take(sortedBP, numTS))
24:    return filteredBP
25: end function

26: function getWM(minBusinessPerf : Float, power : Power)
27:    ▷ get only the WM that have a sufficient performance
28:    filteredWM ← filter(workingModes, workingMode.businessPerf > minBusinessPerf)
29:    ▷ get the WM that have a power closest to the reference power
30:    filteredWM2 ← nearest(filteredWM.power, power)
31:    return filteredWM2
32: end function
```

During those best times, a working mode allowing a better performance than requested by the SLA is selected. During the rest of the hours, a working mode slightly under the SLA is applied. Reward and penalty apply correspondingly, they are calculated so that they cancel each other.

5.5 Experimentations and evaluation

In this section, we present the experimentations within two trial sites in Trento and Milan. We have measured *RenPercent* metric in order to evaluate the proposed solution in terms of renewable energy usage. *RenPercent* expresses the percentage of renewable energy consumed by a DC in a certain period of time (1 day, 1 week, etc.). This metric is the main metric to assess the achievement of the EASC. With this metric we can compare the coverage of the DC energy consumption of renewable energy before and after implementing the EASC.

5.5.1 Trento trial

In this trial, an EASC encapsulates a real application producing medical reports, within the data centre of the healthcare agency of the Trentino province.

Application specification

Trento trial application is a compute-intensive task-oriented application. Users need to wait some minutes to get the report generation done. The report generation process can be scheduled in advance; thus, the idea is to prepare a cached copy of each report and have it ready once the user asks for it. This leaves an opportunity for EASC to shift the report generation in time during a day. The only constraint (as per SLA) that EASC planning has to respect is that 780 reports have to be generated within a day, from midnight to midnight.

The application business unit for this trial is the *Report* and the business performance (BizPerf) unit is the number of reports generated per minute. Table 5.1 defines five working modes based on the number of parallel processes that run simultaneously to generate reports; *minimum* corresponds to execution of one process at a time (sequentially); *medium1* corresponds to execution of 4 parallel processes, etc. *ServersOff* working mode was defined for referring to the case of no activity execution (main production servers off). The baseline has been chosen with the application producing reports continuously until the SLA is reached, at a low level of performance using the working mode *minimum*.

In this trial, hardware resources include two IBM rack servers as compute nodes to run trial

Table 5.1: Trento trial working modes

WM	Processes	BizPerf	Power (Watt)
ServersOff	0	0	19,5
minimum	1	0,53	665,8
medium1	4	1,88	761,9
medium2	7	3,0	844,4
maximum	10	3,33	864,2

application: IBM xSeries 366 8863-3RG: 4 Intel® Xeon® Processor 7020 (2M Cache, 2.66 GHz, 667 MHz FSB) 4 cores, RAM 32 GB, HDD 30GB-RAID1, HDD 200GB-RAID5.

Energy mix

The trial has a single power source from the Italian national grid. We followed a Measurement & Verification (M&V) methodology[4] and built six days profile that reflects typical days in various seasons in Italy. This methodology allowed us to simulate an entire year trial run with just 6 different profiles, one per day.

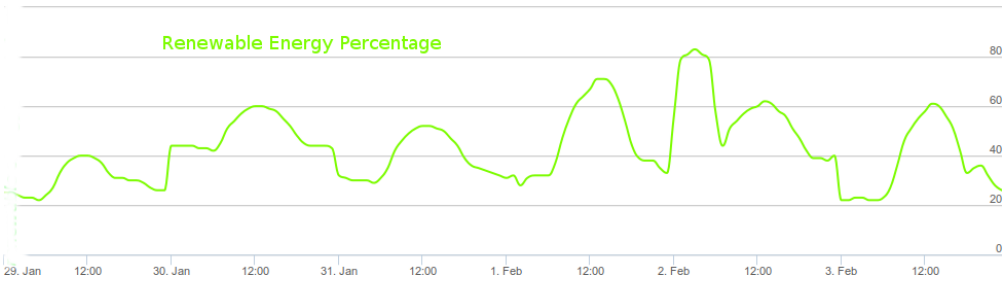
Evaluation

Figure 5.4a shows the renewable energy percentage in the Italian grid for the six profiles consecutively. Figure 5.4b represents power budget, real power consumption, and baseline power consumption. This graph shows that the baseline power (blue curve) is flat as expected, it also shows that the real power (grey curve) follows closely the power budget (orange curve). The peaks in the renewable energy percentage are always accompanied by peaks in the power budget and real power curves. Comparing Figures 5.4a and 5.4b we can observe that the EASC is able to follow the renewable energy percentage curve. In sum, the graphs show that the EASC tries to exploit the peaks in renewable energy by switching to powerful working modes in order to perform all the tasks in the shortest time possible.

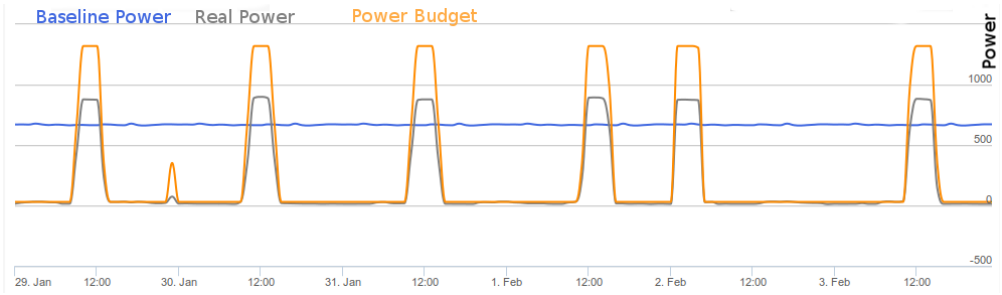
Table 5.2 presents the numerical results of RenPercent metric for all six days. In addition, this table presents average results of all six days. The last row presents results for a simulation of the entire year (six days spread on an entire year).

Considering all six days together the renewable energy percentage went from 43.1% up to 57.9%, an absolute improvement of 14%. The renewable energy usage improvement is greater than 10% for all days. Moreover, the renewable energy usage is very close to the maximum

[4] http://www.evo-world.org/en/



(a) Renewable energy percentage in national grid



(b) Power actual vs power baseline vs power planned

Figure 5.4: Trento trial execution behaviour

Table 5.2: Trento trial results

Profile	**RenPercent**	**RenPercent baseline**
Day 1	37.65%	30.03%
Day 2	58.30%	49.24%
Day 3	49.73%	39.19%
Day 4	66.56%	45.98%
Day 5	77.94%	57.27%
Day 6	56.29%	37.06%
All days	57.88%	43.13%
A year	48.22%	37.39%

percentage of renewable energy available in the grid; this difference is always less than 5%.

5.5.2 Milan trial

In this section, we describe Milan trial that presents a service-based application. The experiment takes place in HP Italy Innovation Center lab-grade resources hosted at HP premises in Milan, Italy.

Application specification

HP trial workload is based on a Web application called HP Life. It is a testing lab for a worldwide service offered by HP. This application needs to be always available to the users (24 hours per 7 days) as it offers a Web based e-learning platform for entrepreneurs spread over the globe. Due to this type of SLA, this trial is quite different from the Trento trial; furthermore the workload cannot be shifted in time. The KPI for this service is the total number of requests served per minute (Requests/minute). There is a specific SLA value for each timeslot of the day expressing the expected business performance during that time interval. The red curve in Figure 5.5 presents the SLA. The other coloured curves represent the trial performance for each day.

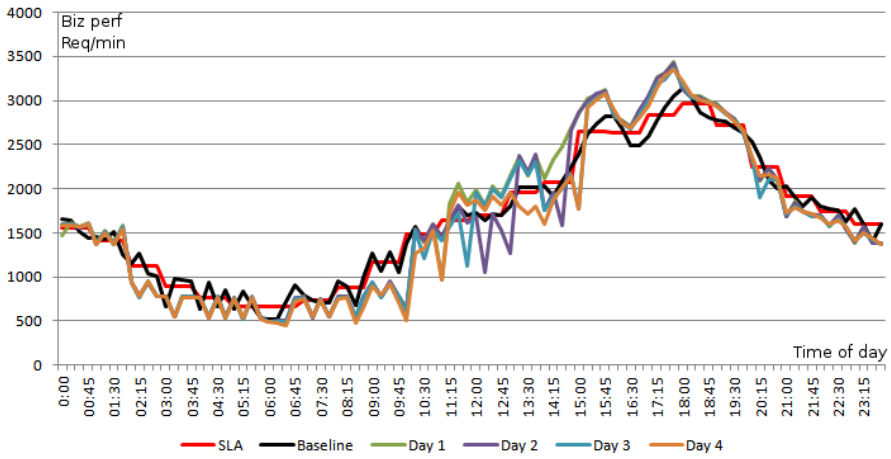


Figure 5.5: SLA and measured performance for each day

In this trial, a working mode can span over distributed data centre resources around the world, i.e. Region US West, Region US East, Region Europe. Working modes definition are aligned to the concept of active region (data centres sites) balancing the workload of the system:

- WM3SF: All 3 sites work at full capacity (WM-ID: 30).
- WM2SF1SM: 2 sites work at full capacity, 1 site works at minimum capacity (WM-ID: 25).
- WM2SF: 2 sites work at full capacity (WM-ID: 20).
- WM1SF1SM: 1 site works at full capacity, 1 site works at minimum capacity (WM-ID: 15).
- WM1SF: 1 site works at full capacity (WM-ID: 10).
- WM1SM: 1 site works at minimum capacity (WM-ID: 5).

WM-ID is proportional to the amount of used resources and corresponds to the working mode power. The WM-ID are numerical values when displaying working modes in the daily graphs. Figure 5.6 represents the overall power consumption of the system when running in a certain working mode (on the Y axis) while delivering a certain business performance (on the X axis). It is visible on this graph that each successive working mode allows to reach a bigger business performance, but at a bigger energetic cost overall: the curve for a given working mode is practically always above the one of the previous working mode.

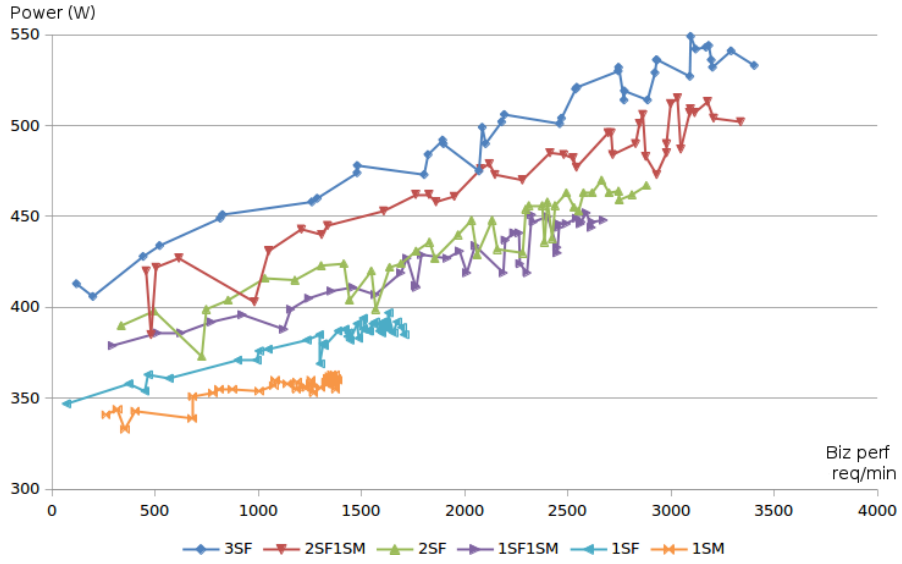


Figure 5.6: Business Performance (Req/min) versus power consumption (W) for each working mode

The baseline has been measured on default system configuration, i.e. when all servers are turned on and operational (a trial run with working mode WM3SF).

