Improved Self-management of DataCenter Systems Applying Machine Learning

by
Josep Lluís Berral García

Advisors:
Prof. Jordi Torres Viñals
Prof. Ricard Gavaldà Mestre

A dissertation submitted in partial fulfilment of the requirements for the degree of *Doctor per la Universitat Politècnica de Catalunya*

Universitat Politècnica de Catalunya PhD program
Departament d'Arquitectura de Computadors

Barcelona, Catalunya (Spain), 2013

Abstract

Autonomic Computing is a Computer Science and Technologies research area, originated during mid 2000's. It focuses on optimization and improvement of complex distributed computing systems through self-control and self-management. As distributed computing systems grow in complexity, like multi-datacenter systems in cloud computing, the system operators and architects need more help to understand, design and optimize manually these systems, even more when these systems are distributed along the world and belong to different entities and authorities. Self-management lets these distributed computing systems improve their resource and energy management, a very important issue when resources have a cost, by obtaining, running or maintaining them. In this Ph.D.Thesis we propose to improve Autonomic Computing techniques for resource management by applying modeling and prediction methods from Machine Learning and Artificial Intelligence. Machine Learning methods can find accurate models from system behaviors and often intelligible explanations to them, also predict and infer system states and values. These models obtained from automatic learning have the advantage of being easily updated to workload or configuration changes by re-taking examples and re-training the predictors. So employing automatic modeling and predictive abilities, we can find new methods for making "intelligent" decisions and discovering new information and knowledge from systems. Summarizing, we try to improve multi-datacenter systems using machine learning by 1) finding automatic learned decision makers, easier to build and update than hand-made ones, and not much worse or even better; 2) being more energy efficient, optimizing resource usage without compromising the quality of service offered by the system; and 3) managing them autonomously enough to free operators from supervising the system constantly.

Agraïments

Permeteu-me en un primer moment, abans de començar amb la *chicha* que pertoca a aquesta tesi, de fer una relació de totes aquelles persones que, durant aquests darrers 5 anys, m'han ajudat o m'han hagut de suportar. Persones de qui no només he obtingut ajut i suport, si no també de qui he après gairebé tot el que he vist i fet durant aquesta tesi, i que m'han servit d'exemple a seguir com a acadèmic, com a professional i com a persona.

Primer de tot agraïr tota la feina i esforç dels meus dos directors, en Jordi Torres i en Ricard Gavaldà. Em van oferir la oportunitat de coneixer el món de la recerca i d'obtenir nous coneixements de dues arees de coneixement tan ortogonals però tan complementàries com son l'Autonomic Computing i el Machine Learning. Sense la seva dedicació i recolzament no hagués estat possible aquest treball.

Cal fer menció especial a dues persones que m'han ajudat en extrem durant tot aquest temps, que son l'Íñigo Goiri, un gran professional de la ciencia i de la vida, i en Javier Alonso, gran investigador i encara millor persona. També fer un reconeixement especial a la gent del grup abans conegut com "eDragon", com en Ramon Nou, en Ferran Julià, en Nico Poggi, en Jordi Guitart i en David Carrera, amb qui treballar o compartir coneixements és sempre un plaer.

Indicar també un reconeixement a la gent del grup LARCA de qui també he après molt, com en Borja Valle, en Dani Alonso, en Miquel Camprodon i en Ramon Xuriguera; així com els companys del LSI, en Sergi Oliva, en Jorge Muñoz, en Javier De San Pedro, en Nikita Nikitin, l'Adrià Gascón i l'Alessandra Tosi, l'Albert Vilamala, l'Eva García i en Carles Creus. Ha estat un plaer servir aqui amb vosaltres.

Important recordar a la gent que ha estat la meva segona familia durant els darrers dotze anys en aquesta arcàdia anomenada Campus Nord, la gent de l'Oasi i Distorsió, a qui dedico aquesta obra. Tant els antics *Oasieros* com en Modesto, Juan Pedro, David, Álvaro i Sergio, i els *Distorsionats* Romà, Guille, Carlos, Miguel, Arnau, Guillem, David, Christian, Ramon, Nicolás, Pedro, Judit, Dani, Jordi i Alex; així com els nous (antics nous i nous de trinca) Alberto (F), Alberto (K), Hector, Eric, Elisabeth, Natàlia (que s'ha currat aquesta fantàstica portada), David, Arnau, Agustín, Lino, Magí, Sergi, Ignacio, Marc, Quimi, Alejandro L., Alejandro N. i Imanol. Només dir que ens veiem al bar, ara baixo.

Un agraïment especial també a la gent del GCO-DAC, companys de dinars i cafès, i també de feines presents i futures, en Luis Velasco, en Marc Ramirez, l'Alberto Castro i Lluís Gifré. Espero que poguem seguir fent ciència plegats i potser fer-nos rics algun dia.

No voldria oblidar-me d'aquella gent que van abandonar els seus paisos per trobar-se a Nou Brunswick, Nova Jersey, amb qui vaig compartir uns mesos molt valuosos a Rutgers: en Cheng Li, en Quingyuan Deng, en Guilherme Cox i el professor Ricardo Bianchini. Espero de veritat que els vagi molt bé la vida i que cadascú trobi el seu camí per ser feliç.

In Memoriam al Ministeri d'Innovació i Ciència d'Espanya, sense el qual això no hagués estat possible, i que fou una de les primeres víctimes de la involució en cultura i coneixement soferta en aquest pais recentment.

Finalment, dedicar aquesta feina a la meva família, per haver-me suportat tot aquest temps i haver tingut paciència amb mi.

This work has been supported by the Spanish Ministry of Science and the Spanish Secretariat for Research (nowadays part of the Ministry of Finances and Competition), under contract TIN2011-27479-C04-03 and under FPI grant BES-2009-011987 (TIN2008-06582-C03-01), by EU PASCAL2 Network of Excellence, and by the Generalitat de Catalunya (SGR-2009-1428).

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

BWD Bandwidth

CAPEX Capital Expenditures

DC DataCenter

DDoS Distributed Denial of Service

DVFS Dynamic Voltage/Frequency Scaling

EEFSIM Energy-Efficient Simulator

HPC High Performance Computing

IO Input/Output

ISP Internet Service Provider

IT Information Technologies

MAE Mean Absolute Error

MAPE Monitor/Analize/Plan/Execute

MSE Mean Squared Error

MILP Mixed Integer Linear Program

ML Machine Learning

OPEX Operational Expenditures

PM Physical Machine

PUE Power Usage Efficiency

QoS Quality of Services

RL Reinforcement Learning

RR Round-Robin

RT Response Time

SLA Service Level Agreement

SLO Service Level Object

VM Virtual Machine

WS Web-Service

Chapter 1

Introduction

"Intelligence": The art of good guessing. - H.B.Barlow
"Artificial Intelligence": Artifact able to guess good, given limited resources, time or space.

1.1 Introduction

The Web 2.0 and the computing/storage web service business have contributed to *democratize* the Internet, allowing everybody to share information, services and Information Technologies (IT) resources around the network. With the arrival of social networks and the introduction of new IT infrastructures into the business world, the Internet population has grown enough to make the need for computing resources an important matter to be treated. While few years ago enterprises had all their IT infrastructures in privately owned DataCenters (DC), nowadays the big IT corporations have started a *DataCenter-race*, offering computing and storage resources at low prices, looking for outside companies to trust their data or IT needs on them.

These offered resources are also referred as "the Cloud", a place in the Internet where you can access from anywhere, upload and download all your information, send your data to compute, without the need of knowing where and how the physical infrastructure is. For the user, everything is reduced to a user-friendly web service, so companies can delegate their e-mail boxes, web applications, and the computing jobs to *pay-as-you-go* IT resources companies like Amazon, Google, etc, instead of spending money on private datacenters, with all the extra infrastructure it requires. And this makes not only enterprises to work into the Cloud, but also casual users start using it. Most of the existing social networks and popular web-services began from user initiatives, and as these services grow popular, more IT resources are required to run them. Only a few extremely large companies such as Facebook need their own Cloud for handling all the users and data stored into.

Handling these requirements to give good Quality of Service (QoS) requires not only a powerful enough single datacenter. A single web application or piece of data can be easily used by people around the world, so often it mush be available from everywhere, keeping in mind things like quality of service and service level agreements between users and service providers. For example, Google and Youtube receive queries from around the world at the same time, and must keep replicas of their web-services near each client, and move each piece of data as close to its final user as possible. For a service that can be replicated, it requires moving just data, otherwise copies of unique set of services must be placed wisely among the *datacenter farms* among the planet, to ensure QoS and reduce costs. This means coordinating all jobs, services and applications in the Cloud system with all its resources, and given the amount of jobs running nowadays on it, this becomes a hard optimization problem.

Before the Cloud and the Web 2.0, technological improvement sufficed to cover the increasing IT demand, bringing faster processors, larger storage devices, and faster connections between resources. But now the demand is growing faster than technological improvement, so every day we require larger datacenters and more computational power, with a larger energetic demand, and the requirement of strategic placement of resources and distribution of load around the world. Further more, in most situations more computational power or larger DCs do not mean better

QoS and more availability, and reaching a (near) optimal performance of Cloud services and resource management passes through an *Intelligent Management*, aware of the state and capability of each resource used and the needs of each user. This Intelligent Management complements the technological improvement, allowing better resource use, borrowing and lending resources when it is convenient to do so, and improving the quality of service without scaling the DataCenters unnecessarily.

In order to proceed with decision making, knowing in detail the structure and environment of every element involved in the system is crucial. Not having this information prevents us of having experts (automatic or human), advising about what to do in each situation, giving us hints for best solutions and best practices, and having reactive elements keeping an eye over the system constantly to cover any incident or change.

Unfortunately, in the Cloud all of this is not possible, as 1) it becomes an abstract set of resources, each one with abstract set of properties. Each domain of resources has its own resource broker and interface for dealing with resource borrowers and lenders, so a part of the Cloud can not manage or get all the information to other parts of it. 2) Systems running on the Cloud are hard to model explicitly by hand and hard to predict due its internal complexity and also complex dependence on external input including human input. For most of the applications and services running in the Cloud there are no experts, and most variables indicating the system status are hidden to the naked eye, making it difficult to predict the behavior of the hole (or even a part of) the system. 3) Given the amount of elements to control and monitor, also the amount of data generated by each element to be checked, keeping a human operator watching over each warning, change or information, reacting in front of each, becomes practically impossible. In front of these problems, automation and novel techniques for understanding the system become part of the solution, "understanding" what is happening in the system and handling all the decisions to be made in the most autonomous way as possible.

Current datacenters and large-scale distributed computing systems (the underlying structure of the Cloud) are recently implementing techniques of autonomic computing, a field on science focusing on the automation of computation systems (from single computers to multi-DC systems). The need for managing large-scale systems has made this automation a hot topic on research, opening sub-topics focusing on different areas of improvement: self-healing, as the automation of detecting and solving system failures; self-protection, as the automation of enforcing security over systems and data; self-configuration, as the automation of deploying services and applications on systems; and self-optimization, as the automation of improving performance of services and systems. By default, the research on these topics is been done taking advantage of ad-hoc handcrafted expert systems, statistic models and rules. We put forward the central hypothesis of this thesis: that these solutions can be improved if the system is able to generate models and rules from what it can observe, adapting each model to the given states and environment, "learning by itself" how the system behaves.

Machine learning (viewed as part of a larger field, data mining) brings a methodology to, given a set of observations from a given system, discover knowledge from its behavior, and predict elements of it. The advantage of using machine learning instead of systems built from explicit modeling is that 1) we do not always we have an expert, 2) we have too much information to treat, 3) this information may be incomplete, missing, or uncertain, 4) the system may change over time. Machine learning methods focuses on these issues, some of them being robust against sparse scenarios (scenarios with states sharing low similarity among them), some robust against changes (scenarios changing constantly) or easy to update, and ready to handle high dimensionality (scenarios with too much data and features for each piece of information to be treated).

The usual methodology for creating models using inductive learning require 1) a period of observation of the system to be learned from, 2) a learning process to create a model describing the system behavior, and 3) a testing period.

The ability of using this learned knowledge to improve performance of large-scale systems like Cloud systems opens a new wide research area combining all the capabilities of Autonomic Computing and the capabilities of learning and discovering knowledge from these systems.

1.2 Motivation

Currently, intense research is done related to learning and artificial intelligence application into networks and systems. Many existing works focus on specific details of a given system, like a specific component or studying a specific policy. This thesis and research goes beyond this kind of solutions requiring fixed models created by experts or specific ad-hoc details over components or single policies. Often, in large-scale computing systems, we do not know about the whole system; often it is too complex to understand each piece or detail; often the system is in constant change, enough to invalidate all fixed models or established policies.

This thesis is about making management of large-scale systems to be as automatic but accurate as possible. We want management to be adaptive, able to get and adjust the models describing web-services and infrastructures, in order to

- let administrators and operators to make their work easier
- make decisions (scheduling, allocate resources and switch on and off resources) automatically, including energy management
- offer good quality of service to the users of the infrastructure also to the users of the web-services relying on the infrastructure
- obtain knowledge from the system learning from the generated models

This thesis researches and develops a set of techniques, methods and strategies, letting the system learn about its own behaviors and responses given expected and unexpected scenarios, model these, and find the adequate policies to improve the system management using the resulting models. We focus on a particular scenario, very common in datacenter infrastructures and services (like e.g. web-service hosting at Amazon), the virtualized environments. The methods and strategies can be studied by observing how machine learning improve management in autonomic scenarios, or how if made decisions make the system perform as well as ad-hoc made expert systems, all of this obtaining adaptively by itself this expert knowledge.

1.3 Goals of this thesis

The main goal of this thesis is to demonstrate that, with the use of machine learning techniques, management of virtualized web-services on a multi-datacenter system can be improved in quality of service terms and energy consumption.

In order to achieve this main goal, this thesis takes a walk from the state of the art where the management of virtualized web-services is based on administrators expertise, well known data, ad-hoc studied algorithms and models, and elements to be studied to be from computing machine point of view; to reach a novel state of the art where the management is driven by models learned from the same system, providing useful feedback, making up for incomplete, missing or uncertain data, from a global network of datacenters point of view.

The steps in this walk can be seen as a movement between scaling the scenario, from computing machines to multi-datacenters, and the required level of knowledge and prediction, from having a given explicit model to having no a-priori knowledge of the system potential environment and behavior. We start by automatically modeling a single machine, pass by the level of a computing cluster, and reach the level of a multi-datacenter infrastructure. In between we model each system mathematically, identify the relevant information required to manage it and find methods to predict that information and apply it.

Stage 1 - *Tailoring Jobs and Resources*: We depart from a point where, having a datacenter or a set of machines hosting virtualized jobs (be High Performance Computing (HPC) or Transactional jobs like Web-Services (WS)), everything must be arranged and scheduled so as to maintain these jobs accomplishing their deadlines and respecting their quotas of resources (CPU or Memory). This first stage covers the scenario where the decision maker works knowing all pieces of information from the system: how much will each job consume, how is and will be the

desired quality of service, what are the deadlines for the workload, etc. All of this focusing on each component and policy of each element involved in executing these jobs. The work included in this stage coincides in part with the PhD thesis of Íñigo Goiri, working on modeling in detail virtualized environments towards economic profit, where we had an active involvement as a collaboration on the experimental part. Here we only present the modeling work corresponding to this thesis.

Stage 2 - *Predictions on Decision Making*: When controlling components in a real environment, and also when making decisions that affect the future, we need to predict what the future may look like including the effect of our decisions and actions. Usually estimators are built by experts who know the system, but the experts may not be available when the system becomes large, or the situation to be handled is complex enough to design an ad-hoc decision maker worth it. In this stage the work contemplates the scenario where instead of fixed oracles that provide us information from an expert formula or set of conditions, machine learning is used to create these oracles. Here we stand in a point where we look at components and specific details while some part of the information is not known and must be learned and predicted.

Stage 3 - *Mathematical Modeling of DCs*: When energy becomes an important factor, the function optimizing costs and benefits must take care of consumption, and reduce the usage of resources while maintaining performance. It should not only be limited to CPUs in a machine, but extended to a whole datacenter by reducing the on-line machines to the minimal required. So the optimization can be expanded by looking to the global datacenter environment. This stage begins reducing the problem of optimizing the resource allocations and requirements for virtualized web-services to a mathematical problem, indicating each factor, variable and element involved, also all the constraints the scheduling process must attend to. Our scheduling problem can be modeled as a Mixed Integer Linear Program (MILP). Here we are in a point where we face an scenario of a full datacenter, further we introduce some information prediction (for only CPU resources at this step) using the methodologies from the previous step.

Stage 4 - *Learning in DataCenters*: Our mathematical problem can be guided also by learned modules providing predictions for each placement so a solver will require less expert ad-hoc speculations, also less information from noisy monitors. Each element can be modeled so the scheduler can use a predictor for each piece of data coming from monitors, speculate with tentative schedules, and choose the predicted best situation. We complement the previous stage by expanding the predicted elements, studying the main resources, this is CPU, Memory and IO, that can suffer from noise, inaccuracy or unavailability. Once learning predictors for certain components let the decision making improve, the system can become more "expert-knowledge independent" and research can focus on an scenario where all the elements provide noisy, uncertainty or private information.