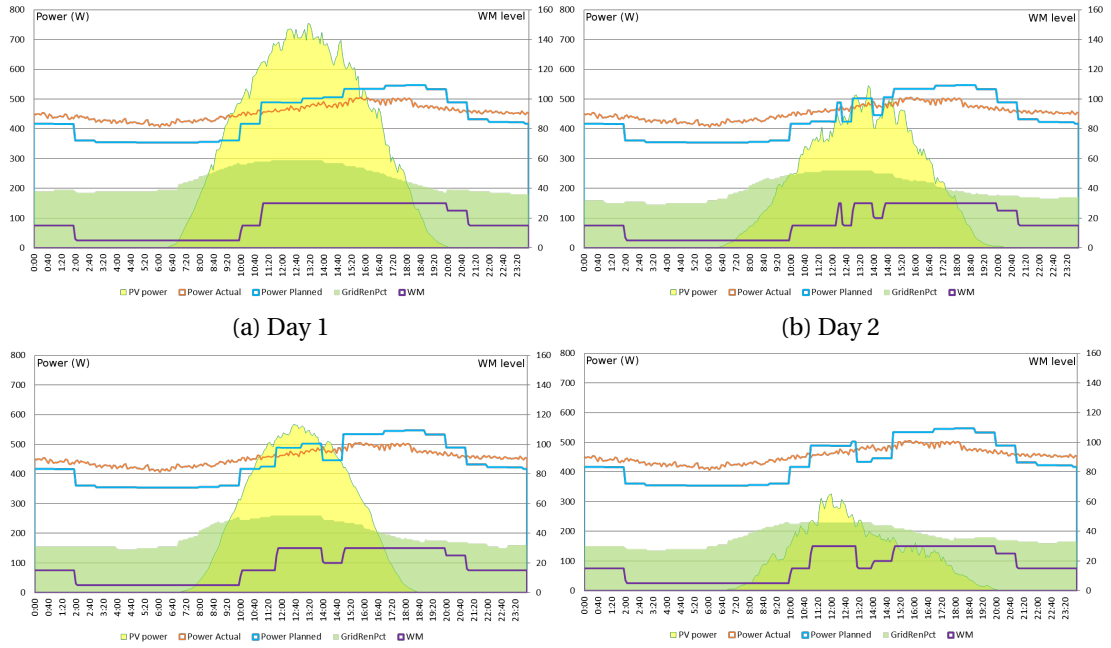
Energy Mix

The HP experiment has a dual power source: one from Italian national grid, and the other from local renewable generation. On top of the roof of the lab premises, 4 dedicated photo-voltaic (PV) panels have been installed. Each panel provides a maximum power of 250W; thus with 4 PVs configuration, the maximum power generation is 1KW. This energy context offers a unique combination of energy sourcing.

Following M&V methodology, we recorded 4 profiles for the PV production on typical days. With its aggregation with the Italian national grid we built four days profile that reflect a full-year behaviour of energy ecosystem for HP Experiment in Milan.

Evaluation

HP trial has been run for each 4 day profiles once with the baseline, and once with the EASC.



(a) Day 1

(b) Day 2

(c) Day 3

(d) Day 4

Figure 5.7: HP trial execution behaviour with the EASC

Figure 5.7 represents an in depth view of trial run for each day. The X axis is the hour of the day; the yellow area represents the PV power (primary Y axis is power in Watt) while the light green area is the renewable percentage of the grid (secondary Y axis on the right). The light blue line is the planned power (power budget in terms of EASC) and the red line is the actual power. The purple line in the bottom part is just a qualitative indication of the selected working mode (numerical WM-ID), ranging from 5 (1SM) to 30 (3SF) in steps of 5 units (see previous WM-ID table).

Similar to the Trento trial, the graphs in Figures 5.5 and 5.7 show that the working mode power and business performance curves follow the planned power curve (power budget). The peaks in the light blue line (the planned power) are always accompanied by peaks in the purple line (working mode power), and the red line (actual power). Overall, the graphs demonstrate how the EASC exploits the peaks in renewable energy by switching to powerful working modes, and vice versa to lower performance working modes when there are less power budget while still respecting the SLA (the red curve in Figure 5.5).

Table 5.3 presents the numerical results for RenPercent metric. The average improvement for

Table 5.3: HP experiment trial results.

Profile	RenPercent baseline	RenPercent EASC	RenPercent increase (pp)	RenPercent increase (%)
Day 1	68.20%	70.34%	2.14pp	3.13%
Day 2	61.85%	64.68%	2.83pp	4.57%
Day 3	58.57%	61.65%	3.08pp	5.25%
Day 4	53.99%	57.26%	2.27pp	6.05%
Trial Days	60.65%	63.52%	2.87pp	4.73%

the RenPercent metric is 4.73%. We observe that the improvement is less than the Trento trial. This is due to the demanding SLA of HP trial, and being a service-based application.

Table 5.4: HP experiment trial: power and performance comparison

Profile	Total Power Cons. (KWh)	Total BizPerf (req. served)	Efficiency (Req/W)
SLA		479K (ref)	
Baseline	10.91 (ref)	480K (100.0%)	43.98 (ref)
Day 1	9.42 (-13.60%)	484K (101.0%)	51.40 (+16.87%)
Day 2	9.28 (-13.14%)	474K (98.87%)	51.14 (+16.28%)
Day 3	9.46 (-13.24%)	471K (98.25%)	49.80 (+13.23%)
Day 4	9.43 (-13.51%)	466K (97.22%)	49.43 (+12.29%)

Table 5.4 summarizes the total power consumption, the total amount of work done (requests served during the day), and the "requests served/energy consumed" for each day of the trial and for the baseline. In terms of energy efficiency (second column in the table), this trial presented a significant amount of energy saving (over 13% energy consumption reduction). In addition, the Figure 5.5 illustrates that EASC optimization is not causing significant violations of the SLAs during the trials. The total work done (BizPerf) ranges between +1% to -2.78% with respect to SLA, as shown by numerical results presented in the Table 5.4 (third column). This demonstrates that work done is not significantly affected. Overall, the energy efficiency of the whole application is increased: the improvement in terms of BizPerf per Power, i.e. Work done/Watt, ranges between 12.29% and 16.87% (fourth column).

5.6 Conclusion and Future Work

The recent adoption of renewable energies to power data centres brings new challenges. While strong efforts are continuously made to reduce the energy consumption, the intermittent nature of the renewable sources imposes also to align the performance of the applications with the energy availability periods.

We presented the notion of Energy Adaptive Software Controller (EASC), a generic software controller that developers can use to make their application adaptive to renewable energy availability. To integrate the notion of EASC into a legacy application, a developer only needs to identify its various KPIs, working modes and to declare the commands to use to enact a working mode. The EASC then provides different scheduling algorithms to continuously choose the most appropriate working mode to use for the controlled application with regards to its SLAs and a power budget.

We validated the portability of the notion of EASC through the complete implementation of two legacy applications. The first one is a task-oriented application used to generate reports in the healthcare domain. The second one is HP Life, an international eLearning lab available through a Web application. We also confirmed the practical benefits of the developed EASCs on two different testbeds: one only powered by the Italian national power grid, and the other that is the case of HP life, through a dual power sources from photovoltaic arrays, and national power grid. Experiments over a week also proved that the EASCs increased the usage of renewable energies by aligning the application performance with the energy availability periods.

6 Energy optimizations within the PaaS and IaaS paradigms

In this chapter, we expand on the notion of EASC and show that IaaS/PaaS hybrid data centres offer specific opportunities in order to increase their usage of renewable energy. We present an infrastructure able to make PaaS applications adaptive to renewable energy availability. Furthermore, we created an energy model able to evaluate simply the power consumption of applications in a shared infrastructure such as PaaS. Our prototype has been trialled in the data centre of the healthcare agency of Trento, Italy. The experimental results show how the EASC allowed to increase the renewable energies usage by 7.07 percentage points, while the error rate of the power model was 16.58%.

This chapter is adapted from the paper (submitted):

- Corentin Dupont, Mehdi Sheikhalishahi, and Michele Santuari. Improving renewable energy consumption in iaas/paas hybrid data centres. Submitted to Futur Generation Computer Systems, 2016

6.1 Introduction

As introduced in the previous chapter, the recent adoption of renewable energies to power data centres opens new research challenges. In particular, it is necessary to schedule the workload of the running applications in order to mitigate the volatile nature of the renewable energies.

On the other hand, data centres are rather heterogeneous environments with different kind of computing styles: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and

Software-as-a-Service (SaaS). In addition, the applications running on those platforms do not cooperate to provide a scheduling of workloads that matches the times where the renewable energies are available.

Hosting a PaaS infrastructure inside a IaaS is a common practice. The IaaS layer is providing the hardware abstraction, such as VMs. The PaaS layer is then deployed inside those VMs and provides the applications and developers abstractions[1]. For example, IBM Bluemix uses this hybrid infrastructure, PaaS supported by IaaS. This allows to decouple the concerns while keeping the advantages brought by both worlds. However current hybrid cloud infrastructures are most of the time not adaptive to energy constraints. For example Cloud Foundry[2], one of the biggest Open Source PaaS platform, does not support automatic Droplet Execution Agent[3] (DEA) scaling. IaaS and PaaS Cloud services require to put in place mechanisms able to achieve consolidation of resources at both IaaS and PaaS infrastructures.

Some other PaaS providers, on the other hand, do provide auto-scaling. For example Amazon Elastic Beanstalk will boot new VMs [4] based on user defined thresholds. Heroku provides some possibility for auto-scaling[5] Dynos (which are custom Linux containers), however it is not known if the underlying infrastructure is scaled up and down too.

In this chapter, we present an adaptive and reactive cloud infrastructure composed of IaaS and PaaS Cloud services capable of minimizing the amount of infrastructure resources usage according to applications Service Level Agreements (SLAs) and to optimize the usage of renewable energy consumption. We design and implement a coordination mechanism between PaaS and IaaS able to achieve consolidation of resources at both levels. This hybrid cloud architecture has been trialled in a real data centre within APSS[6], the healthcare agency of Trento in Italy.

In order to schedule correctly the PaaS applications in the data centre according to energy constraints, we first need to predict the power consumption of those applications running in certain execution levels. We present an energy prediction model able to predict simply the energy consumption of each application within a shared infrastructure, e.g. PaaS. This research finding is another contribution in which to the best of our knowledge has not been explored in the literature. For instance, in [66], authors mentioned that PaaS has no means to

[1] http://www.ibm.com/cloud-computing/bluemix/
[2] https://www.cloudfoundry.org/
[3] https://docs.cloudfoundry.org/concepts/architecture/execution-agent.html
[4] http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.as.html
[5] https://elements.heroku.com/addons/adept-scale
[6] https://www.apss.tn.it

estimate the energy consumption of applications; however they did not provide any insight for this problem.

This chapter makes the following contributions in the field:

- We design and implement a cloud architecture for an adaptive and reactive cloud infrastructure composed of IaaS and PaaS cloud services able to respond to external factors like applications' SLAs and renewable energy availability.
- We present a power prediction model able to predict the power consumption of applications running within a PaaS-IaaS infrastructure.
- We experimentally validated the cloud architecture within such a hybrid Cloud infrastructure. It has been trialled in a real data centre within healthcare agency of Trento in Italy with proof-of-concept results.