Stage 5 - *Extending to Multi-DCs*: What can be achieved by managing efficiently a datacenter can be exported to a model where this datacenter is spread along the world, or what is the same, several datacenters distributed along the world are united as a federation. New factors are involved in the optimization, as for each datacenter, costs may change as energy has different prices in different locations. Also each datacenter may provide different times of service due to proximity to the client.

Stage 6 - *Green Multi-DataCenters*: A final approach, once we have a system relying on a datacenter network, is to optimize the energy costs by powering them using green energy (energy from renewable sources). Unlike other chapters, in this one we do not address the management of an existing computing structure, but its design and placement before it is built. Planning and building the datacenters near renewable sources and take advantage of solar and wind energy availability results cheaper and more affordable than the usually expected. We finish this thesis with a view of the cost of placing datacenters depending on green energy sources, and distribute the load according to green energy availability.

At the end of the journey we have moved from a stage where every policy is calculated ad-hoc for each job and component, to a stage where the scenario is a full system of datacenters. All important information that can become inaccurate or missing is predicted, driving the decision manager to grant quality of service to datacenter customers and web-service clients, having into account very important factors like energy consumption.

Note that all the steps above are orientative: they may overlap and often are anticipated elements of the next.

1.4 About this thesis

This thesis is a multidisciplinary work joining two different research areas, autonomic computing and machine learning. While Autonomic Computing cares about self-management techniques like self-healing, self-protection, self-configuration and self-optimization, machine learning brings a set of methods for modeling systems, predicting system values, making "intelligent" decisions, and adapting these models and predictions when the system changes. Further, this thesis is being developed as a research project inside two research groups:

- The LARCA research Group (Laboratori of Relational Algorithmics, Complexity and Learnability) is a research group composed by members of LSI *Departament de Llenguatges i Sistemes* and mostly of the UPC, working on data mining, complexity and computational learning, and its applications; directed by professor Ricard Gavaldà.
- The Autonomic-HPC (Autonomic Computing, at the High Performance Computing research Group) is a research group composed by members of the DAC *Departament d Arquitectura de Computadors* of the UPC and the BSC-CNS *Barcelona Supercomputing Center - Centro Nacional de Supercomputación*, working on Autonomic Computing and new methods for self-management in distributed systems; directed by professor Jordi Torres.

Also, this work is or has been financially supported by:

- The Spanish Ministry of Science FPI grant BES-2009-011987
- The Spanish Ministry of Science MOISES-BAR project (Individualized Modeling of Symbolic Sequences) TIN2005-08832-C03-03, coordinated by Ricard Gavaldà.
- The Spanish Ministry of Science SESAAME project (SEcuencias Simbólicas: Análisis, Aprendizaje, Modelado y Evolución) TIN2008-06582-C03, coordinated by Ricard Gavaldà.
- The Spanish Ministry of Science BASMATI project (Biological and Social Mining: Algorithms, Theory, and Implementation) TIN2011-27479-C04-03, coordinated by Ricard Gavaldà.
- The Spanish Ministry of Science CAP-VI project (Computación de Altas Prestaciones VI) TIN2012-34557.
- The Generalitat de Catalunya (2009-SGR-1428).
- The EMOTIVE-Cloud BSC Project directed by Jordi Torres.
- The EU COST Action IC0804, Energy Efficiency In Large Scale Distributed Systems, coordinated by Jean-Marc Pierson.
- The EU PASCAL2 Network of Excellence. Pattern Analysis, Statistical Modelling and Computational Learning.
- The EU CoreGRID Network of Excellence. European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies.

1.5 Document Outline

The remainder of this document is structured as follows:

Chapter 2 provides some Background Concepts and the State of the Art in the relevant areas: autonomic computing, data mining and machine learning concepts, and the current applications of machine learning to Autonomic Computing. Given that the project is multidisciplinary, a broad survey becomes necessarily.

Chapter 3 visits works done previous to targeting the current goal and stages, in order to explore and perform some experiences toward the possibility of applying machine learning modeling to direct applications in distributed systems and autonomic computing, related to web-service self-optimization, self-defense and self-healing. This chapter is based on already published material (GDN07 [109], UM07 [131], TR-DAC07 [135], ComNet09 [133]), also (DSN10 [3], DPDNS10 [4]), also material presented as Master Thesis (AISec08 [32]).

Chapter 4 presents work on analytical and theoretic ad-hoc modeling on HPC and transactional virtualized jobs from a computing machine/datacenter (Stage 1). This chapter is based on already published material (ICAC08 [156], GRID10 [67], Cluster10 [70], FGCS12 [66]).

Chapter 5 presents work on machine learning applied to predicting components and variables to drive decision managers concerning to autonomic computing issues like self-optimization for a computing machine/datacenter (Stage 2). This chapter is based on already published material (EEnergy10 [30], GreenBook12 [31]).

Chapter 6 presents work on modeling a full datacenter reducing it to a mathematical program, optimizing the costs for running it taking into account quality of service for datacenter customers and web-service clients, and applying the machine learning techniques developed in previous stages (Stage 3). This chapter is based on already published material (TR-LSI10 [24], GRID11 [25],TR-LSI11 [26]).

Chapter 7 presents work on the study of learning and prediction of datacenter and computing machine component performance towards web-service load and different resource allocation scenarios (Stage 4). This chapter is based on already published material (SAC13 [27], TR-LSI13 [28]).

Chapter 8 presents work on managing world-wide datacenter systems, focusing on distribution of jobs according on energy obtaining, energy consumption, service placement and proximity to the web-service clients (Stage 5). This chapter is based on material being published (PASA13 [29]).

Chapter 9 presents work on building a world-wide datacenter system, oriented towards the availability of green energy sources, like solar and wind capacity (Stage 6). Material in this chapter is subject of a paper to be submitted to a main conference.

Chapter 10 draws the main conclusions of the thesis and proposes new issues to continue the research field and proposes some lines to continue the research in this thesis.

Chapter 2

Background

Nowadays data mining techniques and other stochastic models are being introduced into the field of autonomic computing, improving on modeling and prediction for decision making. This section explains some background concepts and provides some examples of applications to Autonomic Computing, Cloud Computing and self-management topics. Mostly, works provided in this section are focused on the direction of this thesis, by applying autonomic strategies, artificial intelligence or statistical learning in some way. Further, each chapter will include its own *related work* section pointing on their specific relevant concepts and important related works..

2.1 Relevant Concepts on Autonomic and Cloud Computing

2.1.1 About Autonomic Computing

Autonomic Computing definition can be found in the manifesto and challenge visions written by J.Kephart [91, 90] from IBM Watson Research Center:

As systems become more interconnected and diverse, architects are less able to anticipate and design interactions among components, leaving such issues to be dealt with at runtime. Becoming too massive and complex for even the most skilled system integrators to install... the only option remaining is computing systems that can manage themselves given high-level objectives from administrators.

So, the more complex and large systems become, operators and architects are less able to manage and create components that will fit accurately into these systems. For this, autonomic computing is responsible of giving *self-management* abilities to such complex systems, allowing degrees of autonomy for making decisions, like following self-healing, self-optimizing, self-configuring and self-protecting policies.

- Self-Healing: Self-Healing techniques are in charge of automatically detecting, diagnosing and repairing localized software and hardware problems. There is a lot of work done on recovering systems from a failure, but predicting these failures before they happen can help us to prepare the system for these recovery processes. If we are able to anticipate a failure, we can take some actions to prevent it or to attenuate its consequences. Some works are done on self-healing considering that recovering systems can be more efficient than prevent systems from failures, like Patterson et al. [38]. Anticipating the kind of failure, the moment of failure and as most details as possible, we can set up the recovery mechanism ready minimizing the impact of the process. E.g. works presented by Alonso, Silva et al. [6, 5, 143] are about these system rejuvenation methods, by predicting crashes

for web-service components and enabling replicas before crash. Prediction and anticipation require some modeling. Works like Cherkasova [181], Cohen [48] or Barham et al. [20] try to discover if a system is going to crash or not from its behavior by modeling it. While Cherkasova [181] relates a methodology obtaining linear regressions for system exhaustion, Cohen and Barham [48, 20] are about the system instrumentation in order to find a good model for diagnosis purposes.

- Self-Configuration: Self-Configuration techniques are in charge of configuring components and software of an autonomous system following high-level policies, often being these components from different vendors and providers. This makes installing and configuring them in a proper way a consumption of time and effort for operators, also finding the best integrated configuration is hard enough. There are some examples of works on self-configuring, like the ones from Wildstrom et al. [175, 174, 173], where reinforcement learning techniques are used to change configuration of software components, looking for improving execution of these components.

- Self-Optimization: Self-Optimization techniques are in charge of automatically improving the system goals, usually performance or economic values where performance is heavily implied. The decisions to be made are usually deciding the best values for parameters, selecting the best strategies to be followed, enabling the best policies to reach the system goals, or finding solutions to General Allocation Problems like scheduling tasks. As having a lots of manually set parameters, variables and decisions to be made over a given system, as it becomes complex these decisions become harder; another reason to automate the self-optimization tasks and proceed through "intelligent management". Self-Optimization is a challenging area where several works have been done. Examples are in Parashar, Chandra et al. [183, 42], where frameworks for self-managing like PRAGMA are extended by using predictive behavior models for applications and system status, adapting the policies on runtime. The challenge here is on perform this optimization over heterogeneous workloads and heterogeneous resources. While in systems where the owner is always the same (e.g. a grid system), optimization could be easier to perform as it was possible to know the dimensions of the whole domain and have most of the information of it available, but now in Cloud systems the optimization becomes harder as management systems may not know its full shape, or parts of the information required to manage the system is not available; another reason to proceed with data mining and machine learning techniques, exploring the data finding information and knowledge.

- Self-Protection: Self-Protection techniques are in charge of detecting attacks towards the system or cascade failures, letting management to react, warn or anticipate them, preventing a system-wide failure or data and information compromise. They are also in charge of preventing intrusions, malicious behaviors, exploitation of errors or any kind of happening that may harm the system or the users of such system. Recent advances have helped to take some of the burden of security maintenance of overloaded system administrators, such increasing of usage of automatic intrusion detection systems, secure embedded processors, proactive security measures, and automated virus response mechanisms, but there is much more to do. In the work of Chess et al. [46] some keys to be researched in the field of self-protection are summarized: ways to represent the security and privacy policies, represent the security states and trust relationships, resistant algorithms against fraud and persuasion, taxonomies for communication the privacy states and policies, methods for differentiating normal status and attacks and ways to construct autonomic elements so that their collective behavior is both trustworthy and trusted. Currently most of works on automated security are focused on Intrusion Detection Systems and Anti-Spam and Flooding Avoidance. E.g. works presented by Lee et al. [95] exposing IDS frameworks being updated using Data Mining techniques, incorporating new policies and patterns when needed. Or works presented by Y.M.Wang et al. [172] exposing frameworks to avoid spammers to promote their spam links into top search results using reinforced statistic techniques. Further works presented by Parashar, Berral et al. [178, 32] with methods for enhancing Intrusion Prevention Systems to avoid network and web-services flooding attacks automatically

from collecting and sharing information between agents in the network, learning patterns of attacks, predicting them and reacting.

Most of the current advances on Autonomic Computing are converging on the usage of more sophisticated techniques than simple statistical models and heuristics, introducing techniques from approximate algorithms, artificial intelligence, modeling and prediction, and data mining. Approaches with more capabilities by understanding better the system and making automatically decisions, and with better adaption to changes and noise.

2.1.2 About Cloud Computing

Nowadays the concept of Cloud has become one dominating paradigm in the externalization of information and IT resources for people and enterprises. In the "Cloud" business, the concept of "Cloud" can be defined as a "resource pool with on-demand resource allocation", a pay-as-you-go everything-as-a-service system, spread on an abstract infrastructure as seen by the common user (also more details for its definitions are found on [106]). The fact is that the Cloud is actually a name for the abstract concept of having data, web-services and computing resources available through the Internet, with no need of knowing details of the supporting infrastructure, but just a user-friendly interface to manage them. The infrastructure maintaining these systems, given their complexity and operational costs must be organized and managed in the most optimal, secure, robust and adaptable way. Again, the main focuses of Autonomic Computing.

The possibility of offering "everything as a service" (platform, infrastructure and services) has allowed companies to move their IT, previously in private, owned data-centers, to external hosting. Resource renting and Cloud resource offering has become an important business for IT companies, who have started, in the late 2000's, the business of offering infrastructures, platforms and services as a service. For companies that have IT needs, it is cheaper and more comfortable to rent resources (computational, storage or back-end and front-end services) than own IT equipment, paying just what the client company requires. As a consequence, the Cloud led to the creation of computing-resource provider companies, starting a *datacenter race* offering computing and storage resources at low prices. E.g. companies like Amazon with its EC2 and S3 [8], Google with its AppEngine and Cloud services [1], and IBM with the Blue Cloud [84], offer these resources with costs lower than a company buying and maintaining new IT that will be used temporarily [56].

The datacenter administration will essentially try to maximize its revenue by executing as many hosted services as possible, but is constrained by the its infrastructure: If too many services are accepted, quality of service will degrade, leading to immediate penalties as per Service-Level-Agreements (SLA), and eventually to prestige and customer satisfaction losses. Additionally, power consumption costs are becoming a growing portion of the operationg costs, besides increasing societal and environmental concerns. Naturally, high job throughput and user-satisfaction can be obtained by deploying a large amount of resources, but this incurs in high resource usage costs.

So, research is also on Cloud management, as it actually relies on (multi) datacenter infrastructures to be managed, optimizing processes and operations, always helping the operators and always transparent to its users. Works on self-management of these multi datacenter infrastructures are generally oriented to manage workloads, broker resources, and provide high quality of service to the resource customers. A large part of this work is focused on the middlewares managing the different Cloud layers (E.g. [118] for media file storage, [117, 122] for job and virtualization management), and different Cloud kind of works and workloads [40].

Resource Brokers

In the driving middlewares is where the autonomic computing techniques are applied. Resource brokering is in charge of manage resources to be given/obtained for each job or virtualized service. High-level brokers manage resources according to SLA policies, while low-level brokers attend to CPU/Memory/IO costs, distribution of the systems, system overheads for migration, virtualization, and running resources costs. All in all, the goal for resource brokers are to optimize the resource usage and assignation. This is why one commonly kind of resource broker

used is the economy-driven resource managers [37, 35, 171], where resources are driven, shared or allocated following market-oriented policies. Several works developed frameworks or methods for this economic resource management, e.g. projects Grace, Sorma, Spawn, Tycoon, etc.

The Sorma project [113, 120] presented an economic approach for efficient resource allocation in a Cloud, using market mechanisms. Their "Decentralized Local Greedy Mechanism" satisfies desirable economic properties for each supplier and requester promising to enable an efficient allocation of resources. With this mechanism, the participants in a network (companies, research institutes, etc.) try to selfishly maximize their individual benefit from participating in the network, and setting the right incentives for suppliers and requesters, for an efficient usage of the limited available resources, motivating the participants to cooperate and provide their idle resources. The Grace project (Grid Architecture for Computational Economy) [36] also presented another economy-based architecture for efficient resource allocation, where both resource owners and users maximize their objective functions, defining their charging and access policies, interacting with the resource pool by defining their requirements through high-level tools.

There are more frameworks (like [93], etc.) apart of these two, all of them with the purpose of allocating resources in a fair way, where clients and providers specify their conditions in service level agreements, and the brokers try to satisfy all the agreements as most as possible, considering the profit for each exceeded or violated agreements and other cost-benefit variables. Further, apart of market-based metrics, also other works treated the resource provisioning based on other ones, not only on the cost of the resources, like the waiting time for the resource or the runtime execution time of the workload [137].

2.1.3 Related Concepts: the Grid and Virtualization

Some concepts should be explained or visited to understand the fundamentals of Cloud Computing research area: "the Grid" and Virtualization techniques.

The Grid

Before the idea of "Cloud" developed, the research on managing distributed systems was focused on Clusters and "Grid" systems. I. Foster and C. Kesselman defined Grid systems as a *hardware and software infrastructure that provides dependable, consistent, pervasive and expensive access to high-end computational capabilities* [63, 64]. They also designed a list of requirements or goals for what a Grid system should be able to do:

- Coordinate resources that are not subject to centralized control, integrating and coordinating resources and users within different control domains.
- Do it using standard and general-purpose protocols and interfaces, built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access.
- And deliver non-trivial qualities of service to be used in a coordinated fashion to deliver various qualities of service meeting complex user demands so that the utility of the combined system is significantly greater than that of the sum of its parts.

The concepts and implementations of the Cloud have inherited many of the properties, concepts and problems from the Grid systems.

Virtualization

Virtualization is a technique (or set of techniques) to run processes, jobs, guest OS and also "machines" inside one or several Physical Machines (PM). The main capabilities of virtualization are to provide a confined environment where applications can be run, limit hardware resource access and usage or expand it transparently for the applications, adapt the runtime environment to the application, use dedicated or optimized OS mechanisms for each application, manage the whole applications and processes running within Virtual Machines (VM). These virtualized elements can be migrated among physical machines with different relying infrastructures or OSes,

dimensioned dynamically defining the size of the virtual machine (resources that will demand to the physical environment), also treated as a process for the host operating system. For more information about the architecture of VMs see [144], among other works on that area. Here we summarize the basic aspects of current products and solutions, obtained from [168]:

- Operating system-level approaches: these approaches allow to virtualize a physical server enabling multiple isolated and secure virtualized servers to run on a single physical server. No guest OS are used and applications run in a specific view of the only one OS as if they were alone running on the operating system. Some of these approaches are VServer [101], a kernel patch based on partitioning, using a "security context" inside a UNIX OS, FreeBSD Jail [136], and also Solaris Containors, OpenVZ, etc.

- Emulators: Virtual machines simulate the complete hardware used by a guest OS. VMware [169] is a virtualization software for machines based on x86 architecture, where virtualization works at the processor level, the virtual machine privileged instructions are trapped and virtualized by the VMware process and other instructions are directly executed by the host processor. All hardware resources of the machine are also virtualized. Other solutions are Microsoft VirtualPC, Oracle VirtualBox, QEMU [23], etc.

- Operating system in user space: These approaches provide virtualization through the execution of guest operating systems directly in user space. Some approaches are User Mode Linux [82], that allows launching Linux OS as applications of a host machine running Linux, and also coLinux, Adeos, L4Ka based projects, etc.

- Paravirtualization: the paravirtualization technique does not necessarily simulate the hardware, but instead offers a special API requiring modifications to the guest OS. The hardware resources are abstract resources not necessarily similar to the actual hardware resources of the host machine. Xen [21] is a virtual machine monitor for x86 architecture, allowing the concurrent executions of multiple OS while providing resource isolation and execution confinement between them. Other projects using this paravirtualization approach are Denali and Trango.

- Hardware-assisted virtualization: this virtualization allows to run unmodified guest OS giving to the VM its own hardware. This is possible thanks to an increase set of processor instructions provided by Intel VT (IVT [86]), AMD (AMD Pacifica x86 virtualization [9]), IBM (IBM Advanced POWER virtualization [83]) and Sun (Sun UltraSPARC T1 hypervisor [107]).

The design of Cloud systems take advantage of the possibility of virtualizing services, jobs and processes, as one physical machine can hold several of them. Further, services and HPC jobs running in the Cloud can be easily run in isolation, tailoring their resources and consumptions, moving them across the network of multi-DCs for placing them in the most proper location, and stopping and resuming the virtual machines conveniently. Encapsulating these jobs in virtual machines, the system manager can migrate the job across the network of datacenter looking for proximity of services toward clients (improving the quality of experience), scheduling the job with the required resources (speeding up job deadline or enforcing resource agreements, and thus improving quality of service), or placing the job in a place where operational costs can be reduced (e.g. energy availability, looking for cheap available energy or green energy, further applying consolidation techniques to reduce resource consumption).

Virtualization has to pay usually overheads, as it implements a software layer in the OS software stack; also VM operations like VM migration requires time to get accomplished, where the service or job is not running or providing service. Although that, as we see in works using virtualization, if well used it becomes an affordable cost in front of all the capabilities this technology provides.

2.1.4 Cloud Resource Business

The main actors on the datacenter business model are:

- The *datacenter manager*, or cloud service provider, wants to maximize its final revenue by optimally using the physical and virtual resources she has provisioned. To this end he/she has a certain freedom to allocate his customer's tasks to physical and virtual resources over time.
- The Cloud *customers* want to run their services on the cloud. In order to do this, they negotiate with the the cloud provider a certain amount of quality of service or service level agreements that they deem sufficient for their desired level of client satisfaction.
- The service *clients* want to accede to the services on the Cloud, paying the service owners for it. Client satisfaction affects whether the service will produce revenue or not.

Typical commercial datacenters are designed so that customers can run web-services running without knowing details of the infrastructure. Customers pay the providers on a usage-basis, and providers ensure that the web-services will be running according the service level agreements, negotiated by customers to accommodate their clients requests at sufficient satisfaction level. The datacenter contains physical machines, and each one holds virtual machines containing data and services from different customers. Virtualization is used to ensure isolation of customer and client contents, privacy of sensitive data, and allow for VM and service migration among PMs. The provider enables a VM for each customer, adjusts the granted resources for the web-service given the QoS, and set it into a PM with these available resources. His/her goal is to maximize the profit of each VM fulfilling the QoS and reduce the cost of running resources. Figure 2.1 shows the business infrastructure.

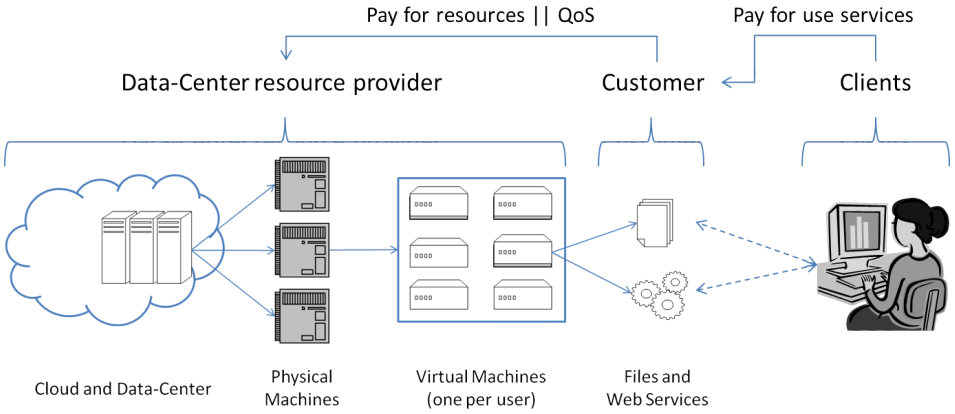


Figure 2.1: Commercial hosting infrastructure

A middleware software such as OpenNebula [147], Eucaliptus [122] or EmotiveCloud [162] is typically used in cloud-like architectures in order to manage PMs, VMs, network elements and traffic. Figure 2.2 shows the typical cloud middleware infrastructure. An agent on each PM controls VMs and monitors PM resources and VM requirements; the decision maker, on the machine hosting the resource manager and scheduler, reads all monitored lectures and makes decisions on a "Monitor/Analize/Plan/Execute" (MAPE) control loop [43]; also an agents on the network gateways redirect and also monitor traffic.

2.2 Relevant Concepts on Green Computing

Energy-related costs have become a major economical factor for IT infrastructures and datacenters because of the power's price escalation. Companies are now focusing more than ever on the need to improve energy efficiency. A new challenge has appeared besides the energy cost, the reduction of the carbon footprint, due to many EU regulations and campaigns demanding greener businesses. Commercial electricity consumption is a major contributor to the rising atmospheric CO_2 levels and datacenters are one of the foremost parts of the problem. Energy costs are rising, datacenter equipment is stressing power and cooling infrastructures, and the main issue is not

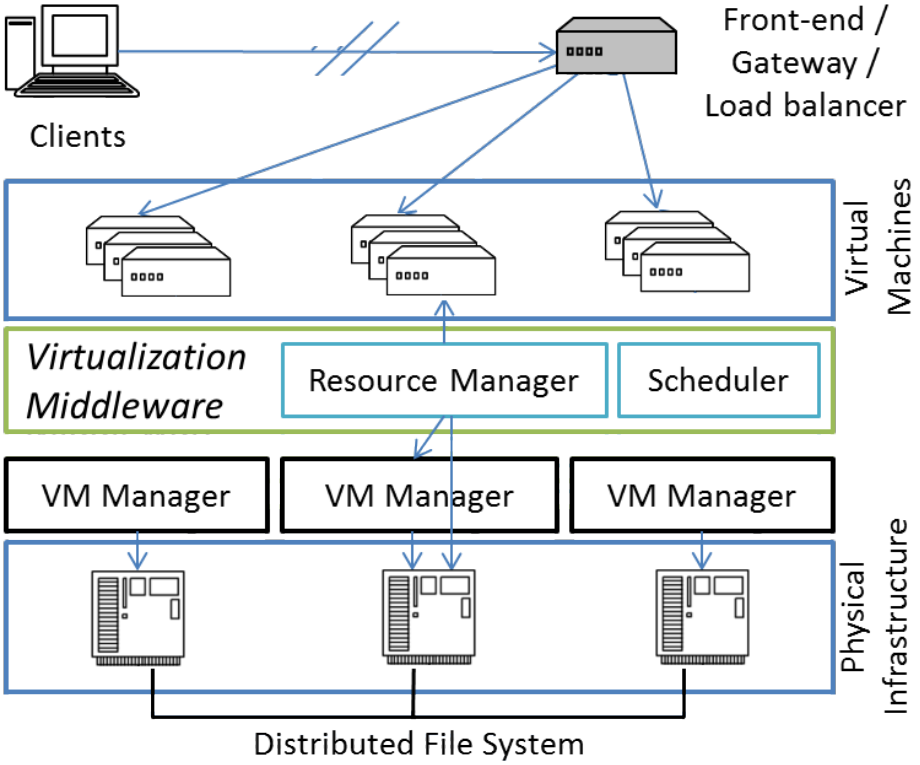


Figure 2.2: Virtualization middleware schema

the current amount of datacenter emissions but the fact that these emissions are increasing faster than any other carbon emission. For this reason nowadays there is a growing interest in "Green" datacenters and supercomputer centers [72].

In this area, the research community is being challenged to redesign datacenters, adding energy efficiency to a list of critical operating parameters that already includes service availability, reliability, and performance. A large variety of power-saving methods has been presented in recent literature. Two of the most representative ones, namely workload consolidation and turning off spare servers, have been shown as an effective way to save energy. Server consolidation implies combining workloads from separate machines and different applications into a smaller number of systems (e.g. Figure 2.3). This approach solves some interesting challenges: less hardware is required, less electrical consumption is needed for server power and cooling and less physical space is required. Intelligently turning off of spare servers that are not being used is an obvious way to reduce both power and cooling costs while maintaining good performance levels.

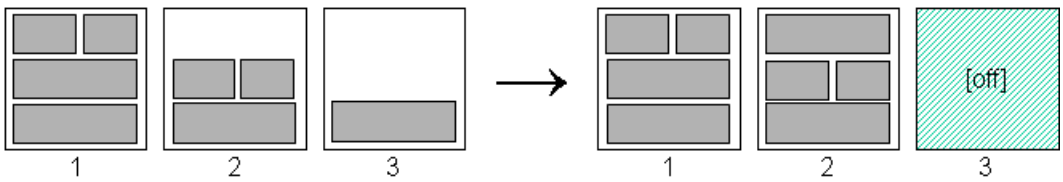


Figure 2.3: Consolidation Strategy

2.2.1 Green Computing Techniques

Power management in cluster-based systems is an emerging topic in the resource management area. As explained in previous chapters, there are several works proposing energy management for

servers that focus on applying energy optimization techniques in multiprocessor environments, such as Lefurgy et al. [97] and Bianchini et al. [33]. Another proposal for load balancing for power and performance optimization in this kind of environment can be found in Pinheiro et al. [129]. Economical approaches are also used for managing shared server resources in e.g. Chase et al. [44], where authors use a greedy resource allocation distributing a web workload among different servers assigned to each service. This technique demonstrates to reduce server energy usage by 29% or more for a typical Web workload. B.G.Chun et al. [47] proposes a hybrid datacenter architecture that mixes low power systems and high performance ones.

Dynamic Voltage/Frequency Scaling (DVFS) is one of the techniques that can be used to reduce the consumption of a server and minimize the total energy expenditure, as seen in Lee et al. [96] and Horvath et al. [81]. Other works such as the presented by Elnozahy et al. [59] propose adding other power management mechanisms to DVFS, as the dynamic turning off of idle machines when the workload is low.

Khargharia et al. [92] introduce a theoretical methodology for autonomic power and performance management in e-business data centers. They optimize the performance/watt at each level of the hierarchy while maintaining scalability. The authors opt for a mathematically-rigorous optimization approach that minimizes wasted power while meeting performance constraints. Their experimental results show near 72% savings in power as compared to static power management techniques and 69.8% additional savings with the global and local optimizations. Petrucci et al. [125] developed a mixed integer linear programming formulation to dynamically configure the consolidation of multiple services/applications in a virtualized server cluster. The approach is power efficiency centered and takes into account the cost of turning on/off the servers. This turning on-off technique is also applied in by Kamitsos et al. [88], which sets unused hosts to a low consumption state in order to save energy; also in energy concerned frameworks like GREEN-NET [51], where hardware components are set-up/shut-down, driven by the schedule of jobs to be run.

The advantages of consolidation using virtualization were pointed out, e.g., Vogels et al. [170], while e.g. Nathuji et al. [112] widely explored its advantages from a power efficiency point of view. This idea is also shown in Petrucci et al. [125], which proposes a dynamic configuration approach for power optimization in virtualized server clusters and outlines an algorithm to dynamically manage the virtualized server cluster. Following the same idea, Liu et al. [102] aims to reduce virtualized data center power consumption by supporting VM migration and VM placement optimization while reducing the human intervention. Based on these works, Goiri et al. [69, 70] introduce the SLA-factor into the self-managing virtualized resource policies. Other works presented by Verma [166, 167] also propose a virtualization aware adaptive consolidation approach, measuring energy costs executing a given set of applications, also dealing with uncertainty, where statistic methods based on correlation are used to predict usage and so consolidate works. They measure SLA but do not take it directly into account when consolidating tasks.

Using heterogeneous workloads leads to SLA's where some of the applications in the system can have stringent conditions to be met. There have been several proposals into resource capacity planning and dynamic provisioning issues for QoS control (e.g. [15, 138, 141]). Chen et al. [45] states that new power saving policies, such as DVFS, or turning off idle servers can increase hardware problems as well as the problem to meet SLAs in this reduced environment. Also, Fitó et al. [62] show that decreasing the number of on-line machines obviously decreases power consumption, but impacts the ability to service workloads appropriately, so a compromise between the number of on-line machines and energy saving must be found.

Filani et al. [61] offer a solution that includes a platform resident Policy Manager which monitors power and thermal sensors and enforces platform power and thermal policies. They explain and propose how the PM can be used as the basis of a data center power management solution.

We want to indicate that other works focusing on green computing and related techniques will be introduced at the proper time in next chapters, comparing their approaches to the ones presented in each work of this thesis.

2.2.2 Green DataCenters

Despite the optimization techniques for reduce energy consumption on datacenters, another way to reduce the impact of computing on environment is to reduce the "brown energy" consumption (energy generated from non-renewable sources). Focusing on the availability of "green energy" (energy obtained from renewable sources) when building datacenter networks, is a way to contribute on green computing, building the datacenters as independent of "brown-energy" as possible. As a result of increasing societal awareness of climate change, governmental agencies, non-profits, and the public at large are starting to demand cleaner datacenters. Multiple companies have announced plans to build "green" datacenters (DCs at least partially powered by renewable energy that they themselves generate). I.e. Apple is building a 20MW solar array for its North Carolina datacenter [54], whereas McGraw-Hill has just completed a 14MW solar array for its datacenter in New Jersey [55]. However, the rate of adoption of these sources of energy in datacenters has been limited by their perceived high cost.

As far as we know, at this time no other works have considered the placement of exclusively green powered datacenters at global scale. The closest related work is Goiri et al. [115], which presents a framework to place Internet services oriented datacenters, characterizing different US regions including costs of building, costs of utilities and distances to the end-user cities, looking for placing the datacenters in an intelligent way saving in costs. Other works considered datacenter placement by ranking potential locations based on operating costs, but without considering all the relevant costs involved on datacenter deployment [2, 123, 149].

Works like Stewart et al. [150] focuses on how energy policies can manage the usage of renewables (wind energy in their case) fitting datacenter workloads to wind energy production. In that work they put emphasis in wind energy intermittent behaviour and the need of policies to hold or move load according to the green energy production.

Considering load management depending on green energy availability, some works presented approaches to schedule properly the deployment of load depending on the predicted amount of available solar energy, like GreenHadoop [71] and Liu et al. [103]. The workload schedule also considers the energy consumed by the cooling systems and the quality of service offered by datacenters.

After datacenter placement, strategies must be applied to reduce the impact of brown or maximize the usage of green energy. To that effort works like Le et al. [94] study policies from exploiting datacenter with different variable electricity prices to the usage of datacenters near green energy sources. Further, focusing explicitly on green energy, the strategy of "following the renewals" has become a relevant topic on energy and computer science, and works like Lin et al., Liu et al., Zhang et al. [99, 104, 182] present studies of algorithms to solve geographical load optimizing the usage of renewable energy depending on availability, constrained by budgeting, latencies and switching cost.

Finally, in the works presented by J.M.Pierson [127, 128] we see a detailed relation of elements to be taken into account for green task allocation, as counting on all energy-consuming elements, the sources of energy and the fluctuation of energy availability on time.

2.3 Relevant Concepts on Artificial Intelligence

2.3.1 AI and Intelligent Management

Even though adaptive and updatable mechanisms have been developed to optimize Cloud management, knowledge-based and data mining techniques are being applied to improve the resource usage and improve quality of service. As the multi-DC systems are becoming more complex and application requirements are increasing, Artificial Intelligence (AI) and also Machine Learning (ML) can be applied to deal with problems difficult to solve, predictions and information retrieval, letting the system to make these decisions with more autonomy in a treatable time.

To help with this *intelligent* management, several techniques and science fields are available. The first ones are standard AI-based applications, using prediction and heuristic algorithms in order to anticipate the system performance and act in consequence. Using generally expert knowledge to tune the heuristics, it includes fuzzy logic, genetic algorithms and other AI space-

search methods. The second ones are the Machine Learning-based applications, using observed behaviors to create models that fit best to key behaviors, letting the system to create its predictors and decision makers without so much expert knowledge.

This thesis is focusing on the application of machine mearning techniques in the decision processes of the control loop in autonomic computing. The main works and solutions done in this joint of fields are promising approaches proposing applications and solutions for specific problems including the idea of supervised learning, reinforcement learning, and other data mining or artificial intelligence methods.