The remainder of the chapter is organized as follows. Section 6.2 is a review of prior work. Section 6.3 presents the Cloud architecture. Section 6.4 presents the energy sharing prediction model. Section 6.5 presents the trial and the experimental results. Finally, Section 6.6 describes the future works and concludes the chapter.

6.2 Related Work

In this section, we review the works on PaaS/IaaS coordination and on containers energy management.

6.2.1 IaaS/PaaS coordination

As introduced in previous chapters, there has been a lot of research on VM consolidation approach to reduce energy consumption at IaaS layer [67, 68, 38, 69, 6]. However, in cases in which a PaaS is deployed within a IaaS the consolidation mechanism becomes more complex and it requires a mechanism to coordinate PaaS resource fluctuations with IaaS consolidation. At present, little research exists to address the optimization needs of a hybrid cloud infrastructure, e.g. PaaS inside IaaS, through an adaptive and reactive cloud architecture. On this front, [66] proposes a co-design of IaaS and PaaS layers for energy optimization in the cloud. For that, two complementary approaches for establishing such cooperation, namely cross-layer information sharing, and cross-layer coordination, have been proposed. Significant energy gains could be obtained by creating a cooperation API between the IaaS layer and the PaaS

layer. However, as a position paper they do not provide any blueprint of an architecture for their proposal. The architecture that we propose envision a very minimal communication between the IaaS and PaaS layers: the PaaS layer is only triggering the IaaS layer for reconsolidation, with no other information being transmitted between the two. This allows to keep the separation of concerns between the IaaS and PaaS paradigms.

Regarding multi layer coordination, a related research domain is Autonomic Computing. Autonomic Computing has been exploited in the design of cloud computing architectures in order to devise autonomic loops aiming at providing coordinated actions among cloud layers for efficiency measures, turning each layer of the cloud stack more autonomous, adaptable and aware of the runtime environment [70, 71].

In order to reach a global and efficient state due to conflicting objectives, autonomic loops need to be synchronized. In [70], authors proposed a generic model to synchronize and coordinate autonomic loops in cloud computing stack. The feasibility and scalability of their approach is evaluated via simulation-based experiments on the interaction of several self-adaptive applications with a common self-adaptive infrastructure.

In addition to elasticity, scalability is another major advantage introduced by the cloud paradigm. In [72], automated application scalability in cloud environments are presented. Authors highlighted challenges that will likely be addressed in new research efforts and present an ideal scalable cloud system.

6.2.2 PaaS and containers energy management

Although VM performance have considerably improved over time, containers have several advantages over them [73, 74]. Overall, they are faster to boot: in the order of 50ms while it is several seconds for VMs. Their RAM footprint is smaller than the VMs, due to the absence of kernel redundancy. However both virtualization forms introduce negligible overhead for CPU and memory performance. This small RAM footprint and fast boot time makes them ideal candidates for the scheduling of renewable energies.

Regarding energy management, in a very recent work [75], the authors propose a framework and algorithm for energy efficient container consolidation. Their proposed deployment is very similar to ours, with VMs hosting containers. This type of deployment is sometime called Containers as a Service (CaaS). As a difference, our work goes further by also using the capacities of the PaaS to describe the application capacities in term of scalability and flexibility, in order to increase the use of the renewable energies.

According to [42], approaches to real-time power modeling fall into two categories: detailed analytical power models, and high-level black-box models. With difference with the analytical power model proposed in Chapter 4 (and the related works), the power model proposed in this section behaves as a "black box". This allows to simplify drastically its configurations.

6.3 PaaS architecture

In this section, we will present the concept of EASC adapted for the PaaS context.

6.3.1 Overview and context

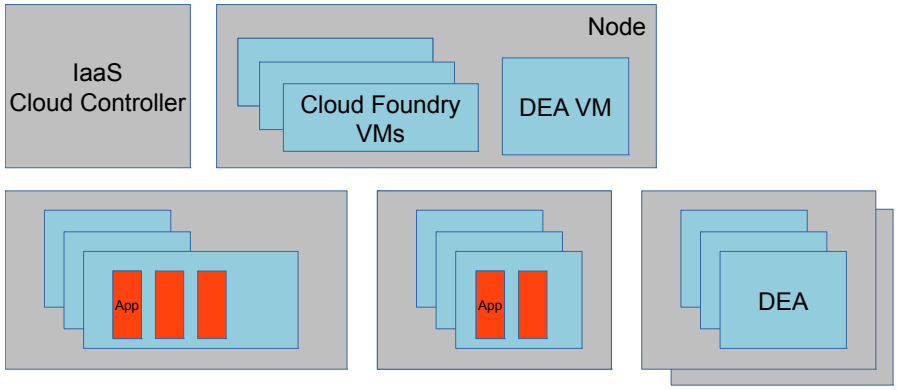
The EASC role is to build a workload scheduling plan for each application in a data centre according to a power budget, to enact the plan and finally to monitor its activities and energy consumption. The complete description of the EASC is provided in the Chapter 5. The EASC design has been extended to support Platform-as-Service based applications with EASC-PaaS. PaaS provides services to easily provision, scale, and monitor applications with a limited user/administrator intervention. In EASC-PaaS the concept of working mode is mapped to the scaling operation services provided by PaaS infrastructures.

However in a traditional PaaS environment, scaling up and down an application will not necessarily have a big impact on energy consumption. The reason is that most PaaS architectures have static provisioning: scaling down an application or a group of applications will not result in the switching off of physical servers. In the Cloud Foundry[7] PaaS environment for example, a certain number of VMs[8] able to host application containers are provisioned when the infrastructure is initially deployed, and does not change afterwards unless an operator redeploys the infrastructure manually. The applications are in turn embedded inside so-called *containers* hosted on the VMs, as can be seen in Figure 6.1. In this figure the red boxes represent the containers that run the apps. Those applications can be scaled up and down using the EASC. The blue boxes are VMs. Some of the VMs contains Cloud Foundry services (such as databases or HTTP servers). Other VMs are instrumented so that they can contain Linux Containers: such VMs are called DEAs in the Cloud Foundry context. Finally the grey boxes are physical nodes. This type of deployement is sometime called Containers as a Service (CaaS).

The energy management must then necessarily take place in the three layers:

[7] https://docs.cloudfoundry.org
[8] https://docs.cloudfoundry.org/concepts/architecture/execution-agent.html

Figure 6.1: IaaS-PaaS deployment

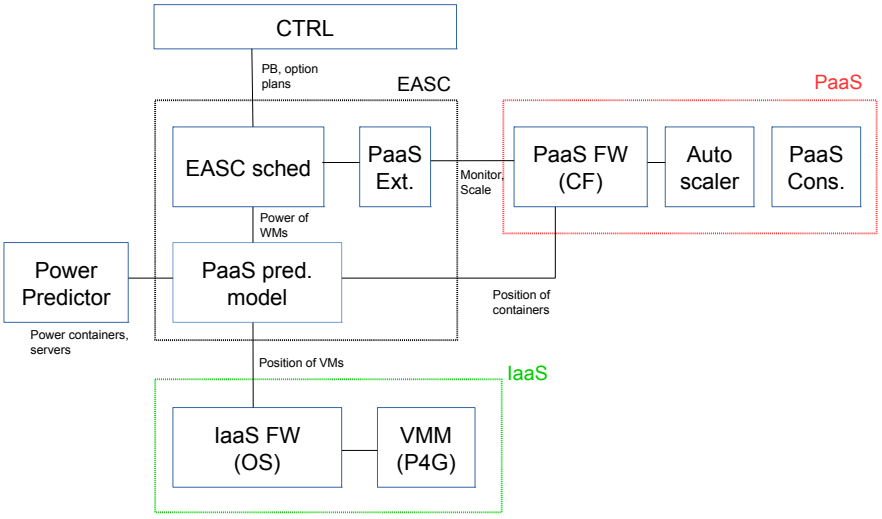


- *application layer:* The EASC will scaling up and down the number of containers owned by an application based on the renewable energy availability,
- *PaaS layer:* the containers must be consolidated inside the minimum number of VMs,
- *IaaS layer:* the VMs must be consolidated inside the minimum number of physical servers. The freed-up servers should then be switched off to save energy.

In the following, we show how to implement this reactive consolidation in each of the layers, together with the communication mechanims.

6.3.2 Architecture & implementation

Figure 6.2: EASC-PaaS architecture



The Figure 6.2 provides a high-level view of the EASC-PaaS interactions with its environment.

In this figure, the *EASC-Scheduler* module is a legacy scheduler for task and service oriented applications such as the one presented in Chapter 5. Based on this scheduling, the scaling of the application is performed through the *PaaS Extension* module, that connects to the PaaS infrastructure. We created en external component called the *PaaS Consolidator* able to perform the consolidation of the containers in the DEAs. This component is also in charge of augmenting or reducing the number of VMs as needed.

In practice, Cloud Foundry is used as the PaaS platform. The PaaS Consolidator is able to collect monitoring information through BOSH[9] and Cloud Foundry. BOSH is a tool used for the deployement of Cloud Foundry. If the resource usage reaches a configured threshold, the PaaS Consolidator will trigger BOSH in order to changes the CF deployment. This change will increase/decrease the number of deployed DEAs based on the new configuration. The changes applied to the infrastructure are deployed incrementally without service interruption. In addition, BOSH evacuates all applications installed on a DEA by migrating them to new DEAs before destroying it. In this way, we achieve both scaling and consolidation at container level.

In order to provide an energy efficient VM consolidation mechanism at IaaS level, we extended Plug4Green (presented in Chapter 4). The VM consolidation and server state management are achieved using live migration of VMs and powering off/on the servers. In practice, we selected the OpenStack platform to provide IaaS virtualization, together with block and object storage services. However, OpenStack does not provide any dynamic consolidation mechanism to provision/consolidate VMs based on the resource usage. Plug4Green was then extended to provide VM consolidation in OpenStack.

The final objective of this tool-chain is to make sure that the scheduling operations at application level performed by the EASC results effectively in energy consumption optimization. Finally, the EASC-PaaS contains a component called *PaaS Prediction model* able to predict the behaviour of the underlying infrastructure and to compute the power consumption of each working mode accordingly, so as to permit an accurate scheduling.

6.4 PaaS energy model

In order to allow an energy aware application scheduling algorithm to take sound decisions, we need a model to evaluate and predict the energy consumption of an application deployed in a PaaS setting. The algorithms described in Chapter 5 try to follow an input power curve

[9] https://bosh.io/

in order to maximize the renewable energy consumption. To perform this scheduling, such algorithm need to have an estimation of the power consumption of each application in the data centre. This will allow them to choose which activity to postpone in the data centre in order to match the input power curve.

However, estimating the power consumption of an application in a PaaS environment is not trivial because the architecture is shared between a great number of applications. Each application thus share a part of the infrastructure physical servers, and sometime a single server can host several different applications as can be seen on Figure 6.1. It is thus necessary to find a way to split the power of the infrastructure between the various applications in a fair yet simple way, allowing the energy aware scheduling algorithm to function correcly. The model presented is in two parts:

- Evaluation of the power consumed by the applications,
- Prediction of the power of an application scaling up/down.

The first model is concerned with finding an adequate splitting of the power of the data centre among the applications currently running. The second model is concerned with finding what will be the power of a specific application if this application scales up/down.

6.4.1 Evaluation of the power consumed by the applications

The objective of the model is to find a good approximation for P_{app}, the power of a certain application in the data centre. This power must be calculated using only measurable activity of the application and of the data centre infrastructure.

Model requirements and criteria

The study of the energy aware algorithms in Chapter 5 shows that the following criteria are necessary for a well behaved energy model.

- *Summation criteria:* the sum of the power yielded by the model for each application must be equal to the total power of the data centre ICT:

$$\sum_{i \in A} P_i = P_{totDC} \tag{6.1}$$

  where A is the set of all applications in the DC, P_i is the power of an application and

P_{totDC} is the total power of the data centre ICT, as can be measured by power meters.

- *Convergence criteria:* on a small infrastructure with few nodes and few applications, adding a new application will obviously change the splitting of the power between the applications. However when there are a lot of applications, the power yielded by the model for the contribution of each application to the total power idle should not vary any more with the number of applications. For example, in a big infrastructure, adding a new application should change only marginally the power calculated for the applications already deployed. We can derive this equation:

$$\forall i \in A, \lim_{card(A)\to\infty} P_i = \gamma \tag{6.2}$$

where $card(A)$ is the number of applications in the DC, i is an application and γ is a constant.

- *Monotonicity criteria:* the order of the applications sorted by their calculated power should not change when adding a new application in the infrastructure.

$$\forall i, j \in A, (P_i > P_j)_{S1} \Rightarrow (P_i > P_j)_{S2} \tag{6.3}$$

where $S1$ and $S2$ are two different states of the infrastructure, A in the set of all applications. $(P_i)_{S1}$ is the power consumption of application i in infrastructure state $S1$. This criteria is necessary in order to ensure that the underlying scheduling algorithm consistently prioritizes the same applications, even if the infrastructure state changes. For that, the ranking of existing applications should not change when new applications are added.

Infrastructure settings

We consider that the infrastructure consolidates applications and DEAs on the minimal number of servers. At PaaS level, the application containers are consolidated onto the minimum number of DEAs. Empty DEAs are removed. At IaaS level, the DEAs (which are simple VMs) are consolidated on a minimum amount of servers. The empty nodes are switched off.

Power distribution

We need to distribute the power of the nodes among the applications in a reasonable way. In this section we will consider that all nodes are equivalent. The node power is divided in two categories: *dynamic power* and *idle power*.

$$P_{totNode} = P_{dynNode} + P_{idleNode} \tag{6.4}$$

where $P_{idleNode}$ is the constant power idle of the node. The $P_{dynNode}$, on the other hand, is variable. According to several previous studies, for CPU intensive activities a linear model based on CPU levels provides a good approximation [42][35][76]. The authors shows that those high-level, "black box" power models sacrifice some accuracy in order to avoid relying on detailed knowledge of the hardware's implementation. The power measurements shows a high corelation between the CPU level and the power consumption.

$$P_{dynNode} = (P_{maxNode} - P_{idleNode}) * CPU_{node} \tag{6.5}$$

where $P_{maxNode}$ is the maximum power of the node. CPU_{node} is the CPU level of the node at a given time, expressed in percentage. Likewise, the power contribution of an application should be the sum of its contribution to the dynamic power and the idle power:

$$P_{totApp} = P_{dynApp} + P_{staticApp} \tag{6.6}$$

Dynamic power

The fraction of the dynamic power attributed to an application will be proportional to its measured CPU level. This choice is reasonable because the CPU is the main contributor to the dynamic power of a node :

$$P_{dynApp1} = \sum P_{dynNode} * \frac{CPU_{App1}}{\sum CPU_{Apps}} \tag{6.7}$$

where CPU_{App1} is the CPU level of the application and $\sum CPU_{Apps}$ is the sum of the CPU levels of all the applications.

Total power idle

In order to determine the total power idle $P_{totIdle}$ to be split among the applications in the data centre, we need to determine the number of nodes that are active:

$$P_{totIdle} = n * P_{idleNode} \tag{6.8}$$

where $P_{idleNode}$ is the power idle and n is the number of nodes necessary.

Having more applications in the data centre implies having to boot additional nodes. The decision to boot an additional node in the data centre is mainly based on the limitation of the RAM . We then need to estimate the number of nodes that are necessary taking into account their RAM size, the DEA RAM size and the applications container RAM size. The number of active nodes n can be calculated with a modified bin packing algorithm as presented in Chapter 3. Another technique is to retrieve the number of active nodes from the infrastructure.

Static power

Taking into account the considerations above, it is reasonable to distribute the contribution of each application to the total power idle in the following way:

$$P_{idleApp1} = P_{totIdle} * \frac{RAM_{App1}}{\sum RAM_{Apps}} \tag{6.9}$$

where RAM_{App1} is the total RAM used by an application and $\sum RAM_{Apps}$ is the total RAM used by all applications.

Summary

Using the previous equations, we can write:

$$P_{totApp} = P_{dynNode} * \frac{CPU_{App1}}{\sum CPU_{Apps}} + P_{idleNode} * \frac{RAM_{App1}}{\sum RAM_{Apps}} \tag{6.10}$$

We showed in the above that the power contribution of an application to the total power of the data centre ICT should be:

- proportional to its RAM footprint for the idle power sharing,

- proportional to its CPU footprint for the dynamic power sharing.

Model inputs

We can deduce from the previous equations the inputs of the model:

- The nodes available RAM,
- The nodes idle power,
- The nodes max power,
- The containers RAM demand,
- The containers CPU demand,
- The current location of the application containers,
- The current location of the DEAs.

The first three points can be retrieved from the nodes specification, the following three can be retrieved from the PaaS manager, while the last one can be retrieved from the IaaS manager.

6.4.2 Prediction of the power of an application scaling up/down

The objective of this section is to present the extension of the previous model with the ability to predict the power of an application under a workload change. In a PaaS setting, a significant workload change will usually trigger a scale up or down of the application. PaaS environments that are relying on containers will launch more containers for the same application, for example. The previous algorithm can be reused for the purpose of predicting the power of the application in its new configuration. In particular for equation 6.10, the RAM footprint of a container is usually known in advance, so it can be added to the application RAM total. However, the CPU usage of the new containers that will need to be launched need to be predicted. We propose to use statistical data gathered on the CPU usage of containers of the same type over a period of time to evaluate the CPU usage of the new container.

Limitations of the model

Predicting the power consumption of the scaling up/down of an application assumes that the rest of the applications running in the data centre will not scale up or down in the same time frame. If other application do scale up or down at the same moment, the number of

active nodes calculated by the model will of course be erroneous. However this effect is only transitory, as shown in Section 6.5, since the prediction is corrected as soon as the rest of the system is stabilized.

6.5 Experimentations and evaluation

In this section we evaluate the EASC-PaaS and the power prediction model. The experimentation is focused on the collection, elaboration, storage and finally presentation of healthcare data in the e-health context. The trial is hosted in the data centre of the healthcare agency of Trento, Italy.

During each day the load of the data centre is about 3000 exams to be collected, elaborated and stored, and about 20000 reports presented, after a further elaboration, to the users/doctors. The load pattern is distributed over a day with high activity in the morning, medium activity in the afternoon and very low activity during evening and night.

6.5.1 Hardware infrastructure

The hardware infrastructure is composed of 6 servers, table 6.1 presents their specification. The state (on/off) of the servers are managed by Plug4Green as mentioned in section 6.5.2. During low load condition, the system is able to consolidate the number of servers down to two (OS controller and Compute 8). On the other hand, during periods of high load, the infrastructure is using all the servers.

Table 6.1: Hardware information

Server name	Role	vCPU (#)	RAM (GB)	Power Idle (W)	Power Max (W)
OS controller	Controller	4	10	465	530
Compute 8	Compute	8	32	132	238
Compute 9	Compute	4	16	430	529
Compute 11	Compute	8	16	436	606
Compute 12	Compute	16	32	607	902
Compute 13	Compute	8	16	465	634

6.5.2 Cloud infrastructure and technologies

The infrastructure of Cloud Foundry is deployed on top of OpenStack. It uses 14 VMs for the various CF services and 1 VM for the BOSH Director, while the number of DEAs can vary from

1 to 8. Those DEAs can be scaled up/down by the PaaS Consolidator specifically. Table 6.2 is giving the characteristics of PaaS components. Disk space is ignored because it is not a limiting factor. From this table it is visible that each DEA can host up to 4 containers, while each Compute node can host from one up to four DEAs.

Table 6.2: IaaS and PaaS infrastructure information

	vCPUs	RAM (GB)	Idle Power (W)
DEA	4	6	3.2
Container	1	1.5	neg.

The deployment includes CloudFoundry version 214[10] on top of Openstack installed with Fuel Juno 6.0[11]. OpenStack provides a virtualization platform, together with block and object storage services. The VM consolidation and server state management are achieved using live migration of VMs and powering off/on the servers.

6.5.3 Application scenario

The application selected is a three-tier application composed of the following elements:

- front-end (FE): a web server able to collect and present the medical exam images from doctors,
- back-end (BE): a CPU-intensive task able to elaborate the images received,
- a database able to store the data to be processed and the results when ready.

In addition, the scenario considers two SLA requirements: BE should elaborate all the exams stored by FE until the next midnight, and FE should respect the load pattern during a day time period.

The whole application is controlled by an EASC, and each tier, i.e. FE and BE, are implemented as separate activities of that application. The BE is a task oriented application: the elaboration of each exam is identified as a separate task. This task can be postponed during the day, however it need to be processed before midnight as per the SLA. On the other hand, the FE is a service oriented application. The web server need to be running at all time, while making sure to match the minimal performance required by the SLA at any given time.

[10] https://bosh.io/releases/github.com/cloudfoundry/cf-release?version=214
[11] https://www.mirantis.com/products/mirantis-openstack-software/openstack-deployment-fuel/

Both BE and FE tiers are deployed as separate Cloud Foundry containers, with the flexibility to increase/decrease the instances of each layers independently and easily using the Cloud Foundry interface. As a consequence EASC is able to adjust the number of instances using CF APIs not only based on the renewable energy availability, but also to respect the SLA/SLO of each single application layer.

6.5.4 Energy mix

The trial has a single power source from the Italian national grid. We followed a Measurement & Verification (M&V) methodology[12] and built six profile that reflects the energy mix of typical days in various seasons throughout a year in Italy. This methodology allowed us to simulate an entire year trial run with just 6 different profiles, one per day. Table 6.3 shows the mapping of each profiles on the full year.

Table 6.3: Profiles mapping to a full year

Month	Normal		Exceptional	
	Profile	Number of days	Profile	Number of days
January	Profile 1	29	Profile 6	2
February	Profile 1	23	Profile 6	5
March	Profile 1	23	Profile 6	8
April	Profile 3	28	Profile 4	2
May	Profile 2	30	Profile 5	1
June	Profile 2	23	Profile 5	7
July	Profile 3	23	Profile 4	8
August	Profile 3	19	Profile 4	12
September	Profile 1	24	Profile 6	6
October	Profile 1	26	Profile 6	5
November	Profile 1	26	Profile 6	4
December	Profile 1	21	Profile 6	10

6.5.5 Evaluation

The Figure 6.3 shows the power consumed by the infrastructure during the 6 days trial. The renewable percentage curve (in green) is super-impressed so as to show that the peaks of consumption of the trial has been aligned as much as possible with the corresponding maximum availability of renewable energies. However only task oriented activities that presented shiftable workload could be aligned on the maximum renewable percentage. Service oriented activities, on the contrary, were not necessarily aligned with the maximum renewable

[12] http://www.evo-world.org/en/

percentage.