2.3.2 Machine Learning Techniques

Although ad-hoc and holistic methods have been created to self-manage processes and executions on the Cloud, the system is complex enough to have available experts defining the system piece by piece, also operators capable to attend each event on the whole system covering all the problems of uncertainty, lack of information or even worse, with noisy information. A very relevant solution to solve the problem of modeling the system is Machine Learning, a subfield of Data Mining in charge of learning models of very complex systems from examples and observations of the same system, in an easy way to update the model, deciding the model to be enough general or specific, also once the model is built the architects and operators may use the learned description of the system to understand it better, finding weaknesses or flaws, or discovering new details or techniques to improve it. Further, these models can provide classification or prediction for the information we ofter require to drive our system. We base our work in a ML hypothesis:

Hypothesis 1. *For each situation, we can have a model obtained by careful expert modeling and tuning better than any ML-learned model. But, for each situation, ML can obtain semi-automatically a model which is as good as or better than a generic model built without intensive expert knowledge or intensive tuning work.*

The usual procedure for applying machine learning methods is to collect examples of the system (observations from the monitors, snapshots of the system status, etc.), extract from these examples the attributes that most represent the system and can provide information towards the classification or regression the model must learn, create the model that explains the system from this set of attributes and examples, and from this model make predictions we can use to make decisions.

Machine Learning techniques can be divided into supervised learning (like classification and regression), unsupervised learning (discover the relationship between the input data), clustering (find clusters from the examples), reinforcement learning (select the best decision from the past experiences feedback). Here we focus on the supervised and reinforcement learning techniques, directly applied to drive autonomic system.

Supervised Learning

Supervised Learning is the task of inferring a model from labeled training data, this data consisting in a set of training examples. Each training example consists on a set of attributes defining an observation or status of our system to model, and an output value. The learning algorithm should create a model (a classifier or a regression function) relating an observation to its output value, for any valid input, and generalize enough to predict unseen examples with a minimal error. Figure 2.4 shows the basic schema of supervised learning.

The usual methodology for creating and testing models is the following: 1) Two (or more) sets of data are used, the first one (or all sets except one) is used to create the model, while the other set is used to validate it; 2) Observing the results, the model is selected as "good enough", or the modeling algorithm is tuned or the data is treated, in order to train again the model; 3) When a "good enough" validated model is found, a test is done using a new set of data, not seen or used before; 4) Observing the test results, the model is accepted, or the process must begin again, not using this test data as new test data next time. This process ensure that any model is not tested with the same data involved in the creation of the model.

The usage of supervised ML in self-management is quite extended as inference and prediction can be a substitution for any ad-hoc function or model, but here a few examples are related.

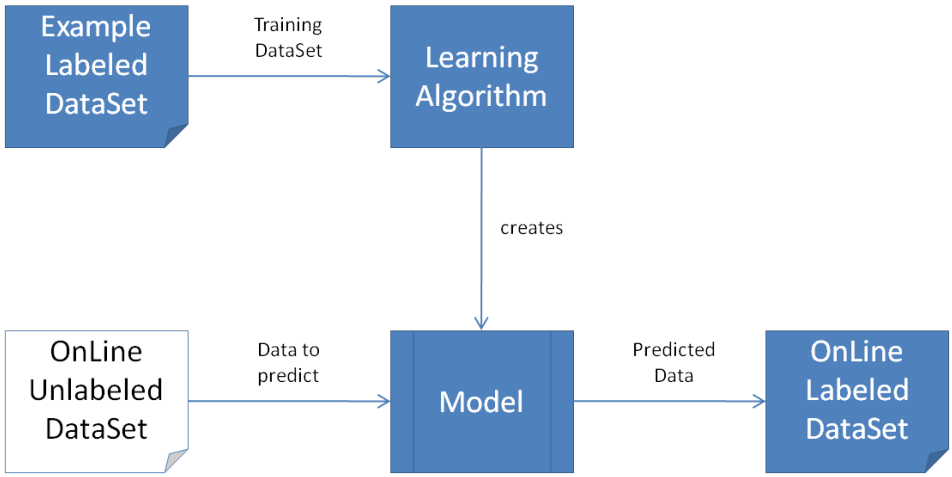


Figure 2.4: Inductive Learning Schema

Works like the PUNCH framework [89] presented a instance-based learning, regression and nearest neighbor algorithms are used to model and predict application performances, in order to be able of allocating or scheduling the application in a Grid environment. Detecting failures in resources and applications, and also discovering the root of failures have become a very interesting area. Hofer et al. [78] presents an application-specific fault diagnosis based on indicators, symptoms and rules, using a supervised classification to find the reason of the failure, and clustering techniques to find what failures are result of the same cause. Other works presented by Cherkasova et al. [181] and Alonso et al. [6, 5] focus explicitly on regression functions to detect memory leaking, focusing on web-service applications also proposing techniques for detecting the leaking component. These supervised learning techniques are usually combined with macro-policies and utility functions. Works presented by Poggi, Moreno et al. [132, 110, 134] show an scenario where, depending on e-shopping user modeling predictions, it is decided which users are kept in the system and how, to reduce the number of machines are kept on-line or shut down, saving energy while preserving the QoS for users providing benefit to the e-shop owner. Also works presented by Wildstrom et al. [175, 174, 173] expose an approach for on-line hardware reconfiguration using the Weka Toolkit [76] algorithms for rules and decision taking.

2.3.3 Relevant Machine Learning Algorithms

The machine learning algorithms we are using along the works presented this thesis are mainly supervised learning algorithms, some of them for classification and others for stistical regression. In order to select the proper algorithm at each time, we first look at the kind of value to predict: if it is a qualifying value (a class or descriptive value) we are classifying, otherwise it is a value indicating "a quantity of something" and we are doing a statistical regression.

Machine learning algorithms have different complexity levels and properties, also their tuning and working methods are really different among them. Deciding which machine learning algorithm we choose to solve a given problem does not rely only in which works better on it. For a regression problem we start from the "simplest" algorithm, e.g. a Linear Regression method, and if results are not favorable or good enough for us, we test more complex algorithms expected to be more powerful and capable of learning. When a complex algorithm returns the same results as a simpler one, we select the simpler one.

Here we present some of the algorithms we use along this thesis:

- Linear Regression: this algorithm returns the linear function that best fits given a set of "explanatory variables" (our data attributes) with a "dependent variable" (the attribute to be learned). Provided a set of inputs shaped as $\langle x_1, \ldots, x_n, y \rangle$, the algorithm finds the linear function $f(\vec{x}) = c_0 + c_1 \cdot x_1 + \ldots + c_n \cdot x_n = \hat{y}$ that minimizes the sum of the squared error of each example instance $(\hat{y} - y)$.

- Regression Trees: algorithms like M5P and RepTree are decision tree algorithms that return a value. These algorithms build a tree from a set of input instances of explanatory and dependent variables, describing a set of rules for input instances are classified and provided a quantitative value (for RepTrees) or a linear regression (for M5Ps). This is, a RepTree is a decision tree with values on its leaves and an M5P is a decision tree with linear regression on its leaves, built using information gain/variance methods (attributes in the tree are set according the amount of uncertainty they solve). The M5P algorithm is specially useful when modeling piecewise functions (as each leaf can represent a segment of such functions) or non-linear functions that can be approximated by piecewise functions.

- Nearest Neighbors: this algorithm assumes that the input instances provided represent reality as is. The algorithm memorizes the example instances with explanatory and dependant variables, and when providing a new instance to predict its dependent variable, the algorithm looks for the k most similar memorized examples. When classifying, it returns the most common class among those k nearest neighbors, may be weighted by neighbor distances. When predicting, it returns the most common value, the average value, the weighted average value, ... among those k nearest neigbors.

- Naïve Bayes: this algorithm is a classifier based on the Bayes rule, and assumes that explanatory variables for each instance are statistically independent among them. Given a set of input instances with explanatory and dependant variables, it computes the probability of each explanatory variable value conditioned to each class, the probability of each explanatory variable value, and the probability of an instance to belong to a class. Using the Bayes rule, given a new instance with explanatory variables, the algorithm computes the probability of belonging to a class conditioned to the explanatory variables, and returns the rank of classes by probability (so we can assume the instance belongs to the most probable class).

When applying a machine learning algorithm we follow a training/validation process. First of all we use a training dataset to create a prediction/classification model, and it is tested using a second dataset different of the training dataset. According to this test, we tune the algorithm until we consider the obtained model returns proper predictions. Then we apply a testing dataset (different from any other dataset seen or used before) over the selected model to see if the model is valid for new unseen data.

For prediction models we evaluate the results by looking at the mean absolute prediction error (or the mean squared error) and standard deviation of prediction errors, so we know how well the model is predicting and how disperse are the prediction errors. For classification models we build a matrix with the values of classification as $\langle Class, ClassifiedAsClass \rangle$: this matrix (called "Confusion Matrix") shows the *Accuracy* as the instances classified correctly (sum of the matrix diagonal); also we can compare the instances of a given class against the instances of such class correctly classified (known as *Recall*), and the instances classified as a class against the instances of such class correctly classified (known as *Precision*).

In our works here presented we are using the implementation of such algorithms from the WEKA Toolkit [76] and the R [80] Statistical package.

Chapter 3

Previous Experiences with Machine Learning and Web-Services

The contributions detailed in this chapter are a set of preliminary experiences with machine learning and web-service modeling, which allowed us to see the connections between autonomic computing and machine learning, gain experience in the field, and sketch the goals of this thesis. They constituted the Diploma of Advanced Studies of Toni Moreno (2008), the PhD thesis of Javier Alonso (2011), the forthcoming PhD of Nicolás Poggi, and our own undergraduate project and Master in Sciences degree thesis. Focusing on the idea of self-management, we did some work to solve real autonomic computing problems like web-service user admission and balancing, around the concept of starting-up and shutting-down hosting machines on a economic saving policy, or denial of service protection and reaction policies. All of this using machine learning models and predictions, willing to automate the modeling and adjusting process. These works have contributed in finding the way of how an "intelligent" autonomous controller component should be structured. All these works are summarized in this chapter, explaining the most important details and some valuable results.

The first works [109, 131, 133, 135] focus on applying machine learning on user modeling and behavior prediction, and user admission macropolicies. Here we have a set of web-service hosting machines that must be set up or shut down for cost saving, being driven by a prediction of the web-service client intentions (buyers, not buyers or automated bots). The cost of setting up or shutting down a hosting machine is given by an utility function, calculating the cost of taking each action, while predicting the number of users willing to buy inside the system.

Other preliminary works focused on self-protection [32]. In particular, introducing a novel method for distributed systems to defend themselves against overwhelming distributed denial of systems (DDoS). The main problem in this scenario is the data collection and reaction mechanism, as all the information to recognize attacks is spread all over the whole network. Also the system have to react in front of the attacks in a distributed way, as the components reacting are routers, gateways and firewalls along the network.

Finally, a collaboration work [3, 4] focused on self-healing, applying machine learning to predict resource exhaustion due to software bugs causing degradation. The main problem in this scenario is to predict the time a web-service will run out of memory due to an unpredictable memory leaking, so it can be rebooted in proper time.

3.1 Our Previous Works on User Modeling

An approach to solve the web-site infrastructure scaling, counting on cost reasons and infrequent client peaks, is presented in [109, 131, 133, 135]. The approach proposes to learn models for

anonymous web user behavior in a real and complex website by learning from observed data. The model is used to support decisions regarding the allocation of web-services users into hosts, by admitting and discarding them depending on their predicted intentions, balancing the benefits of user admissions and the costs of required resources to hold all the admitted clients. As a proof of concept, to model the web-service users we focus on the features related to the visit and actions taken by this one, that make him/her more likely to realize a purchase in the web-site. A user that presents a behavior pattern oriented towards our goals (i.e. realize a purchase) must have priority in being kept inside the system in case of service overloading. In this work we took as metric the completed sessions that end in purchase.

For this we developed a framework prototype, named AUGURES, implementing the model learning and the policies application to admit or discard users. Also we tested it in simulation experiments performed from the weblogs from the servers of a high-traffic online travel agency "Atrapalo.com" [18]. The experiments show that using AUGURES to prioritize customer sessions can lead to increased revenue in at least two situations: the first one, when overload situations occur; that is, the incoming transaction load exceeds the site's capacity and some sessions will have to be queued, redirected to a static site, or dropped; for this study, these should be mostly non-buying sessions, while we try to admit most buying ones. The second one is when keeping a server running has a quantifiable cost; in this case, one could try to group buying sessions a small number of servers, possibly shutting down those other servers that would produce little or no revenue.

3.1.1 The AUGURES Prototype

In this subsection we proceed to describe the architecture of the prototype AUGURES. Currently AUGURES has two subsystems: an offline component (the *learner*) that obtains the historical logfile and produces a predictive model or *predictor*; and a real-time component, the *selector*, implemented as a service that runs along the session manager of the firewall. The selector analyzes the incoming requests, runs them through the predictor, and outputs the priority along with other static information for the session. These two subsystems are presented graphically in Figure 3.1.

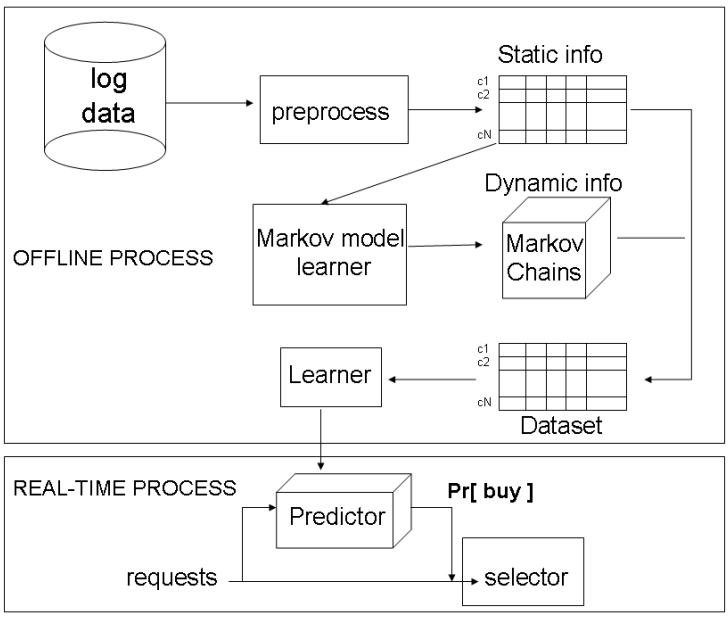


Figure 3.1: AUGURES architecture

The input for the offline component is the logfile produced by the web-site logging module, containing non-ambiguous session and page actions/tags as historical data, which is first cleaned and reorganized by a preprocessor. The preprocessor produces an intermediate file with one

line for each transaction. These lines are largely computed independently from each other, so they do not contain information about the user navigational pattern; that is why we call the information in this file *static*. Next, this file is enriched with *dynamic* information reflecting the user navigation sequence, relating the different transactions of the same session. This is done by computing a Markov model for each value of the class, in our case buying and non-buying; the prediction of these models for each individual request is added as extra information to each line of the preprocessed file. Finally, this enriched dataset is passed to a learning module that produces a *predictor*, some mathematical function that, given a request, assigns it a buying probability. More details are given in the following subsections.

The real-time component, the selector, runs side-by-side with the session manager of the firewall. When an incoming HTTP/S request arrives, the selector reads the entry produced by the firewall, retrieves from a database existing information about the user and session (if any), and then evaluates the request through the predictor, the (offline built) model. The result is a predicted probability of purchase, which is written to the firewall's active session table along with other useful information such as: current load, server conditions, and enterprise policies. This information can then be used by the firewall (in ways outside the scope of this paper) to prioritize and even discontinue some of the sessions according to the current load and policies. We remark that the firewall is *not* a part of AUGURES: it is often a complex, and very sensitive, part of the infrastructure so we do not aim at replacing it. AUGURES, however, provides additional information to the firewall which helps it in taking informed decisions rather than blind or random ones.

In contrast to the selector, which has a real-time requirement, the offline component can be executed at scheduled intervals to rebuild the predictor (daily, weekly, etc.) at periods of low load, and even in an off-site machine. Therefore the requirements of speed and low memory use are not a limitation for this component, while the real-time part needs to be as efficient as possible.

The Preprocessor: Generating Static Information

The goal of the preprocessor is two-fold: First, it should clean the logfile of static content i.e. images, CSS, javascript or other media files. It should also be cleaned of irrelevant and non-user-initiated transactions, such as AJAX autocomplete controls, background checks and offsite requests via web services (Business-to-Business communication). The second goal is to add information that cannot be derived from the logfile only, such as background information on previous sessions and, if available, user details form the company's customer database.

The preprocessor reads the log and produces one output line for each input transaction, producing a dataset relevant to learning containing the following fields:

- *Date* and *time* as a timestamp
- *The tag*, action performed by the page or non-ambiguous URL
- Whether the user has already *logged in* the system during this session
- Whether the customer is a *returning customer*, retrieved from cookies or matching IP address
- Whether the customer has *purchased in the past*, and if so how far back
- *Session length* so far, in number of transactions (clicks)
- The *referer* tag, the tag of the previously visited page, this is an external page for the first click of each session
- The *class* assigned to this session, that is, what the "correct" prediction should be for this log entry, In our case, there are two class values: buyer and non-buyer

Note that all fields except for the class can be computed from information in the previous entries of the log, or from separately stored information. The class, however, can only be computed

by looking *forward* in the same session, and checking whether it contains any tag indicating purchase. Clearly, this is not possible in the online process, since this information is precisely what we are trying to predict. Thus, the class can only be computed in datasets with past information, those used for offline learning.

Generating Dynamic Information

We use the information obtained from the user's navigation sequence as the *dynamic information* of the session; it is the sequence of URLs followed by the user. Unfortunately, most machine learning algorithms are not well adapted to dealing with variables that are themselves sequences. In AUGURES we propose to use high-order Markov chains to address this issue.

A Markov chain describes (is) a probability distribution on the set of all finite paths along a finite set of states S. In general, for a path $s_1 s_2 \dots s_n$ and any probability distribution we have the following rule

$$\Pr[s_1 s_2 s_3 \dots s_n] = \Pr[s_1] \cdot \Pr[s_2 \mid s_1] \cdot \Pr[s_3 \mid s_1 s_2] \cdots \Pr[s_n \mid s_1 \dots s_{n-1}].$$

For general distributions these probabilities can be all distinct. The assumption in a kth order Markov chain is that, given any previous history, only the k most recently visited states affect the future transitions, formally

$$\Pr[s_n \mid s_1 \dots s_{n-1}] = \Pr[s_n \mid s_{n-k} \dots s_{n-1}], \quad \text{for } n - k \geq 1.$$

As an example, in a Markov chain with $k = 2$ the rule above simplifies to

$$\Pr[s_1 s_2 s_3 \dots s_n] = \Pr[s_1] \cdot \Pr[s_2 \mid s_1] \cdot \Pr[s_3 \mid s_1 s_2] \cdot \Pr[s_4 \mid s_2 s_3] \cdots \Pr[s_n \mid s_{n-2} s_{n-1}].$$

Therefore, a k-th order Markov chain is described by giving, for each state $s \in S$ and path p of length at most k, a probability that the next state is s given that the k last visited states are those in path p. This implies that the distribution given by the Markov chain can be specified by giving at most $|S|^{k+1}$ numbers, rather than infinitely many.

Furthermore, given a set of data consisting of paths along S, one can build a kth order Markov chain that approximates their distribution as follows: compute all the empirical probabilities $\Pr[s_{i+1} \mid s_1 \dots s_i]$ for $0 \leq i \leq k$ on the data. By the discussion above, these figures are enough to approximate $\Pr[p]$ for each path p of every length. Of course, whether the figure computed in this way approaches the real probability of p in the source of the data depends on 1) the amount of training data available (the more data, the better approximation), and on 2) the degree to which the Markovian assumption is valid for the source.

In our methodology, we define the set of states S to be the set of tags in our log data. Then, for some parameter k, we create a k-th order Markov chain for each of the classes, each one modelling the typical sequences of tags (requests) for that class. In our case, we train two models: one for buyers and one for non-buyers. Given the path followed in the current session, these two chains can be used to compute probabilities $\Pr[p \mid \mathit{buyer}]$ and $\Pr[p \mid \mathit{nonbuyer}]$, where p is the sequence of previous k tags in the session. Using Bayes' rule, we can then estimate the converse probabilities $\Pr[\mathit{buyer} \mid p]$ and $\Pr[\mathit{nonbuyer} \mid p]$. For example,

$$\Pr[\mathit{buyer} \mid p] = \Pr[p \mid \mathit{buyer}] \cdot \Pr[\mathit{buyer}] / \Pr[p]$$

where we approximate $\Pr[\mathit{buyer}]$ as the fraction of buyers in the data, and $\Pr[p]$ can be ignored because what matters really is the *ratio* of $\Pr[\mathit{buyer} \mid p]$ to $\Pr[\mathit{nonbuyer} \mid p]$. That is, given that the user has followed this path, the Markov chains guess the probabilities that later in the future s/he buys or does not buy. At training time, these two figures (the *buying* and *non-buying* probabilities) are added as new variables to the line describing the current transaction in the training set. At prediction time, these two figures are added as new variables to the information passed to the predictor for the current transaction.

We have used $k = 2$ (second-order Markov chains) for the experiments reported in the next sections. After some experimentation, this value seemed to provide the best results ono our attemps. It is intuitively clear that remembering the last two visited pages gives more information

than remembering only the last one. On the other hand, as k grows, each individual path is less frequent in the data, the approximations of the probabilities are coarser, and predictive accuracy is reduced (i.e., overfitting tends to appear). This effect is especially harmful on buying paterns which are rare on our datasets. In particular, $k = 3$ gave results comparable to $k = 2$, and predictions were significantly worse for $k > 3$. This conclusion may, of course, be different in other contexts.

Learning Module

The resulting sequence of transformed and enriched log entries can be treated as a dataset where the order of examples is irrelevant and each example is a tuple of simple values (numerical or categorical values). In this first prototype we have chosen the Naïve Bayes classifier as a learning algorithm [76], for a number of reasons: 1) it is easy to understand, has no user-entered parameters, and has very low CPU time and memory requirements, both for training and for prediction; 2) in preliminary experiments [132], it performed about as well as more sophisticated methods, such as decision trees and boosting; and 3) it assigns probabilities to its predictions, rather than hard buy/non-buy decisions, and this is essential for our prototype. Naturally, there is ample room for trying other and more advanced prediction methods in later versions, which administrators can choose according to their data and available resources.

3.1.2 Experimental results

We tested the method using a dataset consisting in transactions collected over approximately 5 days (3.7 million transactions). We consider a transaction a user-initiated action (click) to the site that he/she views as an atomic operation. To log user actions only, the dataset was produced by the web-server logging mechanism; additional code was added at the end of each executing script to log the transaction data after actions were executed. By doing so, the data is already cleaned and more accurate. A session is considered as a sequence of transactions initiated by a user in a definite timespan. We also discovered that some transactions are produced by automated bots, i.e. crawlers or web fetching form other sites, of course never ending in purchase. We kept them in the dataset as it is important that our system learns to identify these as non-buyers: since search queries to B2B providers have a cost and the bots could be abusive or even malicious, they should be assigned low priority or denied access. The relevance of B2B communication could be studied afterwards, as it could be relevant to our web-service.

After building a classifier using the training dataset, we compute for each transaction in the testing set a "true" buying/nonbuying label and a "predicted" label. Our ultimate goal is to use these predictions for prioritizing sessions, so that low priority sessions can be queued, redirected to a static page, or even dropped when the server is under heavy load condition. In our case, since we are using the Naïve Bayes classifier, we have good control over the %admitted quantity. Indeed, this classifier provides a probability of buying $p(t)$ for each transaction t. Set some threshold value $T \in [0, 1]$, then we can decide to admit those transactions t such that $p(t) > T$. By increasing T, we will make it more difficult for a transaction t to pass this test, hence we will admit less transactions. Conversely, if we lower T, more transactions will be admitted. Once the Naïve Bayes classifier is built, we use the training set to tabulate the function of T to the actual %admitted, for future use.

A set of results was obtained applying the learned Naïve Bayes classifier (containing the Markov models prediction) on the testing dataset (Figures 3.2). *Recall* represents the fraction of real buyers that are admitted by the predictor, while *precision* is the fraction of predicted buyers. There is a nontrivial relation between %admitted, recall, and precision. As we become more restrictive in the number of admissions, we loose true customers, but at a rate smaller than if we were choosing at random. Example given, if we choose to admit 50% of the transactions, AUGURES will still admit 91% of those that will end in purchase (rather than 50% as we would if we were selecting them randomly).

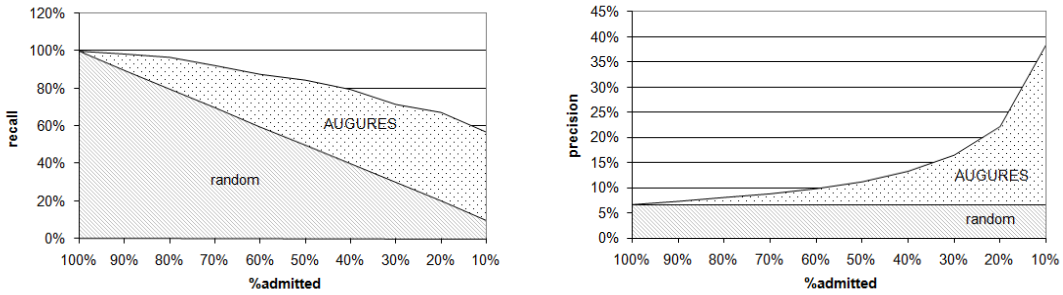


Figure 3.2: %admitted vs. recall and %admitted vs. precision

3.1.3 Conclusions for User Modeling

When a peak situation occurs most infrastructures become stalled and throughput is reduced, and to prevent this, load admission control mechanisms are used to allow only a certain number of sessions, but as they do not differentiate between users, users with intentions to purchase might be denied access. In this approach, we took data from a high-traffic online travel agency and learned to predict users' purchasing intentions from their navigational patterns.

In our experiments we are able to train a model from previously recorded navigational information that can be used to tell apart, with nontrivial probability, whether a session will lead to purchase from the first click. The maximum number of allowed users to the site can be regulated, according to the infrastructure's capacity and goal specification, by placing a threshold over the predicted buying probability of incoming transactions. That is, the model can adapt itself dynamically to the workload while maintaining reasonable recall and precision.

The relevance of this work towards the current thesis is a first approximation towards decision making at web-service and hosting machines load distribution, minimizing the costs of maintatining resources online while maintaining the revenue from clients given an exceptional overloading situation. Also we got a first experience on applied machine learning, that although being also included in the thesis of N.Poggi, the DEA of T.Moreno and also our own undergraduate project, it is a relevant work done for this thesis as a guide to frame the next works done here.

3.2 Our Previous Works on Self-Protection

In our work [32], we presented an approach to look towards the utility of machine learning classifiers inside a self-controlled system and the capability of recognizing online network patterns. Also this work had the component of *cooperation* among the distributed classifiers, so all the information and activities involves a set of components that must be aggregated and coordinated. This work is an extension of previous work by Zhang and Parashar [179], but including the machine learning component to classify traffic as attack.

For this case of study we took Distributed Denial of Service (DDoS) flooding attacks as example. The distributed mechanism presented in the work gathered information from each important point of the system, sharing and using it to discover patterns with machine learning techniques, that let the network stop and avoid a distributed attack, abuse, or flooding. And what is more important, the mechanism let the network configure the traffic classification system the best as possible by itself, using this example-based learning. Each element involved in the mechanism computed its own local probability of abnormal behaviours and abnormal network status, shared it with the neighbours, aggregated the received information to its own one, and classified the received traffic with all the obtained knowledge. The method let each element learn about the behaviour of its portion of network, adjusting its classifiers to its location in the network and its usual traffic.

In that work we showed that an *intelligent* autonomic system needs a part of data obtaining, a trained predictor able to classify the system status and some judgement to act in front the

prediction. The relevance of this work towards the thesis relies in the fact that we can complement the system information with new predicted one using machine learning, often required to know in more detail the status of the system. Once it helps we can apply prediction mechanisms to complement monitors and system statistics, as we will see in next chapters.

3.2.1 Distributed Data Collection

In front of distributed attacks, self-protection requires knowing what is happening along all the system and network. While other kind of attacks and failures can be detected in single points in the network, DDoS uses distribution techniques not only to launch a more powerful attack but also to be undetected on any specific point, firewall, or bordergate node. To know that an attack is going on, information must flow through the affected network area to warn all the affected points. In this case, the required data must be collected from all the intermediate network nodes, informing of abnormal behaviours and circulating the required feedback among them, letting each element of our intermediate network have a local and a global vision about what is happening. This information-sharing mechanism is composed by an overlay network intercommunicating the intermediate nodes, and a detection mechanism able to record all the signals of attack. These signals are transported with the complementary data captured from the network to the prediction module in order to determine whether an attack is ongoing.

Sharing the Obtained Information

Our scenario is composed by a network, with bordergate nodes, intermediate nodes, and services to be protected. The data to be collected must be obtained from all the network and for that an overlay network is used for passing messages with the shared information. This overlay network joins the most important nodes we want to use for the detection and packet filtering. The selected nodes are equipped with the detection and classification capabilities, and are the responsible to share information about abnormal behaviours and also information about previous detections of attack made by another node.

Detecting Abnormal Behaviour

In order to detect the malicious traffic, the monitors must identify first changes on the traffic, such as “amount of traffic towards a given node”, “relation between amount of traffic sent/received to a particular service”, or others depending on protocols, message headers, or specific behavior patterns. In that work we used only the first, amount of traffic towards a given node. For the particular case of distributed flooding attacks which, by definition have “large amount of traffic” as the only common feature, we worked specially using the variation on the amount of traffic received towards a specific victim. We used CUSUM algorithm to determine when the traffic towards a particular node changes significantly.

For that current setting, once a node is declared to be under attack, it is impossible to distinguish the packets belonging to the attack from the legitimate packets still flowing to it, because we define an attack not by the kind or intention of the packets arriving to a node but by its number. In particular, it could happen that our system totally blocks the well-intentioned traffic directed to e.g. a site that has become suddenly popular and sees an increase in the number of visitor. In this sense, the method was not strictly speaking an attack-filtering one but rather a method for protecting sites from dangerous overloads.

Prediction for Specific Traffic

After a sample of traffic is recorded, the information is used to classify that traffic at each node. We collected the information from the same node and the aggregated information from other nodes. For our approach we used the well-known Naive Bayes method as the classifier method, which declares when an attack is occurring from the local information and the one obtained from the neighbours. Also we used decision trees and feature selection algorithms in our first tests in order to check the relevance of the obtained information, and see how useful was is aggregated

information. However, we eventually decided to use the Naive Bayes method because it obtained better accuracy and the method is simpler anyway.

3.2.2 Reaction and Feedback

Once threatening behaviour is detected locally (this is, on a single node), the traffic classified as an attack is stopped and this information is shared among the neighbour nodes, letting them aggregate this information with the one-self collected. An abnormal increase of traffic happening in a single node can easily be result of a heavy connection but if this increase is repeated in other nodes of the network simultaneously, the aggregation of this increase on the victim could become a threat.

Details of the Reaction

The reaction in front of the attack is the result of evaluating, using the predictor, the two kind of messages sharing the information: gossips, indicating suspicion of a flooding attack, and warnings, indicating a high possibility of being under attack. Gossips transmit the knowledge information about the attack, and warnings transmit the feedback of other nodes that also detect and confirm the attack. Also, while gossips are sent to and received from neighbors in any direction of the overlay network, warnings are sent only in the direction of (apparent) attackers, in order to strangle attacks as close to the source as possible.

So, when a detector (the CUSUM algorithm for traffic volume towards a particular victim) considers that traffic to a victim is increasing in an abnormal way, a gossip is spread among the neighbors indicating that a victim can be under attack. In one extreme, all neighbors receive the gossip, and in the other extreme, only the next-step-node receive the gossip. The message contains the possible victim ID and the confidence, that is, the number of gossips received referring to the same possible victim. Also, when a classifier determines that a victim is under attack while classifying a message destined to it, a warning is sent to node from which the message is received (i.e., towards the source of the attack), and sent too to the neighbours if required. Each node uses the aggregated gossips and warnings received as inputs to its classifier.

In other words, the idea of this scheme is the following: With the warnings, victim-closest nodes indicate to the bordergate nodes to stop the attack flow, and with the gossips, the source-closest nodes indicate to the intermediate network nodes that the aggregated traffic is still attempting to enter the network, instead of letting the attack flow in. The result should be both to improve the accuracy in classification, but also, when an attack is detected, to stop undesired traffic as close to the sources as possible, thus freeing resources in the network.

3.2.3 Summary of the Node and Network Algorithm

The detection algorithm at a node compares the accumulated sum of means for each time unit with a threshold that may be different for each destination. When a traffic packet arrives, the accumulated mean for its destination is updated and if it reaches the given threshold, the *confidence value* for that destination increases, and the node sends gossips containing this confidence value to p neighbours. Each node, at each time unit, accumulates self-confidence values and neighbour confidence values. Also, for each time unit (empty time units included) means and accumulated means are updated according the last time-unit statistics.

Also, for each message received the node applies its classifier to the message attributes, adding the aggregated information from gossips about confidences for the destination, and the alerts about the source, from warnings. If the classifier concludes that the message belongs to an attack, the message is not forwarded and also a warning was sent to the backward node. The evaluation algorithm is provided in Algorithm 1, and the schema for the information spreading is shown in Figure 3.3.