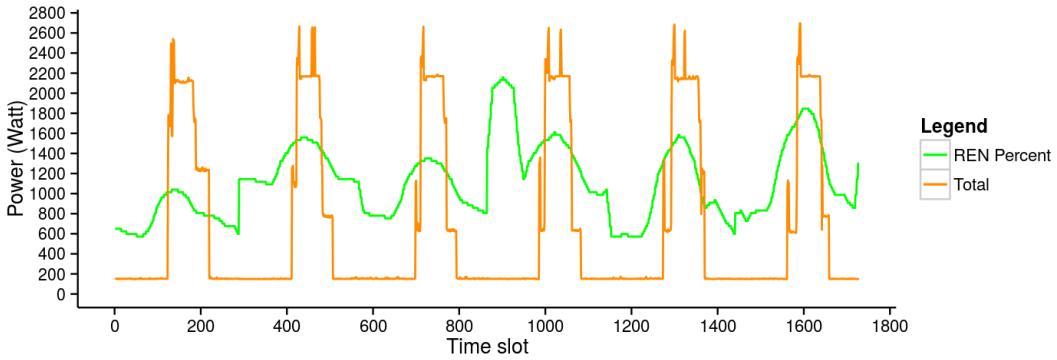


Figure 6.3: six days trial overview

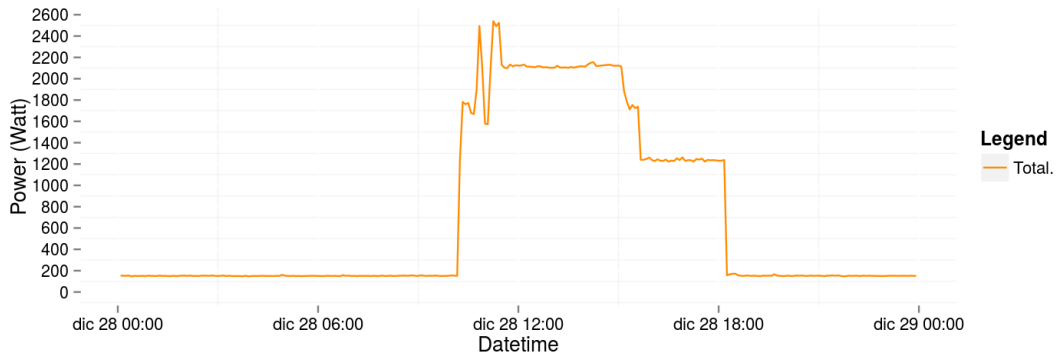
Table 6.4: Trial results

Profile	**RenPercent**	**RenPercent baseline**
Profile 1	35.37%	30.19%
Profile 2	55.75%	49.24%
Profile 3	47.24%	39.41%
Profile 4	61.28%	47.00%
Profile 5	58.02%	56.49%
Profile 6	50.19%	37.68%

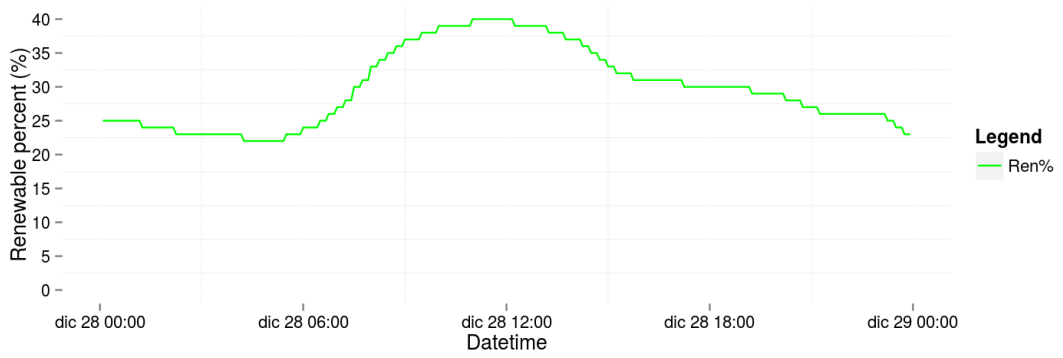
The Table 6.4 presents the improvements in the percentage of renewable energy consumed, for each profile of the trial. Using our M&V methodology, we can map each profile to a number of days during the year. This allows to compute an improvement on renewable consumption of 7.07 percentage points for the full year.

The Figure 6.4 presents the renewable percentage and the total power consumption of the trial, during the first day of trial (using profile 1). It shows how the trial power consumption becomes adaptive to renewable energy availability. We observe that the workload has been scheduled during high RenPercent periods, considering however that front-end SLA should be met. The minimum power is around 150 Watt, corresponding to minimum working mode of front-end; this workload cannot be shifted because it is a service-oriented activity that should be available 24 hours.

The Figure 6.5 presents the total actual power, planned power, and the actual power of each activity during the first day profile. The figure shows how the power consumption is distributed



(a)



(b)

Figure 6.4: Adaptation to renewable energies

between the two activities according to their resource consumption at any point in time. This effect is particularly visible when back-end activity has no load, and therefore all power is allocated to the front-end activity. However, when the activity of the back-end increases, the power distribution changes and therefore the power allocated to the front-end decreases. In addition, the graphic shows that the planned power is reasonably close to the total actual power. The mean absolute percentage error between the planned power and the actual power measured in the system over the 6 days of trial is 16,58%.

The Figure 6.6 shows the PaaS and IaaS orchestration during the trial run. This figure shows that when front-end scales up more containers (red curve) in order to meet front-end load, the number of PaaS runners VMs (green curve) increases to host front-end containers, and consequently IaaS powers on more servers (dark orange curve) in order to host PaaS's runners VMs. Similarly, this happens when back-end scales up (blue curve). In addition, when back-end or front-end scales down, the PaaS scales down the number of PaaS's runners VMs, and IaaS powers off some servers.

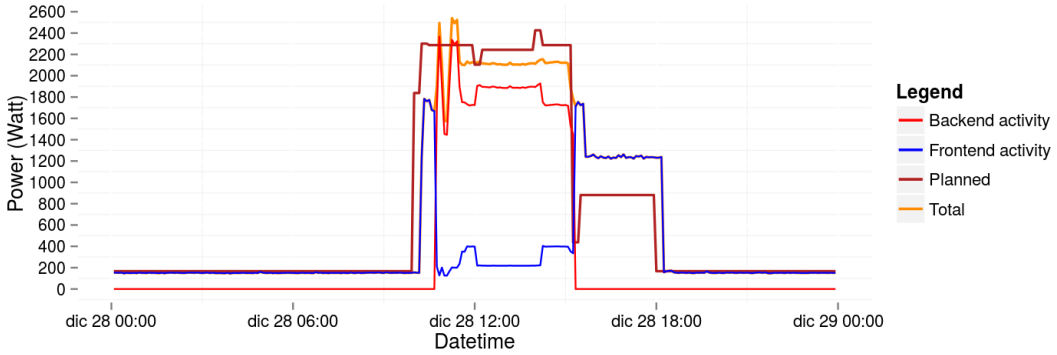


Figure 6.5: Power of activities

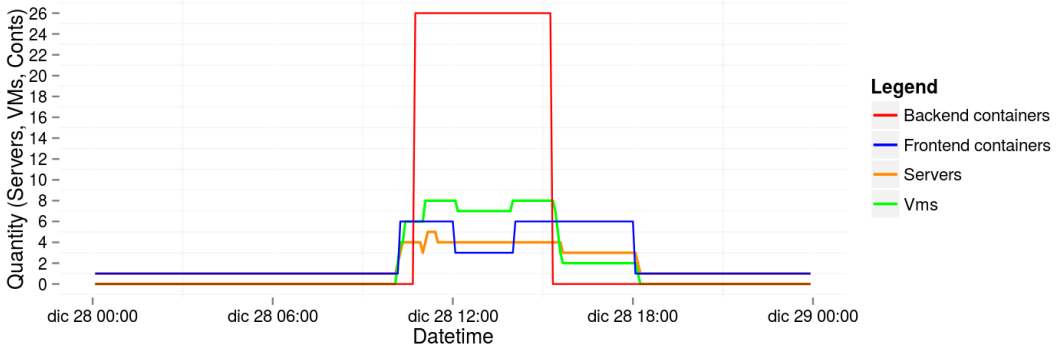


Figure 6.6: PaaS-IaaS Cloud infrastructure orchestration

Figure 6.7 presents the front-end actual business performance compared to its SLO. The Figure shows that the front-end business performance follows quite exactly the SLO, with however a small delay corresponding to the time to scale up or down the number of containers.

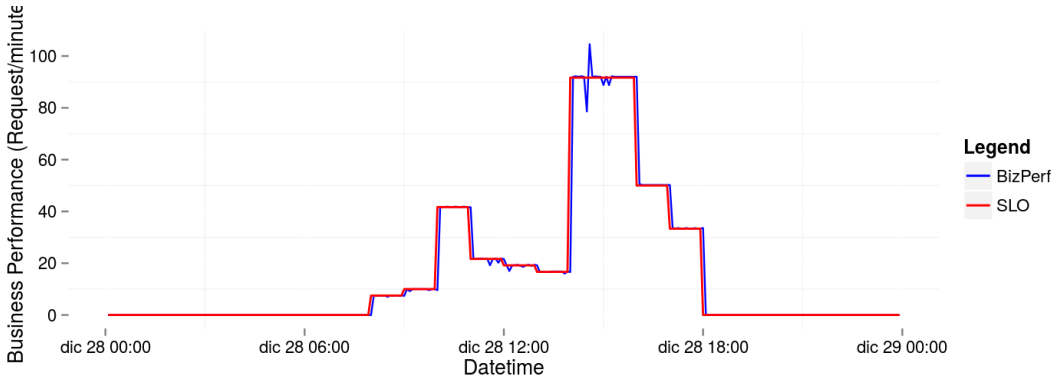


Figure 6.7: Front-end following the SLA

6.5.6 Scaling process analysis

The full scale up process, when scaling from 0 to 13 containers, takes 24 minutes. This number includes the consolidation of the VMs at IaaS level. When the command for scaling up the number of containers is launched, a first container is launched on the only available DEA. The PaaS Consolidator will then detect that more DEAs need to be created, and will reconfigure BOSH in order to do so. We configured BOSH to instantiate the VMs in parallel, four at a time. This setting allowed to optimize the DEA scale up time: four at a time was the optimal number, allowing to use the available nodes in parallel. Instantiating the VMs is done by triggering OpenStack. The VMs were up an running 6 minutes after being instantiated.

Once the VMs are ready, the longest process is to configure them. This configuration, done by BOSH, takes up to 10 minutes on our infrastructure. This process consists in the installation of DEA software inside the VMs launched by OpenStack. After the DEAs are ready, instantiating the containers inside them is fast: from 55 seconds to 1 minute 50 seconds when the system is more loaded.

After the PaaS scaling is finished, the ultimate step is to trigger the re-consolidation of the IaaS infrastructure. For this, Plug4Green migrates the DEAs created in order to consolidate them on the same server. This process took 4 minutes on our infrastructure.

This scale up process could be shortened using more recent hardware. Secondly, using VM images already containing the DEA software would allow faster instantiation.

The scale down process, on the other hand, is much faster. Scaling down from 13 containers to zero took 4 minutes 18 seconds. This is due to the fact that destroying VMs take a very small amount of time. Furthermore, on an empty infrastructure, Plug4Green doesn't need to consolidate the VMs.

6.6 Conclusion and Future Work

Using the renewable energies to power data centres is challenging. This is due to both the intermittent nature of the renewable sources and the complex nature of data centres.

In this chapter, we expend on the notion of Energy Adaptive Software Controller (EASC), using the opportunities offered by the PaaS paradigm when it comes to energy management. We showed how a PaaS infrastructure, itself based on a IaaS, can be used to respect energy constraints. The complete scaling operations happens at three different levels: application,

PaaS containers and finally IaaS VMs. We presented an energy model able to evaluate simply the energy consumption of application within a shared infrastructure such as PaaS.