The threshold for a destination node should be such that it indicates the maximum amount of traffic that the node can reasonably handle. This number can vary wildly among nodes. For example, current routers can process several million packets per second, while a few thousand, or even a few hundred, packets per second may be enough to damage a server if these packets

Algorithm 1 Message evaluation algorithm in each node

```
for each received message do:
    if accumulated mean > destination threshold then:
        gossip[destination] ++
    end if
    if time unit changes then:
        if gossips[destination] > 0:
            send (destination,gossip[destination]) to neighbour node
        end if
        update the traffic mean and accumulated mean
        clean gossips and warnings
    end if
    evaluate line <message, gossips for dest, warn for src>
    if classified as attack then:
        warning[source]++
        send (source,warning[source]) to backward node
    else
        forward message to destination
    end if
end for
```

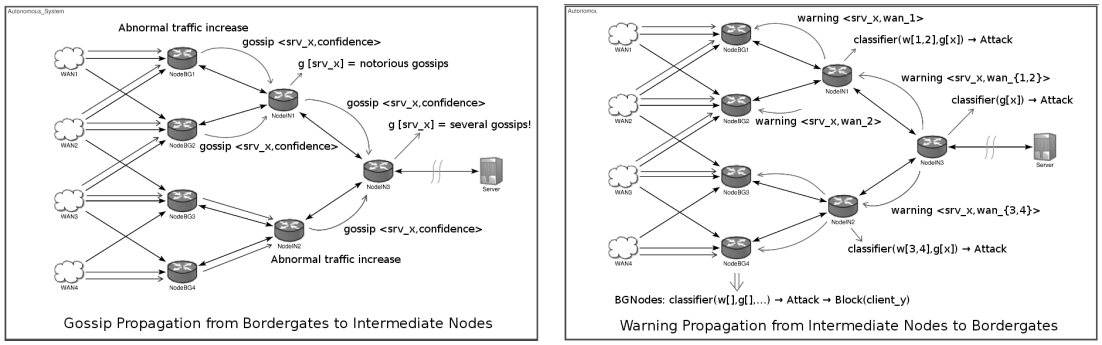


Figure 3.3: Detection and Reaction Mechanism

correspond to expensive requests. Thus, we expect these thresholds to vary by several orders of magnitudes among nodes in a real network.

Experimentation Results

We tested the mechanism with ad-hoc scripts and more realistic simulations, using different ratio attacks and different threshold for the detection algorithm. We performed experiments assigning all nodes the same threshold and also assigning internal nodes (routers) a threshold 100 times larger than terminal nodes (servers). We also wanted to check that our mechanism is able to detect both massive-scale attacks and small-scale ones. We have thus run the experiment with an attack that is 100 times the normal volume of traffic with the victim and with an attack that is only 2 times the normal volume of traffic with the victim. The possible dangers were, in the first case, that still too many packets filtered through the network and reached the victim and, in the second case, that the attack is confused with normal traffic and is undetected. The results in Figure 3.4 show the four combination of these two values for these two parameters, and show that these two dangers are actually avoided.

The number of gossips and warnings vary depending on the threshold set for the CUSUM algorithm. When giving the intermediate network a higher threshold value, the closest nodes to the server detected the major part of the attack and sent gossips when they were really overwhelmed. The intermediate nodes detected the attack guided by the closest nodes when

Classified as → Really ↓	Legitimate	Attack
Legitimate	20000	0
Attack	300	20000

(a) Attack Ratio = 2; Same Thresholds

Classified as → Really ↓	Legitimate	Attack
Legitimate	22800	0
Attack	0	21000

(b) Attack Ratio = 2; Interm. Thresholds = Threshold * 100

Classified as → Really ↓	Legitimate	Attack
Legitimate	250000	0
Attack	21000	13500

(c) Attack Ratio = 100; Same Thresholds

Classified as → Really ↓	Legitimate	Attack
Legitimate	284000	0
Attack	2000	13500

(d) Attack Ratio = 100; Interm. Thresholds = Threshold * 100

Figure 3.4: Confusion Matrix on Attack Ratios and Thresholds

they received these gossips and warnings, causing less false negatives. The classifiers on the intermediate nodes also learned to react with less gossips in the intermediate network. We observed that the intermediate nodes have higher accuracy and less false negatives when having a higher detection threshold than the closest nodes. Also we tested higher threshold ratios (>100), but the number of gossips in the intermediate network is so reduced that some classifiers only memorize the attack sources for the training, invalidating the learning process; this problem can surely be solved by a slightly more involved learning algorithm. As a detail, we observes that there are no false positives. As we cannot detect, by now, a legitimate request to the victim from an attacker during an attack, we did the tests without introducing this kind of connections.

Finally, we observed that the Naive Bayes classifier achieves high accuracy (above 95%) and all classification mistakes are false negatives, that is, some attacking messages are let through. We observe no false positives when no attack is occurring, all traffic arrives to the destination service. Moreover, most of the mistakes in one node are corrected in the next node in the path to the victim.

3.2.4 Conclusions for Self-Protection

In this work we saw how to complement a previous work of flooding detection (intrusion detection system) towards a mechanism to detect and prevent flooding (intrusion prevention system), using machine learning mechanisms. The mechanism is able to stop all flooding traffic close to the source, freeing the intermediate network of flooding traffic and preventing service overload during the attack. All in all, the network blocked flooding traffic with accuracy above 95%, with no false positives. The false negatives (attacking messages not classified as such) at the bordergate nodes were stopped in the intermediate network, but far enough from victim service.

The data used here is as simple as the aggregation of counters all around the network, but the useful contribution was the use of this distributed information recollection for train and run the classifiers and have knowledge of the system status from all the points of the network, react in front of the bad behaviours, and also use the information of the action to reinforce all the decision elements of the system.

The conclusion of this work for this thesis is that machine learning methods can provide improvement to complex and distributed systems, as the system is able to provide information of its status or the information about the load it must hold. This work also was presented as our Master thesis.

3.3 Collaboration Work on Self-Healing

In our work [3, 4] we presented an approach on applied machine learning towards predicting the time to crash for a web-service, due to memory leaking. This is another example of how we can apply machine learning to predict consequences on a web-service environment, letting us time

to react in front of an unwanted situation, like degradation of the quality of service or crash of the service itself. In this work, we focus on a software aging prediction model based on M5P algorithm and its evaluation in front of a varied and complex software aging scenarios, predicting the resource exhaustion time.

3.3.1 Software Aging Scenario

Because system complexity is growing day by day, the number of failures due (directly or indirectly) to this complexity has also been growing, resulting in undesirable behaviors, poor levels of service, and even total outages. The need to prevent or gracefully deal with outages of businesses and critical systems is clear, given the industry huge loss due to the downtime per hour. Studies like [158, 58] reported that one of the causes of unplanned software outages is the software aging phenomena: the accumulation of errors, usually provoking resource contention, during long running application executions,causing applications/systems hang or crash [74]. Software aging has been observed in web-servers [13], spacecraft systems [151], and even military systems [105], with severe consequences such as loss of lives.

Software rejuvenation, a set of strategies to deal with software aging, can be divided into Time-based (rejuvenation is applied regularly and at predetermined time intervals, e.g. [161]) and Proactive/Predictive-based strategies (system is continuously monitored and action is triggered when a crash due to software aging seem to approach). If we can predict the crash and apply rejuvenation actions only when required, we can reduce the number of rejuvenation actions. Traditionally, software rejuvenation has been based on a restart of the application/system or even a whole machine, but we can also micro-reboot [39], i.e. rebooting only the suspicious web-service component. Predicting the time until resource exhaustion due to software aging is far from easy, as its consumption may not be linear or change along time. It could be related to the workload or be undetected due to granularity used to monitor resources. Even it could happen due to two or more resources simultaneously involved in the service failure [41].

A lot of effort along modeling resource consumption has been concerned with capacity planning and predict resource exhaustion. In [48] Tree Augmented Naive Bayesian Networks are proposed to determine which resources are most correlated to performance behavior, in performance analysis and post-mortem analysis of the causes of Service Level Objective (SLO) violations. In [180], Linear Regression is used to build an analytic model for capacity planning of multi-tier applications, showing how it offers successful results for capacity planning and resource provisioning, even under variable workloads. In [160], authors use a semi-Markov reward model using the workload and resource usage data collected from the system to predict resource exhaustion in time. In [98], authors use time-series ARMA models from the system data to estimate the resource exhaustion due to workload received by the system. In [11] evaluation of three well-known ML algorithms: Naive Bayes, decision trees and support vector machines to evaluate their effectiveness to model and predict deterministic software aging. These previous works did not treat software aging changing with time, did not keep in mind to learn which resource is involved in software aging, or did not test their approach against a dynamic and more than one resource involved in order to evaluate the effectiveness of their approach.

Modeling and Prediction Assumptions

This work proposes to use ML to predict time until failure caused by software aging. Due to the complexity of modeling these growing complex environments and with low knowledge a priori about them, we decide to build automatically the model from a set of metrics easily available in any system like CPU utilization, system memory, application memory, Java memory, threads, users, jobs, etc. Among many ML algorithms and models available we have chosen to use the M5P algorithm, included in WEKA package [76], a decision tree with linear regression on its leaves. The rationale is that while a global behavior may be highly nonlinear, it may be (or approximate by) a linear-per-parts function. This may well be the case for many system behaviors of the kind we want to analyze, where the system may be in one of a relatively small number of phases, each of which is essentially linear. A preliminary comparison of the M5P and other regression algorithms like Linear Regression or Decision Trees on this scenario was presented in [5].

The model is trained and validated using samples of different executions that finished in crash. An important detail is that the model predicts time until failure if the state of the system and workload do not vary in the future. If the situation changes the model has to be able to recalculate the time until failure under the new circumstances. As variables we not only use direct metrics but also derived ones to achieve a more accurate prediction. Some variables representing the consumption speed of resources are computed using sliding window averages, collecting the last n observations and calculating their averages, to smooth out noise and fluctuation. This n is a certain trade-off and it must be set by considering the expected noise and the frequency of change in our scenario.

We conducted a set of experiments to evaluate the effectiveness of the approach for complex software aging scenarios. The prediction accuracy is shown used the Mean Absolute Error (MAE). However, predicting exactly the time until failure is probably too hard even as a baseline, so we also used as a accuracy indicator the Soft Mean Absolute Error (S-MAE), that counts accuracy not from the real value but from a margin around 10% of the real time until crash (named security margin). As the moments when crash is near imminent are more important towards the rejuvenation mechanism, we have calculated the MAE for the last 10 minutes of every experiment (POST-MAE) and for the rest of experiment (PRE-MAE). The idea is that our approach has to have lower MAE in the last 10 minutes than the rest of experiment, showing that the prediction becomes more accurate when it is more needed.

3.3.2 Prediction Experiments

Experimental Setup

The experimental environment simulates a real web environment, composed by the web application server, the database server and the clients machine. We have used a multi-tier e-commerce site that simulates an on-line book store, following the standard configuration of TPC-W benchmark [157], with Tomcat 5.5.26 servlets [14] and MySQL 5.0.67 [111], on a 4Core Intel XEON 1.4 GHz with 2 GB RAM. TPC-W allows us to run different experiments using different parameters and under a controlled environment, allowing us to conduct the evaluation of this approach. TPC-W clients, called Emulated Browsers (EBs), access the web site (simulating an on-line book store) in sessions, a sequence of logically connected requests from the EB point of view, with proper thinking times between requests. In our experiments we have used the "shopping distribution" of the TPC-W set of workloads.

We have introduced changes into the TPC-W to simulate software-aging errors consuming memory or java threads. Random memory consumption is produced by an altered servlet (TPCW search request servlet) computing a random value between 0 and N, determining how many requests will handle the servlet before the next memory leakage is injected. This makes the variation of memory consumption to depend of the number of clients and the frequency of servlet visits, so with high workload our servlet injects memory leaks more often, and with low workload the injection becomes low. Thread injection is produced by calculating two random values, one between 0 and M indicating how many threads are injected, and other between 0 and T indicating the time until next thread injection. This injection is independent of the workload.

System information and used variables include elements from the load like the current throughput, requests load, response time per request; elements like the cpu/mem/IO usage and characteristics; further elements from the java virtual machine like the java heap usage and properties; also elements from the Tomcat web-service like the number of threads, opened connections and mysql connections. In order to train the model, the time to failure is labeled into the training and validation observations.

Deterministic Software Aging Experiments

The first experiment was to evaluate M5P to predict the time until failure due to deterministic software aging. We decided to inject a 1MB of memory leak with N = 30. We trained our model, generated using M5P, with previous 4 executions with 25 EBs, 50EBs, 100EBs and 200EBs, becoming 2776 instances. The M5P model generated was composed by a tree with 33 Leafs and 30 inner nodes, using 10 instances to build every leaf. The four training experiments were

executed until the crash of Tomcat, to let the M5P to learn the behavior of the system under a deterministic software aging. Finally, to evaluate the accuracy of the model, we evaluated the model built with these four experiments using two new experiments with different workload (75EBs and 150EBs). Table 3.1 shows the results obtained. We can observe how M5P obtains better results than simple linear regression due because it handles better the trend changes due to the Heap Memory Management actions, even when we do not add the specific information.

	Linear Regression	M5P
75EBs MAE	19 min 35 secs	15 min 14 secs
75EBs S-MAE	14 min 17 secs	9 min 34 secs
150EBs MAE	20 min 24 sec	5 min 46 secs
150EBs S-MAE	17 min 24 secs	2 min 52 secs
75EBs PRE-MAE	21 min 13 secs	16 min 22 secs
75EBs POST-MAE	5 min 11 secs	2 min 20 secs
150EBs PRE-MAE	19 min 40 secs	6 min 18 secs
150EBs POST-MAE	24 min 14 secs	2 min 57 secs

Table 3.1: MAEs obtained predicting time until failure on deterministic software aging

Dynamic and Variable Software Aging Experiments

Our next experiment was to evaluate our model to predict progressive but dynamic software aging under constant workload. We trained the model with 4 different executions (1710 instances): one hour execution where we did not inject any memory leak and three executions where we injected 1MB memory leak with constant ratio (N = 15,N = 30 and N = 75 in every respective execution). The model generated was composed by 36 leafs and 35 inner nodes, using 10 instances to build every leaf. It was tested with an execution with injection ratios changing every 20 minutes. MAE and S-MAE measures are adjusted to the moment injection rate.

In the experiments predicting time vs. Tomcat memory evolution during the execution, we observed that while there is no injection, the predictor determines that the time to crash becomes "infinite" (actually it returns "3 hours" periodically, standing for "very long time to crash"). When injection begins, the Tomcat memory starts decreasing gradually and predicted time to crash decreases. Important details are noticed, like some anomalies appearing in memory consumption, as the Heap Management interferes spontaneously with the memory occupation. Further, the sliding window introduces some delay in trend change detection.

Another detail found is when memory is almost depleted. Then, there is a moment where prediction is not quite accurate, as the injection rate is so slow the model has trouble detecting it, and it keeps the prediction almost constant. Furthermore, when the java memory heap is almost full, the heap manager starts more often cleaning rutines interfering with our monitored values. When the heap manager stops interfering, the predictor reacts again reducing the estimated time until crash. The MAE and S-MAE obtained in this scenario was 16 min. 26 secs. and 13 min. 3 secs. respectively, which we believe is a quite reasonable accuracy. On the other hand the PRE-MAE and POST-MAE were, respectively, 17 min. 15 secs. and 8 min. 14 secs. The experiment was running for 1 hour and 47 minutes.

Software Aging Hidden within Periodic Pattern Resource Behavior

Our next experiment was to evaluate the model in front of a deterministic software aging masked by a periodic pattern of memory acquisition followed by memory release. The experiment is similar to the previous one, but now we control the periodic memory acquisition and release to a 20 minutes phase, leaking memory at each one,so a crash is bound to happen after several periodic phases. The workload is set constant with 100EBs, and the injection phase follows N = 30 and release phase follows N = 75, allocating and releasing 1MB each time.

This experiment showed us that the M5P model should be retrained after applying a more detailed feature selection, following the conclusion extracted in [79], as the variables used for previous models were now irrelevant and the model should focus directly on java heap monitor

variables only. The new model was formed by 17 inner nodes and 18 leafs. For this scenario, the time to crash is linear and the Java Heap memory is being consumed and released in every phase until exhaust the memory resource. M5P can manage the periodic pattern, unlike Linear Regression which obtains worse results. However, we have to point that in this case M5P has problems to be accurate in the last 10 minutes of the experiment.

	Linear Regression	M5P
MAE	15 min 57 secs	3 min 34 secs
S-MAE	4 min 53 secs	21 secs
PRE-MAE	16 min 10 secs	3 min 31 secs
POST-MAE	8 min 14 secs	5 min 29 secs

Table 3.2: MAEs obtained predicting time until failure on software aging hidden within periodic pattern

Dynamic Software Aging due to Two Resources

Next step was to consider aging caused by two resources simultaneously, memory and threads. An important point in this experiment is that creating java threads have impact over the memory java heap, relating the two causes of aging.

The model for this experiment was trained with several executions at different constant workload and injection rates (N = 15, 30, 75, M = 15, 30, 45 and T = 60, 90, 120, so 6 executions, 2752 instances), but in all of them only one resource involved in the execution. The model generated was composed by 35 inner nodes and 36 leafs. The MAE and S-MAE, PRE-MEA and POST-MAE obtained by M5P in this experiment was: 16min. 52secs., 13min. 22secs, 18min. 16secs. and 2min. 5 secs. respectively: this is about 10% error (MAE and S-MAE), given that the experiment took 1 hour and 55 minutes until crash. We can observe how the M5P is able to predict with great accuracy the time to crash, when it is near, and most importantly, the model never was trained using executions where both resources were injecting errors simultaneously. These results show the promising adaptability of this approach to new situations not seen during training.

An interesting point to remark is that when inspecting the models generated by the M5P algorithm at each experiment, some clues about the root cause of failure can be drawn. Relevant variables are in relevant positions in the decision tree, while important threshold values on the system are learned to make decisions over these variables. So, interpreting the models generated via ML models has an additional interest besides prediction.

3.3.3 Conclusions on Learning Towards Self-Healing

In this work we proposed a machine learning approach to build automatically models from system metrics to detect and predict software aging and being able to decide when to trigger self-healing actions. ML is used due to the complexity of the resources behavior and the complexity of the environment. We studied the effect of software aging errors which gradually consume resources until its exhaustion, in a way that cannot be attributed to excess load. Our approach is oriented towards feeding our modeling algorithm and predictor with a set of monitored and derived variables on memory status and web-service load, where the most important variables resulted to be the consumption speed of the relevant resources. All of this smoothed out using average sliding windows of instantaneous measures. The M5P predictor is evaluated in different, complex and dynamic software aging scenarios, showing how M5P obtains acceptable accuracy. and ability of adapting scenarios never seen during training. Finally, we suggested that interpreting the models generated by M5P can help to determine the root cause software aging.

This work, although being also included in the PhD thesis of Javier Alonso, provided experience on the field of prediction applied towards decision making, also the importance of feature selection depending on which scenario and resource we are focusing on. Also this work provided interesting details on web-services memory behavior, to be used for future memory modeling and

prediction in the following thesis works, driving virtualized web-services. Further, this works remarks the idea of creating human-readable models to help system operators and architects to improve their systems. A very important detail not only at web-service level but at any level.

Chapter 4

Tailoring Jobs and Resources in Clusters and DataCenters

As long as virtualization has been introduced in datacenters, it has been opening new chances for resource management. This chapter exposes the work done proposing a manual modeling from expertise and scheduling policy for virtualized datacenters, which mainly focuses on the multiple facets of VMs and datacenter nodes (energy efficiency, virtualization overheads, and SLA violation), optimizing the provider's profit. The work presented in this chapter is a co-work with Íñigo Goiri, mainly presented as part of his PhD Thesis, and it serves as base and starting point on this thesis research, as this approach presents models and decision policies made manually from expertise. In this chapter we report on work that is, on the one hand, mostly our contribution and, on the other hand, a direct basis for the forthcoming chapters, unlike the research presented in Chapter 3 which are side tracks.

4.1 Introducing the Resource Modeling Approach

Datacenters must be able to deal with new challenges: new virtualization capabilities [146], economic and social pressure to reduce their energy consumption [102], and offer high availability [19] and performance to their users bound to SLAs. All in all, while being economically profitable. In this chapter we see a way of modeling a virtualized datacenter, focusing on the allocation of VMs in host machines, from its different facets representing benefits and costs. All derived from revenues of running virtualized jobs and the costs of running them according to requirements and quality of service, looking for optimizing the provider benefit. We take into consideration revenues for running HPC and web-service jobs and workloads, and we consider running costs for energy consumption, VM operation overheads, penalizations for SLA violations, outsourcing of resources, etc.

The modeling proposed here is an analytical way to provide managers and schedulers a way to compute benefits and costs of performing operations over the virtual machines and hosts on a datacenter. As experts on virtualization, we detail a set of features and issues we find in the VM execution and management, important details we consider are relevant for improving benefit or enforcing service level agreements. After modeling these datacenter features, we design a scheduling method to attempt to find the best operations to be performed over VMs (basically allocation and migration decisions).

Related Work

As explained in the previous Background Chapter 2, energy efficiency is a hot topic addressed in cluster-based systems [126], treated by different kind of strategies like DVFS or turning on/off machines depending on workload requirements, also depending on trade-offs between perfor-

mance and power consumption [47]. Following these ideas, our approach is able to take profit of homogeneous or hybrid datacenter architectures.

Works like [125] propose a dynamic configuration approach for power optimization in virtualized server clusters, and outlines an algorithm to dynamically manage them. Their approach also takes into account the cost of turning on or off servers. Nevertheless, the approach can lead to a too slow decision process as it is focused in a very static environment and highly focused on HPC jobs. In this sense, our proposal is more suitable for an on-line scheduler.

VM migration and VM placement can be used to reduce virtualized datacenter power consumption [102]. Following this idea, we propose the use of VMs for executing heterogeneous applications taking into account virtualization overheads. However, virtualization makes the overall system more complicated and requires well-designed policies which take VM management problem into account. Until today, virtualization management overheads have been only considered individually when managing virtual resources [145, 12].

We remark that previous works only take into account individual factors in terms of managing a datacenter, while our contribution in this work is an integrated and complete solution for managing virtualized datacenters.

4.2 Ad-Hoc Modeling of Virtualized DataCenter

The main idea of the model presented here is to calculate all the costs and revenues for a given schedule, deciding whether an allocation of a VM in a given execution platform will provide a benefit or not. This platform is usually a local host of the datacenter which can execute VMs, but it can refer to assets of an external provider (i.e. outsourced) or any other kind of resources. Nevertheless, from the model point of view, all the execution platforms will be seen as a *host* with different costs and features.

Having a model that involves several factors and measures, it is usual to have different measuring units representing their values. Power related values are expressed in watts or watts per hour in case of energy, economic revenues are expressed in currency units, timing variables are expressed in time units, and the penalties and weights we define to enforce policies are usually non-dimensional values. All these variables should be unified in a same unit, as our goal is to optimize benefits from a single benefit function:

$$Benefit(h, vm) = Revenue(vm) - \sum_{i}^{i \in Costs} Cost_i(h, vm) \tag{4.1}$$

The benefit of placing a virtualized job vm in a host h is the revenue from the VM minus the sum of costs derived from having the VM deployed at h. This benefit function represents a currency unit. Power and energy can be converted to costs by applying the cost of watt per hour, the revenue of the VM can be expressed as the revenue of CPU per hour, the penalties can be converted also to costs using a penalty to cost function, agreed by the service provider and the clients and owners of VMs, etc. The following Table 4.1 shows the list of symbols, factors and functions involved on our datacenter modeling. After it we explain with more detail each one of them.

4.2.1 Time References

The execution time for a job in a dedicated machine $T_d(vm)$ is used as a reference for the scheduler, and it will be used for billing and calculating the final revenue according to the SLA. This means that, in case the execution of vm goes far beyond this time, it will violate the SLA terms. The extra time added due to operations or virtualization overheads is represented as $T_{extra}(vm)$, the estimated remaining time as $T_{rem}(vm)$, and the estimated remaining time plus extra time $T_r(vm)$. Other times are the time of operating a virtual machine vm in a given host h as $T_c(h, vm)$ and $T_m(h, vm)$ (creation and migration time), and the aggregation of these times as $T_op(h, vm)$.

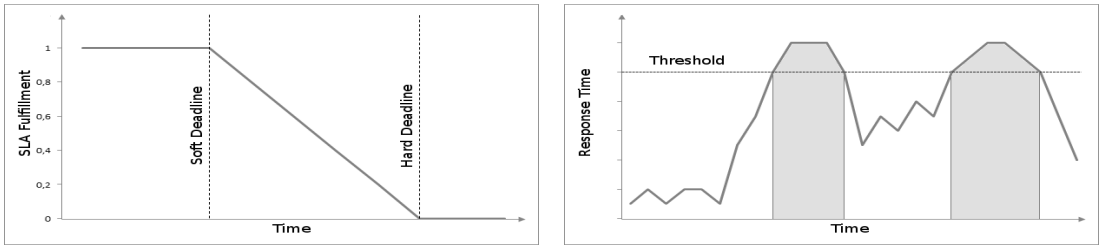
List of Time references	
$T_d(vm)$	vm execution time in a dedicated machine
$T_{extra}(vm)$	Extra time added to vm
$t(vm)$	Time since vm submission
$T_{rem}(vm)$	vm remaining time: $T_d(vm) - t(vm)$
$T_r(vm)$	vm remaining time including virtualization overheads $T_{rem}(vm) + T_{extra}(vm)$
$T_op(h, vm)$	Extra time over vm produced by operations in host h
$T_c(h, vm)$	Time to create vm in host h
$T_m(h, vm)$	Time to migrate vm to host h
List of Factors	
$Host(vm)$	Host where vm is allocated
$R(vm)$	Revenue to be obtained for running vm
$Pr_{hour}(vm)$	Price to be paid by customers per vm per hour
$He(h)$	*Health*: Ratio between available resources and required resources at h
$Speed(h, vm)$	Speed at which a vm job will perform at a host h
$Perf(h, vm)$	Performance of a vm placed at host h
$Occup(h)$	Occupation of host h
$Power(h, o)$	Power consumed by a host h given its occupation o per hour
List of Penalties	
$Pen_{DL}(h, vm)$	Penalty for SLAs based on deadlines, for vm in host h
$Pen_{Perf}(h, vm)$	Penalty for SLAs based on performance, for vm in host h
$\hat{Pen}(h, vm_1, vm_2)$	Estimator of Penalties for vm_2 in host h when placing vm_1
$Pen_{mig}(h, vm)$	Time Penalty for migrating vm to host h
$Pen_{virt}(h, vm)$	Time Penalty for operating over vm in host h
$Pen_{conc}(h, vm)$	Time Penalty for concurrent operations over host h if holding vm

Table 4.1: Summary of symbols and functions

4.2.2 Revenue and SLA Factors

We consider heterogeneous workloads composed by two kinds of jobs with different goals and functionality: HPC jobs and web-services. In both the customer must pay $R(vm)$ for executing a VM during an amount of time $T_d(vm)$ at an agreed pricing $Pr_{hour}(vm)$. Furthermore, each kind of job has a specific SLA with the penalties the provider must pay for not offering the desired quality of service, depending on the application and its performance.

In this work we define the performance of HPC jobs through the fulfillment of the execution deadline, determined by a soft deadline where time starts to penalize and a hard deadline where it reaches the maximum penalty. SLA fulfillment degrades from the soft to the hard deadline, as seen on Figure 4.1a. Also penalties in web-services depend on the performance they get and this can be measured in different terms, such as response time. If response time is used, it will depend on the amount of resources assigned to the web-service and the amount of requests that it receives. A service with not enough resources to satisfy all the requests will start increasing its response time and thus will start violating an agreed threshold in the the SLA (and then degrading its fulfillment). In this work the SLA fulfillment depends on the percentage of time that it has been fulfilling the SLA (see Figure 4.1b).



(a) Deadline (DL) (b) Performance (P)

Figure 4.1: Examples of SLA fulfillment kinds

The performance of the job is evaluated when it finishes its execution by using the SLA terms. Nonetheless, an estimation of this SLA fulfillment can be calculated during jobs execution, to preview the possibility of violating an SLA term when performing operations or selecting locations for a given VM. E.g. when the demanded resources in a host are more than the available resources, they have to be shared among VMs and this can lead to low performances.

We introduce here the concept of *health*, referring to the capability of a host to attend the resource demands of the VMs it hosts. We define a function $He(h)$ as a heuristic relating the amount of available resources and the total demanded ones with the performance. The *health* factor can be used to estimate the availability of resources in hosts and the effects of resource overloading over the jobs, web-services, their deadlines and SLA fulfillments. This factor can be applied with the performance factor $Perf(h, vm)$, to adjust it on a non-dedicated machine. The introduction of this factor and its basic idea is extremely important in this thesis, as in next chapters performance predictors will be modeled automatically using machine learning, instead of doing it manually or ad-hoc like here in this work.

One of the most important costs comes with the penalties incurred by violating SLA terms. As explained before, overcrowding a host can degrade the performance of the jobs running on it, so we defined a *health* heuristic to estimate the impact of placing VMs overloading a given host. Here the $He(h)$ function depends on the total CPU capacity in a host $CPU_{avail}(h)$, and the CPU demand of each VM running on that host $CPU_{req}(vm_i)$.

$$He(h) = min\left(1, \frac{CPU_{avail}(h)}{\sum_{vm_i}^{h} CPU_{req}(vm_i)}\right) \tag{4.2}$$

This function follows a behavior observed in our testbed hosting machines, where the performance is directly related to the ratio of offer/demand of CPU (In next chapters we see how to model this function not only with CPU resources but also with Memory and IO resources). This health factor estimates whether and how the host can handle all its load, and in case of overwhelming how much performance is degraded. We will use it for estimating the execution delays and then the SLA penalties.

4.2.3 Performance Factors

Datacenters are usually composed by hosts with different capabilities and speeds (resource heterogeneity). The performance and the speed of applications are highly variable depending on the speed of the nodes and this influences the levels of SLA fulfillment. This model currently focuses on CPU speed as the provider targets HPC jobs, which directly depend on CPU, and Web-services, which CPU is also in part a bottleneck. However, it can be extended to memory or IO resources.

The model calculates data for each vm (e.g. elapsed times, accumulated extra times, etc), for a referenced dedicated host H_{ref}, with a defined speed as $Speed(H_{ref}, vm)$. A given VM can be executed in a host with different speed regarding the reference machine. To consider this, we define a performance factor $Perf(h, vm)$ for a VM running in a host h as follows:

$$Perf(h, vm) = \frac{Speed(H_{ref}, vm))}{Speed(h, vm)} \tag{4.3}$$

This performance factor, as a ratio of progression speeds, can be used to estimate time variations when migrating a vm from one host to another: $T_d(vm, h) = T_d(vm) \cdot Perf(h, vm)$.

4.2.4 Power and Energy Factors

Power consumption varies depending on the utilization of the system. It needs to be modeled from each kind of machine, according to the load on it and the resource usage. Using the information obtained from measuring the power consumption of a physical machine stressed with different loads, we can model the power consumption of a machine given its current occupation or its resources usage $Occup(h)$ (as number of CPUs used, etc). We represent this model with the function $Power(h, o)$.

4.2.5 Other Factors and Important Issues

Those VMs which are not running in any host, including those that have not been yet executed or previous execution has failed must be held in queue, waiting to be resubmitted to a running host. We represent the queue as a *virtual host* holding that non-running VMs not allocated in any physical machine. As the provider wants to run as many VMs as possible in the shortest time, the stay of VMs in an execution queue has to be minimized, and thus holding a VMs in the queue must be strongly penalized. This makes any other available and profitable allocation better than not running the VM.

Also, each job has its own hardware and software requirements, like system architecture, type and number of CPUs, required basic software, or the hypervisor. These requirements can also vary along time.

Another capability of the datacenter business is outsourcing resources. Outsourcing can suppose an extra solution for an overloaded datacenter, as some resources can be externalized paying the cost of acquiring temporarily resources from other clouds or datacenters.

Finally, virtualization technology has overheads that we must deal with, or at least have them into account. Some of the relevant factors are the VM creation overhead (time to create and deploy the VM), and the VM migration overhead (time to stop and pack a VM, move it to another host, and deploy and run it again). These times are computed as $T_c(h, vm)$ and $T_m(h, vm)$, and are part of the extra operation time. Further, when an operation is being applied over a VM, no other operation can be applied to it. Also performing more than one VM operation in the same host machine (*concurrency*) introduces overhead too, as CPU usage for VM operations.

4.3 Modeling the Costs-Benefit

The functions and factors involved in the cost-benefit for this model can be treated as costs of an allocation, and these costs are subtracted from the revenue, finding the resulting benefit for a host×VMs schedule. Also, some of them represent non-economic units, so penalties must be derived to express each handicap and overhead as a cost or benefit. Table 4.2 summarizes the involved costs, and following this we explain each one of them.

List of Costs	
$C_{req}(h, vm)$	Cost of having available the vm required resources in host h
$C_{pen}(h, vm)$	Cost of SLA violation when placing vm in host h
$C_{enh}(h, vm)$	Cost of energy for vm placed in host h
$C_{ops}(h, vm)$	Cost of operating vm when placing in host h

Table 4.2: Summary of Costs

Cost Availability for VM Requirements

Checking the capability of a host for holding a given VM must be addressed by the scheduler. In case the host is not available to execute that VM, the cost of placing this VM into that host can be considered as the maximum penalty for that job as specified in the SLA and the allocations in that host will not be performed. If the maximum penalty can be *infinity* when we do not have a fixed maximum cost. When placing a VM, unfeasible placements will be discarded because of having an "infinite" cost, while feasible placements will not be penalized. This cost is defined as follows:

$$C_{req}(h, vm) = \begin{cases} \infty & \text{if } h \text{ does not have the resources required by } vm \\ 0.0 & \text{otherwise} \end{cases} \quad (4.4)$$

Cost of Estimated SLA Penalties

Violating SLA terms represents one of the most important costs. For each kind of SLA we must define a penalty, that here we will use as a factor or weight upon the revenue obtained from each vm. As much we violate a SLA, less the owner of the virtualized job will pay for it.

Regarding SLAs based in deadlines (HPC jobs), we consider the extra time accumulated due to operations $T_{extra}(vm)$, and the extra time accumulated due to lower than expected performance, derived from $T_{rem}(vm)$ and adjusted according to the host health $He(h)$ and its performance factor $Perf(h, vm)$. With these two extra times we can figure out how much time this VM will be delayed beyond its deadline.

$$Pen_{DL}(h, vm) = max\left(0, \frac{1hour}{3600sec} \cdot \left(T_{extra}(vm) + \frac{T_{rem}(vm) \cdot Perf(h, vm)}{He(h)} - T_{rem}(vm)\right)\right) \tag{4.5}$$

Regarding SLAs based on performance (web-services), managing SLA threshold violations and thus response times and CPU quotas at each time, is a matter of considering the extra CPU load each operation incurs. We can estimate the penalty for a web application trying to figure out the amount of time it will receive less CPU than required during its remaining execution time $T_{rem}(vm)$, which basically depends on the host health $He(h)$ and its performance factor $Perf(h, vm)$.

$$Pen_{Perf}(h, vm) = max\left(0, \frac{1hour}{3600sec} \cdot \left(\frac{T_{rem}(vm) \cdot Perf(h, vm)}{He(h)} - T_{rem}(vm)\right)\right) \tag{4.6}$$

The SLA for two kind of jobs is based mainly on their *health* and in their *performance* factors, so we can generalize the two penalty functions if we consider that web-services have always a $T_{extra}(vm) = 0$. This cost is defined as follows:

$$C_{pen}(h, vm) = Pr_{hour}(vm) \cdot Pen(h, vm) \tag{4.7}$$

We must recall that placing a VM in a host not only affects its own execution and performance, but also the execution and performance of the other Vms running in the same host (We will need to keep this in mind in next chapters, when learning how a VM placement affects the rest of hosted VMs). Then we should have an estimation function $\hat{Pen}(h, vm_1, vm_2)$ that calculates the penalization of a vm_2 in a host h, when introducing vm_1 on it. This is, if we have the vm_1 placed, how would this affect to vm_2. So when we decide where to place a VM, we will try not to degrade the SLA of previous hosted VMs. Notice that all those penalty estimations return positive values, since there is no reward if the jobs finishes earlier or the web-service performs better when running in a better machine.