We validated the notion of EASC through the complete implementation of a legacy applications running in the PaaS layer. The energy model was used in order to make the decisions on the scaling operations. Experiments over a week showed that the EASC was able to increase the usage of renewable energies by aligning the application performance with the energy availability periods.

7 Conclusions and future directions

This section summarizes the research exposed in this thesis and discuses the possible future directions.

7.1 Conclusions

Data centres are power-hungry facilities that host ICT services and consumed from 1.1% to 1.5% of the global electricity consumption, as of 2010 [14]. Consequently, in the last years, research trends in the field focused on mechanisms able to reduce the overall consumption of a data centre so as to reduce its energy footprint. Moreover, new interests toward aspects such as green footprint demands for more than just reducing energy consumption. Aiming at reducing energy consumption in data centres becomes part of a larger equation: being able to consume energy in a better way, prioritizing the consumption of green energies. This thesis presented several energy management techniques in data centres. The first goal, reducing the energy consumption in data centres, has an obvious ecological and economical impact. However, over the course of this research, a secondary goal appeared: to prioritize renewable energies over brown energies. The renewable energy is drawn from a local power source or from the grid.

Regarding the energy saving goal, we presented a preliminary work on heuristics for VM consolidation. This work on the heuristics showed us that, although they are usable in some specific cases, they are not generic enough to cope with the constant evolution of data centres. In order to address this flexibility need, we proposed Plug4Green, an energy-aware VM manager. Thanks to Constraint Programming, Plug4Green can be easily specialized to support various combinations of SLAs, power models and energy policies. Its flexibility has been verified

through the implementation of 23 meaningful SLAs and 2 energy policies. Its practical effectiveness has been evaluated on an industrial testbed. While the default version of Plug4Green reduced the power consumption and the gas emission by 27% and 23% respectively, its specialization to fit the hardware heterogeneity improved the saving by up to 34%. Finally, scalability experiments on simulated data have shown that Plug4Green is able to compute an improved placement for 7,500 VMs running on 1,500 servers in a minute, while respecting their SLA.

We also presented the notion of Energy Adaptive Software Controller (EASC), a generic software controller that developers can use to make their application adaptive to renewable energy availability. To integrate the notion of EASC into a legacy application, a developer only needs to identify its various KPIs, working modes and to declare the commands to use to enact a working mode. The EASC then provides different scheduling algorithms to continuously choose the most appropriate working mode to use for the controlled application with regards to its SLAs and power budget. We showed how a PaaS infrastructure, itself based on a IaaS, can be used to respect energy constraints. The complete scaling operations happens at three different levels: application, PaaS containers and finally IaaS VMs. We presented an energy model able to evaluate simply the energy consumption of applications within a shared infrastructure such as PaaS. We validated the portability of the notion of EASC through the complete implementation of two legacy applications. The first one is a task-oriented application used to generate reports in the healthcare domain. The second one is HP Life, an international eLearning lab available through a Web application. We also confirmed the practical benefits of the developed EASCs on two different testbeds: one powered only by the Italian national power grid, and the other through a dual power sources using photovoltaic arrays and national power grid. Experiments over a week also proved that the EASCs increased the usage of renewable energies by aligning the application performance with the energy availability periods. Results showed that the EASC was able to augment the renewable energy usage percentage by 7.07pp in the Trento trial with a PaaS infrastructure.

7.2 Discussion

In this section we put into perspective the main findings of the research presented in this thesis.

7.2.1 Heuristics vs meta-heuristics

The most efficient technique to save energy in data centres so far has definitely been the consolidations possibilities offered by virtualization techniques. we explored various technologies in order to consolidate the VMs, among them Constraint Programming. This meta-heuristic offers clear advantages regarding the flexibility and evolvability of the solver. However, we noticed two drawbacks: first, it adds a significant overhead in terms of engineering. The qualifications required in term of software engineering are relatively high, in domains such as modelization and operational research. This limits the impact in term of adoption of the technology. One solution could be the automatic generation of constraints and test cases using a specification language. Such a specification language is usually closer to the end user needs. Secondly, the solving duration is relatively high. For example, it took around 3 minutes to solve the problem with 2000 nodes with Plug4Green (see section 4.5). This prevents to run the solver too often. Empirically, we found that a good delay between consolidations is 10-15 minutes.

As hinted in chapter 3, VM consolidation techniques are working, but they should be applied with care. Consolidation of cache-sensitive and storage-intensive VMs is likely to lead to severely degraded performance. However, heuristics that have simplistic methods of treating multi-dimensional resource requirements are reasonably effective in many common practical situations. In complex situations, we showed that using a meta-heuristic such as Constraint Programming allows to take into account much more particular cases.

7.2.2 Analytical power models vs black-box power models

Analytical power models, such as the one presented in Chapter 4, can be very accurate. Their error rate is usually about 5% [49]. However our experience showed that they can prove very difficult to configure. A lot of different components in the data centre are modelized and should be configured according to data sheets or real time metering, such as CPU voltage or number of transistors. Furthermore, they become obsolete very fast because the hardware evolves and is replaced regularly within data centres. This is why, in recent versions of Plug4Green, the analytical model has been replaced with a simpler, linear model similar to the one presented in Chapter 6. These kind of models are less accurate: our evaluation showed a mean percentage error of 16.58% between the planned power and the actual power consumed in the data centre. However, the advantage is that they depend on very few configuration and real-time parameters, such as power idle and CPU levels. In the case of our scheduling algorithms, this proved to be enough to guarantee a reasonable scheduling, as shown by the results obtained

in term of renewable energy percentage improvement.

7.2.3 Technology transfer

Regarding the integration of our solutions into real Cloud solutions, we witness also a certain gap. While some ideas from the research community gets adopted over time in Open Source or commercial solutions, most of them are not. A product like OpenStack, for example, still doesn't support VM consolidation by default. A problem of the proposed solution is their complexity. Some complex algorithms sometimes find commercial applications: for example Google search engine embeds a very complex algorithm. However the visible part of it should be extremely simple, in the case of Google the user interface is a simple string. In our case the problem is that the complexity of our models leaks on the end-users. For example the Constraint Programming model defines a big quantity of new concepts that the user needs to learn. This complexity also makes the system more fragile and prone to corner cases. The scheduling solutions calculated by system, while maybe correct, can be also prove to be difficult to interpret by the end-user due to the high quantity of interacting parameters. This, in our opinion, is the main parameter holding back the adoption of complex algorithms for energy aware systems. Very simple algorithms, such as round robin, are sometime preferred over more efficient algorithms, for the sake of simplicity and predictability.

The new project Watcher[1] is an effort for integrating a resource optimization service in OpenStack. This optimization includes the reduction of data centre operating costs, increased system performance and increased energy efficiency. Plug4Green concepts and technology are currently being discussed within the Watcher community for integration.

7.2.4 Renewable energies adoption

This thesis showed several techniques aimed at increasing the renewable energy usage in data centres. We showed that, with an adequate scheduling, it is possible to increase the renewable percentage used in the data centre, even if only the grid power is available (no local generation). However a major problem, albeit out of scope of this thesis, is to incentivize the data centre owners to deploy such techniques. A system of bonuses should be put in place at the city or national level, in order to reward data centre owners that optimize their usage of the renewable energies. Moreover, data centre owners can put in place a billing policy that rewards application owners that implements flexibility mechanisms such as the EASC.

[1] https://wiki.openstack.org/wiki/Watcher

7.3 Future research directions

7.3.1 Usage of energy accumulators in DCs

Energy accumulators such as batteries are currently used in data centres mainly for emergency energy backups. Accumulating energy for optimizing renewable usage appears to be a good idea. However at the time when this thesis was started, energy accumulators were not deemed mature enough. Batteries are still expensive, and acquiring enough batteries to power a full DC for a significant number of hours every day represents a big capacity. Furthermore batteries manufacturing, maintenance and disposal have a significant economic and ecological impact. However the recent advances in battery conception, mainly due to the automotive market research, may have changed the deal. Indeed batteries are getting cheaper and cheaper, allowing to use more of them. Other kinds of energy accumulator could be used, such as pumped storage hydroelectricity systems.

This evolution could have a major impact on this research topic. The accumulators can store energy and deliver it latter, however they have an inherently limited capacity. Furthermore, the batteries need maintenance time, when they are remove and replaced. A number of areas of this research need to be updated consequently. First of all, the algorithm need to take into account that the renewable energy percentage present in the energy source mix might need to be adjusted when using the batteries. Indeed the batteries allows to change the renewable energy percentage used in the data centre at a given point in time by delivering energy stored before. Secondly, the algorithm needs to take into account the possibility to command and control the batteries. This control need to be coordinated with the availability of renewable energy, but also with the status of the applications controlled, in order to obtain an optimum usage of energy and performance of applications. The correct modelization of the battery behavior when charging or discharging is also an important challenge.

7.3.2 Energy management with Unikernels

This thesis showed the usage of virtualization to manage energy. We used two different supports for virtualization: virtual machines (running in hypervisors) and Linux containers. A third virtualization vehicle is currently emerging: the Unikernel [77].

In the current IaaS implementations, several VMs are running on top of an hypervisor, as shown in Figure 7.1. Those VMs embed a complete Operating System (in grey in the figure, for example Windows or Linux). However, a lot of features of the host OS appear to be redundant

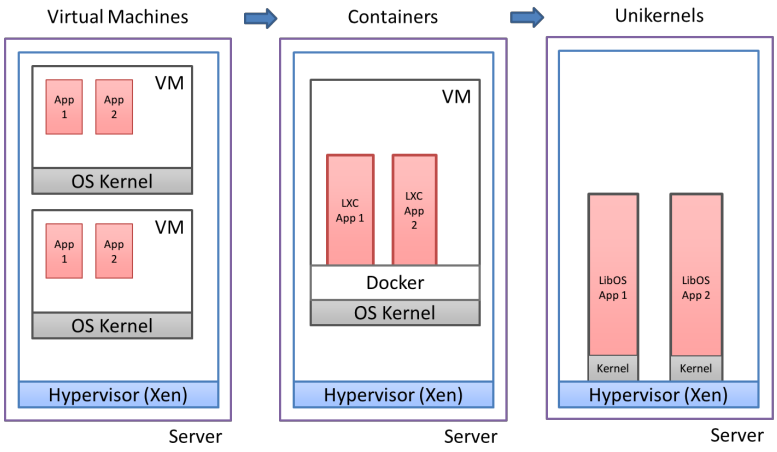


Figure 7.1: Evolution of virtualization vehicules: VMs, containers, Unikernels

with the hypervisor: the hypervisor is already providing isolation between application and resource scheduling between the various VMs. Furthermore, each VM on a server embeds its own kernel (the core of the OS), and a modern server can host up to 50 VMs. Container technologies, used in this thesis, allows to remove a part of those redundancies. Containers use OS APIs to create separate environments, each with their own file system, memory space and processes. This means that the amount of replicated resources for each container on the same server is greatly reduced with respect to virtual machines, which implies smaller footprint, higher per-host density and faster start times.

The next, ultimate step in this progression is the Unikernel. A Unikernel is an application compiled together with a so-called "Library OS". The result of this compilation is not an executable than should run on top of an OS: it is itself an OS. The compiled Unikernel runs directly on Xen[2]. As such, Unikernels can be viewed as single-purpose bare-bone VMs. The advantages of this technology are multiple:

- Extremely reduced size (a Unikernel can be in the order of magnitude of 10Mo, where a VM is 1Go), due to the elimination of many redundancies.
- Fast boot-up speed: the Unikernel being single-purpose, it has a very reduced kernel with no useless drivers.
- Security: a reduced attack surface, due to its small size and elimination of useless drivers.
- Reliability: Unikernels are traditionally based on the functional programming paradigm, which is known to produce more reliable programs.

[2] http://www.xenproject.org

Unikernels are still a new technology. However their characteristics could have a significant impact on the research domain of this thesis. The reduced granularity (low RAM space, low start up time) of the Unikernel suggests the necessity to develop new scheduling algorithms. Their positioning can be more dynamic than for VMs (which have a high start up time). Migrating them between two different sites, for the sake of energy management, can also be faster. Furthermore, Unikernels might find a development avenue with the rise of the micro-service architectures [78]. In micro-service architectures, each component of an application is located in a separate VM/Container/Unikernel and exposes an API to the other components. This implies a high number of VMs that share specific relations with each others. Those relations should be modelized and taken into account in our future research.

7.3.3 Warm data centres

° Since the end of the silicon road, the CPU clock frequency of manufactured computers remained relatively stable. However, huge progresses have been made in the power management of CPUs, allowing them to run at much higher temperatures. Data centres typically operate in a temperature range between 20°C and 22°C, some are as cold as 13°C degrees. However recent studies suggests a much higher temperature. Some investigations, such as in [79], suggests that increasing the room temperature set point in data centres by just one degree could save 2–5% of the energy consumption. How does our research is affected by this evolution? The PUE is the ratio between the power spent in IT equipment divided by the total energy spent in the data centre, including the cooling systems. Raising the temperature set point in the DC will then mechanically lower the PUE. In an environment with low PUE, energy saving techniques then have a lower final impact. On the other hand, a high data centre temperature render the problems of "hot spots" more critical. It then makes sense to include a modelization of the hot spots of the data centre in the VM placement model and in the scheduling system, such as in [80].

7.3.4 Service migrations & edge computing

With the development of the Internet of Things, a lot more objects and sensors will get connected to Internet. Those objects usually have limited computational power, has well as limited energy resources. This is why they use the computing power of the Cloud in order to process CPU intensive tasks. However, moving some key computations in the opposite direction, from the Cloud to the device, also makes sense. For example, if a temperature sensor is only used by an application that need an average temperature value computed every

hour, it makes sense to relocate the algorithm that performs the averaging on the device itself. This allows to lower the transmission rate and to save energy. Other tasks, such as data anonymization and encryption, need to be performed on the device before being sent to the Cloud. However, the full virtualization of most IoT devices is not achieved yet. The Raspberry PI, that can be viewed as a rather high-end IoT device platform, has only recently received attention in this direction [3]. The virtualization of the "edge" devices will allow to migrate virtual machines directly on them, for example for energy reasons. This development could have an impact on the VM positioning algorithms presented in this thesis.

[3] http://blog.flexvdi.es/?p=139

A Implementation of the classical VM packing problem with SMT/Haskell

To show the usability of both SMT and pure functional languages to tackle energy efficiency problems in a flexible way, we implemented the classical problem of packing VMs on servers using the library SBV[1], with only one dimension for the sake of simplicity. In the example[2] showed in Listing A.1, each VM has a demand in term of CPU, and each server has a certain CPU capacity to offer. The objective is to find the placement of the VMs on the servers that minimizes the number of servers needed. The only constraint applied is that the total CPU consumption of the VMs that will be running on a server must not exceed the capacity of that server.

Listing A.1: Example of VM placement problem solved using SMT

```
1
2  --concrete IDs for VMs and servers
3  type VMID = Integer
4  type SID = Integer
5
6  --symbolic IDs of the servers
7  type SSID = SBV SID
8
9  --A VM is just a name and a cpuDemand
10 data VM = VM { vmName :: String,
11                cpuDemand :: Integer}
12
13 --a server has got a name and a certain amount of free CPU
14 data Server = Server { serverName :: String,
15                        cpuCapacity :: Integer}
16
17 --list of VMs
18 vms :: Map VMID VM
```

[1] http://leventerkok.github.io/sbv

[2] The full implementation can be seen at https://github.com/cdupont/Plug4Green-design

```
19 vms = fromList $ zip [0..] [VM "VM1" 100, VM "VM2" 50, VM "VM3" 15]
20
21 --list of servers
22 servers :: Map SID Server
23 servers = fromList $ zip [0..] [Server "Server1" 100, Server "Server2" 100,
     Server "Server3" 200]
24
25 --number of servers ON (which we'll try to minimize)
26 numberServersOn :: Map VMID SSID -> SInteger
27 numberServersOn = count . elems . M.map (./= 0) . vmCounts
28
29 --computes the number of VMs on each servers
30 vmCounts :: Map VMID SSID -> Map SID SInteger
31 vmCounts vmls = M.mapWithKey count servers where
32    count sid _ = sum [ite (mysid .== literal sid) 1 0 | mysid <- elems vmls]
33
34 --All the CPU constraints
35 cpuConstraints :: Map VMID SSID -> SBool
36 cpuConstraints vmls = bAnd $ elems $ M.mapWithKey criteria (serverCPUHeights vmls
     ) where
37    criteria :: SID -> SInteger -> SBool
38    criteria sid height = (literal $ cpuCapacity $ fromJust $ M.lookup sid servers
        ) .> height
39
40 --computes the CPU consummed by the VMs on each servers
41 serverCPUHeights :: Map VMID SSID -> Map SID SInteger
42 serverCPUHeights vmls = M.mapWithKey sumVMsHeights servers where
43    sumVMsHeights :: SID -> Server -> SInteger
44    sumVMsHeights sid _ = sum [ite (sid' .== literal sid) (literal $ cpuDemand $
        fromJust $ M.lookup vmid vms) 0 | (vmid, sid') <- M.assocs vmls]
45
46 --solves the VM placement problem
47 vmPlacementProblem :: IO (Maybe (Map VMID SID))
48 vmPlacementProblem = minimize' numberServersOn cpuConstraints
49
50 main = do
51    s <- vmPlacementProblem
52    putStrLn $ show s
```

When run, this program returns the placement for the VMs that minimizes the number of necessary servers. In this case, it will place all three VMs on the third server. While it is difficult to compare, it is anyway striking that this program is shorter than its equivalent in Java/Choco[3]. The definition of a constraint takes only a few lines (for example $numberServersOn$ takes 2 lines) and flows with the program definition. Furthermore, as it is usually the case in Haskell, the type signature of the functions are carrying a lot of information that can be used both

[3] for example this implementation of bin packing: http://www.dcs.gla.ac.uk/ pat/cpM/jchoco/binPack/CPBinPack.java

by the programmer to understand and reason about the program, and by the compiler to prove its correctness. For example, the type signature *numberServersOn :: Map VMID SSID -> SInteger* makes it clear that the function $numberServersOn$ is a constraint that takes the positions of all the VMs on the servers (denoted as a mapping between the VM ids and the server symbolic ids) and returns a symbolic integer representing the necessary number of servers.

Furthermore, programming at the symbolic level, as it is required when designing a CSP, is not very different than programming in concrete Haskell. This is because a lot of the Haskell standard functions, like the function sum in our example program, can be reused in a constraint programming program. The definition of sum in the standard library of Haskell is generic enough to be able to be used also at the symbolic level. On the other hand, programming in Choco is completely different than programming in concrete Java: all the operators are necessarily different, due to the low genericity of Java. Therefore, the intuition of the Java programmer cannot be completely reused.

SBV is also a theorem prover, and that can be used to prove properties of the constraints expressed. For example, we might want to prove some properties about our constraint $vmCounts$. This function counts the number of VMs present on each servers. We want to prove the property that the count of VMs on a server has for absolute maximum the total numbers of VMs present in the data centre.

Listing A.2: Example of proof about a constraint

```
1
2 *Main> prove $ \x y -> bAll (.<= 2) $ vmCounts' [x, y]
3 Q.E.D.
```

The listing A.2 show how we can ask SBV to prove that the number of VMs per server computed by the constraint $vmCounts$ cannot exceed the total number of VMs (in this simplified example with only 2 VMs and a version of vmCounts defined for lists instead of maps). SBV simply replies with $Q.E.D$, showing that it found a proof of our property (this proof can be exhibited if needed).

Bibliography

[1] Sonja Klingert, Florian Niedermeier, Corentin Dupont, Giovanni Giuliani, Thomas Schulze, and Hermann de Meer. *Introducing Flexibility into Data Centers for Smart Cities*. Communications in Computer and Information Science, 2016.

[2] Corentin Dupont, Mehdi Sheikhalishahi, Federico M. Facca, and Fabien Hermenier. An energy aware application controller for optimizing renewable energy consumption in cloud computing data centres. In *8th IEEE/ACM International Conference on Utility and Cloud Computing*, 2015.

[3] Corentin Dupont and Fabien Hermenier. DC4Cities: Better usage of the renewable energies in data centres. In *ICT4S 2015*, 2015.

[4] Corentin Dupont, Mehdi Sheikhalishahi, Federico M. Facca, and Silvio Cretti. Energy efficient data centres within smart cities: Iaas and paas optimizations. In *2015 EAI International Conference on Smart Grids for Smart Cities*, Toronto, Canada, 2015.

[5] Sonja Klingert, Florian Niedermeier, Corentin Dupont, Giovanni Giuliani, Thomas Schulze, and Hermann de Meer. Renewable energy-aware data centre operations for smart cities - the DC4Cities approach. In *SMARTGREENS 2015*. ACM, 2015.

[6] Corentin Dupont, Fabien Hermenier, Thomas Schulze, Robert Basmadjian, Andrey Somov, and Giovanni Giuliani. Plug4green: A flexible energy-aware vm manager to fit data centre particularities. *Ad Hoc Networks*, pages 505–519, 2014.

[7] Corentin Dupont. Building application profiles to allow a better usage of the renewable energies in data centres. In *Energy-Efficient Data Centers*, Lecture Notes in Computer Science, 2014.

[8] Corentin Dupont. Energy aware infrastructure for green cloud data centres. University of Trento Doctoral School, 2014.

[9] Corentin Dupont. Renewable energy aware data centres: The problem of controlling the applications workload. In Sonja Klingert, Xavier Hesselbach-Serra, MariaPerez Ortega, and Giovanni Giuliani, editors, *Energy-Efficient Data Centers*, volume 8343 of *Lecture Notes in Computer Science*, pages 16–24. Springer Berlin Heidelberg, 2013.

[10] Corentin Dupont, Thomas Schulze, Giovanni Giuliani, Andrey Somov, and Fabien Hermenier. An energy aware framework for virtual machine placement in cloud federated data centres. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, pages 4:1–4:10. ACM, 2012.

[11] Dang Minh Quan, Robert Basmadjian, Hermann de Meer, Ricardo Lent, Toktam Mahmoodi, Domenico Sannelli, Federico Mezza, Luigi Telesca, and Corenten Dupont. Energy efficient resource allocation strategy for cloud data centres. In Erol Gelenbe, Ricardo Lent, and Georgia Sakellari, editors, *Computer and Information Sciences II*, pages 133–141. Springer London, 2012.

[12] Dang Minh Quan, Andrey Somov, and Corentin Dupont. Energy usage and carbon emission optimization mechanism for federated data centers. In *Proceedings of the First International Conference on Energy Efficient Data Centers*, E2DC'12, pages 129–140, 2012.

[13] Corentin Dupont, Mehdi Sheikhalishahi, and Michele Santuari. Improving renewable energy consumption in iaas/paas hybrid data centres. Submitted to Futur Generation Computer Systems, 2016.

[14] Jonathan Koomey. Growth in data center electricity use 2005 to 2010. *Oakland, CA: Analytics Press. August*, 1:2010, 2011.

[15] Íñigo Goiri, William Katsak, Kien Le, Thu D. Nguyen, and Ricardo Bianchini. Parasol and greenswitch: Managing datacenters powered by renewable energy. *SIGPLAN Not.*, 48(4):51–64, March 2013.

[16] Cheikhou Thiam. *Anti Load-Balancing for Energy-Aware Distributed Scheduling of Virtual Machines*. Thèse de doctorat, Université de Toulouse, Toulouse, France, juillet 2014. (Soutenance le 03/07/2014).

[17] Anton Beloglazov. *Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing*. Doctorate thesis, University of Melbourne, 2013.

[18] D. Bradley, R. Harper, and S. Hunter. Workload based power management for parallel computer systems. *IBM*, 47:703–718, 2003.

[19] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: Eliminating server idle power. *SIGARCH Comput. Archit. News*, 37(1):205–216, March 2009.

[20] Donato Barbagallo, Elisabetta Di Nitto, Daniel J. Dubois, and Raffaela Mirandola. A bio-inspired algorithm for energy optimization in a self-organizing data center. In *Proceedings of the First International Conference on Self-organizing Architectures*, SOAR'09, pages 127–151, Berlin, Heidelberg, 2010. Springer-Verlag.

[21] Ray Carroll, Sasitharan Balasubramaniam, Dmitri Botvich, and Willie Donnelly. Dynamic optimization solution for green service migration in data centres. In *ICC*, pages 1–6. IEEE, 2011.

[22] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 215–224, New York, NY, USA, 2010. ACM.

[23] Qinghui Tang, Sandeep Kumar S. Gupta, and Georgios Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Trans. Parallel Distrib. Syst.*, 19(11):1458–1472, November 2008.

[24] Kien Le, Ricardo Bianchini, Jingru Zhang, Yogesh Jaluria, Jiandong Meng, and Thu D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 22:1–22:12, New York, NY, USA, 2011. ACM.

[25] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 119–128, 2007.

[26] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX conference on Networked Systems Design & Implementation*, NSDI'07, pages 229–242, Berkeley, CA, USA, 2007. USENIX Association.

[27] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper: Power and migration cost aware application placement in virtualized systems. In *Middleware 2008*, volume 5346 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008.

[28] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755–768, May 2012.

[29] Fabien Hermenier, Sophie Demassey, and Xavier Lorca. Bin repacking scheduling in virtualized datacenters. In *Proceedings of the 17th international conference on Principles and practice of constraint programming*, CP'11, pages 27–41, Berlin, Heidelberg, 2011. Springer-Verlag.

[30] Anton Beloglazov and Rajkumar Buyya. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '10, pages 4:1–4:6, New York, NY, USA, 2010. ACM.

[31] Sangmin Lee, Rina Panigrahy, Vijayan Prabhakaran, Venugopalan Ramasubramanian, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Validating heuristics for virtual machines consolidation. Technical Report MSR-TR-2011-9, January 2011.

[32] Akshat Verma, Juhi Bagrodia, and Vimmi Jaiswal. Virtual machine consolidation in the wild. In *Proceedings of the 15th International Middleware Conference*, Middleware '14, pages 313–324, New York, NY, USA, 2014. ACM.

[33] Ratnesh K. Sharma, Cullen E. Bash, Chandrakant D. Patel, Richard J. Friedrich, and Jeffrey S. Chase. Balance of power: Dynamic thermal management for internet data centers. *IEEE Internet Computing*, 9(1), January 2005.

[34] Justin Moore, Jeff Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.

[35] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, New York, NY, USA, 2007. ACM.

[36] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., New York, NY, USA, 2006.

[37] Fabien Hermenier, Julia Lawall, and Gilles Muller. Btrplace: A flexible consolidation manager for highly available applications. *IEEE Transactions on Dependable and Secure Computing*, 10(5), 2013.

[38] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 41–50. ACM, 2009.

[39] E. Bin, O. Biran, O. Boni, E. Hadad, E.K. Kolodner, Y. Moatti, and D.H. Lorenz. Guaranteeing High Availability Goals for Virtual Machine Placement. In *International Conference on Distributed Computing Systems*, 2011.

[40] Roman Krogt, Jacob Feldman, James Little, and David Stynes. An integrated business rules and constraints approach to data centre capacity management. In David Cohen, editor, *Principles and Practice of Constraint Programming*, volume 6308 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010.

[41] K. Tsakalozos, M. Roussopoulos, and A. Delis. Hint-based execution of workloads in clouds with nefeli. *Parallel and Distributed Systems, IEEE Transactions on*, 24(7):1331–1340, 2013.

[42] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08. USENIX Association, 2008.

[43] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira, Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '05, pages 186–195, New York, NY, USA, 2005. ACM.

[44] Dimitris Economou, Suzanne Rivoire, and Christos Kozyrakis. Full-system power analysis and modeling for server environments. In *In Workshop on Modeling Benchmarking and Simulation*, 2006.

[45] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: Introduction and applications. *Commun. ACM*, 54(9):69–77, September 2011.

[46] Adrian Paschke and Elisabeth Schnappinger-Gerull. A categorization scheme for sla metrics. In *Multi-Conference Information Systems*, MKWI'06, 2006.

[47] Robert Basmadjian, Hermann de Meer, Ricardo Lent, and Giovanni Giuliani. Cloud computing and its interest in saving energy: the use case of a private cloud. *Journal of Cloud Computing*, 1(1), 2010.

[48] R. Basmadjian, F. Niedermeier, and H. De Meer. Modelling and analysing the power consumption of idle servers. In *Sustainable Internet and ICT for Sustainability (SustainIT), 2012*, pages 1–9, 2012.

[49] Robert Basmadjian and Hermann de Meer. Evaluating and modeling power consumption of multi-core processors. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, New York, NY, USA, 2012. ACM.

[50] Venkatesh Pallipadi. Enhanced intel speedstep technology and demand-based switching on linux. *Intel Developer Service*, 2009.

[51] Robert Basmadjian, Nasir Ali, Florian Niedermeier, Hermann de Meer, and Giovanni Giuliani. A methodology to predict the power consumption of servers in data centres. In *Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking*, e-Energy '11, New York, NY, USA, 2011. ACM.

[52] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09. Springer-Verlag, 2009.

[53] Jing Xu and Jose A. B. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, pages 179–188, Washington, DC, USA, 2010. IEEE Computer Society.

[54] Virtualization penetration rate in the enterprise. Technical report, Veeam Software, 2011.

[55] Paul Shaw. A constraint for bin packing. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 648–662. Springer Berlin Heidelberg, 2004.

[56] Íñigo Goiri, Ryan Beauchea, Kien Le, Thu D. Nguyen, Md. E. Haque, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. GreenSlot: scheduling energy consumption in green datacenters. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 20:1–20:11, New York, NY, USA, 2011. ACM.

[57] Chao Li, A. Qouneh, and Tao Li. iSwitch: Coordinating and optimizing renewable energy powered server clusters. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 512–523, June 2012.

[58] Íñigo Goiri, Kien Le, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. GreenHadoop: Leveraging Green Energy in Data-processing Frameworks. In *Proceedings of the 7th ACM European Conference on Computer Systems*, pages 57–70, New York, NY, USA, 2012. ACM.

[59] Zhenhua Liu, Adam Wierman, Yuan Chen, Benjamin Razon, and Niangjun Chen. Data center demand response: avoiding the coincident peak via workload shifting and local generation. pages 341–342, New York, NY, USA, 2013. ACM.

[60] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009.

[61] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003.

[62] Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W. Moore, and Andy Hopper. Free Lunch: Exploiting Renewable Energy for Computing. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, pages 17–17, Berkeley, CA, USA, 2011. USENIX Association.

[63] Xu Yang, Zhou Zhou, Sean Wallace, Zhiling Lan, Wei Tang, Susan Coghlan, and Michael E. Papka. Integrating Dynamic Pricing of Electricity into Energy Aware Scheduling for HPC Systems. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 60:1–60:11, New York, NY, USA, 2013. ACM.

[64] Zhou Zhou, Zhiling Lan, Wei Tang, and Narayan Desai. Reducing Energy Costs for IBM Blue Gene/P via Power-Aware Job Scheduling. October 2013.

[65] D. Aikema and R. Simmonds. Electrical cost savings and clean energy usage potential for HPC workloads. In *2011 IEEE International Symposium on Sustainable Systems and Technology (ISSST)*, pages 1–6, 2011.

[66] Alexandra Carpen-Amarie, Djawida Dib, Anne-Cecile Orgerie, and Guillaume Pierre. Towards energy-aware IaaS-PaaS co-design. In *SMARTGREENS 2014*, pages 203–208, 2014.

[67] Michael Cardosa, Madhukar R. Korupolu, and Aameek Singh. Shares and utilities based power consolidation in virtualized server environments. In *Proceedings of the 11th IFIP/IEEE Integrated Network Management (IM 2009)*, Long Island, NY, USA, June 2009.

[68] Kiril Schröder and Wolfgang Nebel. Behavioral model for cloud aware load and power management. In *Proc. of HotTopiCS '13, 2013 international workshop on Hot topics in cloud services*, pages 19–26. ACM, May 2013.

[69] Mehdi Sheikhalishahi, Richard M. Wallace, Lucio Grandinetti, José Luis Vazquez-Poletti, and Francesca Guerriero. A multi-capacity queuing mechanism in multi-dimensional resource scheduling. In Florin Pop and Maria Potop-Butucaru, editors, *Adaptive Resource Management and Scheduling for Cloud Computing*, Lecture Notes in Computer Science, pages 9–25. Springer International Publishing, January 2014.

[70] Frederico Alvares de Oliveira, Jr. and Thomas Ledoux. Self-management of cloud applications and infrastructure for energy optimization. *SIGOPS Oper. Syst. Rev.*, 46(2):10–18, July 2012.

[71] F.A. de Oliveira, T. Ledoux, and R. Sharrock. A framework for the coordination of multiple autonomic managers in cloud environments. In *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 179–188, September 2013.

[72] Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.*, 41(1):45–52, January 2011.

[73] Roberto Morabito. Power consumption of virtualization technologies: an empirical investigation. In *IEEE/ACM UCC 2015 (SD3C Workshop)*, 2015.

[74] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. Technical report, IBM Research Division, 2014.

[75] Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya. A framework and algorithm for energy efficient container consolidation in cloud data centers. *Proceedings of the 11th IEEE International Conference on Green Computing and Communications (GreenCom 2015, IEEE CS Press, USA)*, 2015.

[76] Frank Bellosa. The benefits of event: Driven energy accounting in power-sensitive systems. In *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System*, EW 9, pages 37–42, New York, NY, USA, 2000. ACM.

[77] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library operating systems for the cloud. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 461–472, New York, NY, USA, 2013. ACM.

[78] Dmitry Namiot and Manfred Sneps-Sneppe. On micro-services architecture. *International Journal of Open Information Technologies*, 2014.

[79] Nosayba El-Sayed, Ioan A. Stefanovici, George Amvrosiadis, Andy A. Hwang, and Bianca Schroeder. Temperature management in data centers: Why some (might) like it hot. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 163–174, New York, NY, USA, 2012. ACM.

[80] Muhammad Tayyab Chaudhry, Teck Chaw Ling, Atif Manzoor, Syed Asad Hussain, and Jongwon Kim. Thermal-aware scheduling in green data centers. *ACM Comput. Surv.*, 47(3):39:1–39:48, February 2015.