Cost of Power Consumption

The consumed energy of each host can be calculated in watts per hour, if we previously modeled the power consumption function of the host, and we know the occupation of it (number of VMs or amount of resources used). if we have the electricity prices for the corresponding host, we can convert the cost into currency units. This cost is defined as follows:

$$C_{enh}(h, vm) = \frac{CPU_{req}(vm)}{\sum_i^{vm \in h} CPU_{req}(vm_i)} \cdot Power(h, O(h, vm)) \cdot Pr_{wph} \cdot T_r(vm) \tag{4.8}$$

Cost of Outsourced External Resources

We also have the capability of using rented resources from other providers when we do not dispose of the required resources and we do not want to stop or reject a job execution. This cost depends on how much the external provider will demand for its services, the overhead of migrating jobs towards it, and the availability of resources in the external provider. Usually external resources are only used when their costs are cheaper than the penalties to be paid if we keep a VM in our datacenter. Considering h_p as a host from an external provider, this cost is defined as follows:

$$C_{rent}(h_p, vm) = Pr_{hour}(h_p, Type(vm)) \cdot (T_d(vm) + T_{extra}(vm)) \tag{4.9}$$

Cost of Virtualization Overheads

The virtualization overhead can be considered as an overhead over VM basic operations, concurrent VM operations over the same VM, and concurrent VM operations over the same host.

Basic virtualization operations Here, virtualization overheads such as VM creation and VM migration affect as an operation time, and these operations can have an estimated or defined time $T_c(h, vm)$ and $T_m(h, vm)$. The cost for migrating a VM must consider the remaining execution since it is not worth to move a VM which will finish shortly. For this purpose we consider a migration penalty depending on the remaining required execution time.

$$Pen_{mig}(h, vm) = \begin{cases} 2 \cdot T_m(h, vm) & T_r(vm) < T_m(h, vm) \\ \frac{T_m(h,vm)}{T_r(vm)} \cdot T_m(h, vm) & T_r(vm) \geq T_m(h, vm) \end{cases} \quad (4.10)$$

We consider here, as a possible example, that the weight of the migration time becomes the ratio of migration and remaining times, so as fewer execution time remains, migrate a VM becomes more penalized. Also we set a limit for situations where migration time exceeds remaining time as two times the migration time.

Blocking VMs when operating over them When an action is being performed in a VM, starting another action on that VM can put the system under not desired situations like migrating VMs when they are not ready yet, or trying to destroy a VM which is being migrated. For this reason, while the VM is being operated it should be locked for other operations. For this purpose we consider a blocking penalty when operating over a VM.

$$P_{virt}(h, vm) = \begin{cases} 0.0 & \text{if } Host(vm) = h \\ \infty & \text{if action performed in } vm \\ T_c(h, vm) & \text{if } Host(vm) = \emptyset \\ P_m(h, vm) & \text{if } Host(vm) \neq h \end{cases} \quad (4.11)$$

Penalizing concurrent operations on the same host When performing a VM operation, overhead is added on the system. Thus, we consider a penalization for operating a vm on a host h where other VMs are being operated.

$$P_{state}(h, vm) = \begin{cases} T_c(h, vm) & \text{if } h \text{ is creating } vm \\ T_m(h, vm) & \text{if } h \text{ is migrating } vm \\ 0.0 & \text{otherwise} \end{cases} \quad (4.12)$$

$$P_{conc}(h, vm) = \begin{cases} 0.0 & \text{if } Host(vm) = h \\ \sum_{vm}^{h} P_{state}(h, vm) & \text{if } Host(vm) \neq h \end{cases} \quad (4.13)$$

Finally, all these penalties can be aggregated in one only penalty for operations allocating a vm in a host h. Also, as these penalties are time units, as they represent extra time to be added towards the execution (for deadline SLA jobs) or time when the job will not run (and then do not provide service, for performance SLA web-services).

$$T_{op}(h, vm) = P_{virt}(h, vm) + P_{conc}(h, vm) \quad (4.14)$$

Converting operation time to cost The aggregation of all the penalties, representing the operational extra time T_{op} to be added to the execution time of the VM, must be converted to currency units as a cost to be computed with other costs and benefits. A VM in operation time implies different costs: the power consumed during this process, and the cost incurred by the possible violation of the SLAs running in that node. In addition, the cost for creating, migrating, or performing any action in a VM affects to all the VM in that host ($\hat{Pen}(h, vm_i, vm)$). This cost is defined as follows:

$$C_{ops}(h, vm) = \left(\frac{Pr_{kwh} + Pr_{hour}(vm)}{3600} \right) \cdot T_{op}(h, vm) + \sum_{vm_i}^{h} \hat{Pen}(h, vm_i, vm) \cdot Pr_h(vm) \quad (4.15)$$

4.4 Scheduling following the Model

The model of costs and benefits presented in this chapter focuses in taking profit of the different capabilities that a virtualized environment offers, to place VMs on a datacenter in order to maximize its benefit. The model includes most of the relevant factors to drive this scheduling process, like job execution, power consumption, resource outsourcing, operational costs and overheads, and enforcement of service level agreements.

The scheduling process must attempt to increase the datacenter resource provider benefit, by placing the VMs on the most suitable hosts, looking for the best profitable combination of VMs $\times$ hosts. Each possible solution has an economic addition of revenues and costs, the benefit function seen in Equation 4.1, result of each placement and its consequences.

Factors involved in the process are included as costs in currency units, more intuitive for clients and providers (as here we look for optimize an economical oriented datacenter management). So the scheduling will be guided by these values, revenues for VMs, feasibility of placements, costs for placing VMs on hosts, and quality of service enforcement represented by the SLA penalties.

4.4.1 Solving Scheduling

Scheduling is represented here by a matrix $Hosts \times VMs$, where each cell represents the placement of a VM in a Host, as a binary value. Note that a VM can not be in two hosts at the same time (we do not consider transitional effects of migration here). Also, to decide the schedule, we use a second matrix, a *scoring matrix*, also $Hosts \times VMs$, where each cell represents the sum of costs and benefits for a VM to be placed in the corresponding cell host. The VMs are intended to be moved to hosts where their revenues minus costs are maximized. Given a scoring matrix, once masked by the corresponding scheduling matrix (multiplying the two matrices through cell by cell values), the sum of all its values result in the total benefit of the scheduling solution.

The optimization mechanism consists in finding iteratively the best VM movements, improving the overall benefit. Initially, we set the current schedule in the scheduling matrix and we compute its related scoring matrix with the current costs of holding VMs in their places and the costs for each VM to be moved to another host. Centering values in zero, by subtracting the cost of not moving a VM to the rest of placement costs for this VM, is an option that could be taken, so all costs would represent improvement if positive or degradation if negative.

Once having the initial matrices, representing the current status, the optimization algorithm should proceed. We iterate over the matrices looking for the highest value of the scoring matrix, representing the best movement of VM to host to be performed, at only one step beyond. Once movement is selected, we update the scheduling matrix and update the scoring matrix given the new movement. Recall that each VM movement affect the other VMs in the origin host and the tentative host. The algorithm iterates until the scoring matrix has no positive values, we reach a defined number of iterations to be performed, or we reach a previously visited status. As there is a chance of not converging and reach a movement cycle of tentative solutions, we limit the number of movements or set up a mechanism for detecting loops on visited status. When one of the finishing conditions is triggered, we assume to have reached a suboptimal solution for the current schedule. Algorithm 2 shows the scoring matrix optimization algorithm.

We use a *Hill Climbing* based methodology to solve the scheduling and scoring matrices. There is no guarantee of convergence to global optimum, let alone in short time it finds a suboptimal solution much faster and cheaper than evaluating all possible configurations. Each step takes to a more suboptimal configuration until there are no better configurations, an iteration limit is reached, or we are evaluating a previously visited configuration. The algorithm complexity has an upper bound of $O(\#Hosts \cdot \#VMs) \cdot C$ since it iterates over the $\langle$host,VM$\rangle$ matrix C times. Also note that in the current case of study, we can perform some tweaks to cut bad or infeasible configurations by taking advantage of some costs and constraints. I.e. the resource requirement constraint can discard a great amount of combinations by marking as invalid cells with incompatible $\langle$VM, host$\rangle$ placements, at the beginning of the algorithm.

Algorithm 2 Scheduling and Scoring Matrix Algorithm

```
/* Method to fill or update the Matrices */
action calculate_score (Matrix M, Matrix S):

    /* Fill cells with revenues - costs */
    for each VM vm and each Host h:
        M[h][vm] <- calculate_revenues(h,vm);
        M[h][vm] <- M[h][vm] - calculate_costs(h,vm);
    end for

    /* Set cell values to improvement | degradation */
    for each VM vm:
        nop <- sum(M[][vm] * S[][vm]);
        for each Host h:
            M[h][vm] <- M[h][vm] - nop;
        end for
    end for
end action

/* Create Scoring and Schedule Matrices */
M <- Matrix [Hosts][VMs];
S <- get_schedule();

/* Fill Matrices */
calculate_score (M, S);

/* Iterate Matrix */
iters <- 0;
while has_positive_values(M) and iters < limit and no_loop(S) do:

    /* Find best movement */
    bestval <- 0;
    for each VM vm and each Host h:
        if M[h][vm] > bestval then:
            <i,j> <- <h,vm>;
            bestval <- M[h][vm];
        end if
    end for
    S[*][j] <- 0;
    S[i][j] <- 1;

    /* Update Matrices and Iterate */
    calculate_score (M, S);
    iters <- iters + 1;
done
```

4.5 Conclusions on Tailoring Jobs and Resources

Virtualization is making providers more profitable every day thanks to its consolidation capabilities and dynamism. Towards making providers more profitable, energy consumption is a critical issue for large-scale datacenters, which hold hundreds or thousands of machines and cooling equipment. To reduce energy dissipation, dynamic job scheduling is beneficial. The work shown in this chapter identifies a real problem and creates a mathematical apparatus to describe a virtualized datacenter and solves the problem from an economic point of view by merging sev-

eral factors, such as hardware and software requirements, SLAs, virtualization overhead, power efficiency, etc. Based on this model it uses a hill climbing methodology to rearrange VMs using power-aware placement and migration.

This chapter provides to this thesis a starting point towards resource and job modeling, where althought being a collaboration and part of the PhD thesis of Íñigo Goiri, many elements here described were oriented towards setting a base for the forthcoming works in next chapters. The introduction of the *health* function and predictors, to estimate performance on tentative actions and schedules, is key for the introduction of machine learning models on decision making towards scheduling. Also the methods for detecting infeasible VM placements, for making explicit the relations between all the involved elements in the scheduling, also AI algorithms to solve schedules, are basic for the mathematical modeling of the datacenter system and its management, as we will see from here on.

Experimentation and evaluation of the models presented in this chapter have been omitted in this thesis, as we consider that what is relevant on this thesis are the model and methodologies themselves. The experiments concerning to the research and publications of this work follow the thesis line of I. Goiri, not this one. The work presented in this chapters has been published in the journal "Future Generation Computer Systems" [66] (2012).

Next chapters will use as a base the idea of modeling, placing VMs on hosting machines while focusing on energy-efficiency, economic costs of actions, quality of service, and incorporating behavior prediction components, through data mining and machine learning techniques.

4.6 Note on Experimental Environments

Departing from the hand-made expert modeling presented in this chapter, next works present the evolution of this thesis changing the models and formulas introduced by experts, knowing the system and jobs behaviors, by semi-automatically learned models. Here we briefly describe the experimental environment and the changes made on it during this thesis.

4.6.1 Energy-Efficient Simulator

In order to test the following approaches and methods we start from a simulated scenario, where we have an expert who knows how the system works (the same person who set the scenario), to a real scenario, where workloads and machines are real and with real and live experiment executions on these machines. The usage of a simulator shows us (as we will see in next chapter 5) that expert algorithms are able to create exact models to manage the system, but when uncertainty and elements out of the human expert scope are introduced into the system, models using machine learning are able to improve generic algorithms. After using the simulated scenario, we switch to a real scenario where we are able to learn from real systems with non-perfect monitors, inaccurate information about jobs and VMs, and web-service client behaviors. Using our methods on a real datacenter environment let us to learn about real web-service client and computing resources behavior, and validate our models and mechanisms managing real virtual machines and serving real web-service requests.

The simulation environment for evaluating the chapters 5 and 6, is a framework mainly oriented towards energy consumption monitoring. As a testbed simulator we use the EEFSIM, our simulation technology based on the OMNeT++ [124] platform. The simulation uses similar techniques as seen in R.Nou [121], but centered on power usage (measured in a real machine) and the scheduling of the different CPUs in the machines. The EEFSIM simulates the execution of HPC jobs on a set of hosts with a set of CPUs each, scheduling internally CPU quantum times and switching jobs inside the CPU. It can start-up, shut-down or reboot simulated machines, and adds booting times and virtualization overheads, parametrized by the expert programmer.

In chapters 7 and 8 we abandon the simulator to implement the method and scheduler directly on a real datacenter environment, with physical machines, virtualized machines, web-services and complete client-server workloads.

4.6.2 Experimental Real Environments

During most of this thesis we are working with high-performance computing architectures, but we change to low-energy consumption computing architectures for the last works here presented. This change is caused by external factors to this thesis, but it has been taken as an oportunity to learn and test our methods on different architectures with very different properties. As a result we will see in next chapters that our methodologies and models are independent to the kind of environment thanks to using adaptable/re-trainable machine learning models, and when changing elements of the environment the only required operation is to re-learn the ML models describing a given job or resource. The semi-automatic methodology for learning models without so much human expert involvement, provided by machine learning, allows us enough generalization to not depend so much on contexts.

For the experiments done in chapters 5 and 6, where an expert should set up the simulator and models, the HPC machines (Intel Xeon 4 Core, 3GHz 16GB RAM) were taken as reference for CPU and power consumption models. These machines were monitored and measured by the experts completing the simulator, making it to behave as a standard 4Core HPC machine. Also, for experiments on real machines in chapter 7, we deployed an OpenNebula virtualization middleware and our scheduler on a set of 10 Intel Xeons 4Core, with an Oracle VirtualBox virtualization platform on them, and a set of virtual machines containing Apache Tomcat web-services, with LiBCN'10 web sites (seen in next subsection).

For the experiments done in chapter 8, we use low-energy consumption machines (Intel Atom 2 Core, 1.6GHz 4GB RAM). These machines consume much less power than the HPC (an order below), but have much less computational power. We deployed the same software infrastructure of the previous chapter on a set of Atom 2Core machines to test our approaches.

4.6.3 The LiBCN'10 Workload

While in chapter 5 we use a standard Grid5000 workload [155], providing a list of jobs to be executed with their execution times and CPU consumption, for experiments involving web-services we introduce a new workload: the Li-BCN Workload 2010 [26]. The Li-BCN'10 Workload is a collection of traces from different real hosted web-sites offering services from file hosting to forum services (deployed on a Apache v2.1 with PHP 5.1 and MySQL) with different input loads for each site and time. These web-applications correspond to a set of customers who run their services and web-sites on top of the provider, in a virtualized environment. The advantage of these workloads against others is that here we have directly the Apache logs, so we can replicate loads exactly with a client emulator, and we have the workbench web-sites corresponding to the Apache logs, to be placed in the (virtual) machines of the testbed.

4.6.4 Experimental Methodology

As said several times, here we are substituting the human-expert models for semi-automatically learned models using machine learning. Our environment (simulated and real) consists in the implementation of a hardware/software stack as shown in Figure 2.2. We set virtual machines with our jobs (HPC jobs or web-services), managed by a virtualization middleware (OpenNebula + our decision maker/scheduler). The important work is done on the decision maker and the information to be communicated between monitors and the decision maker. Figure 4.2 depicts the information flow and elements composing our decision making schema, showing the monitors, the models and expert functions, and the resulting schedule.

The obtained schedule is communicated to the resource manager, in charge of allocating the physical resources to virtual machines. For now, here the decision making is done centralized, as our algorithms take the whole system into account to optimize performances. A machine (or set of machines) compute the scheduling, another machine (if not the same as before) receives the schedule and execute the planed actions according to the schedule communicating each physical machine (and datacenters in multi-DC systems) the orders and actions to be applied individually. This makes that all elements in the system to communicate their monitorized information to a centralized system. A future work beyond this thesis would be how to perform the processes we are deploying here in a more decentralized way, further in multi-datacenter systems, so not all

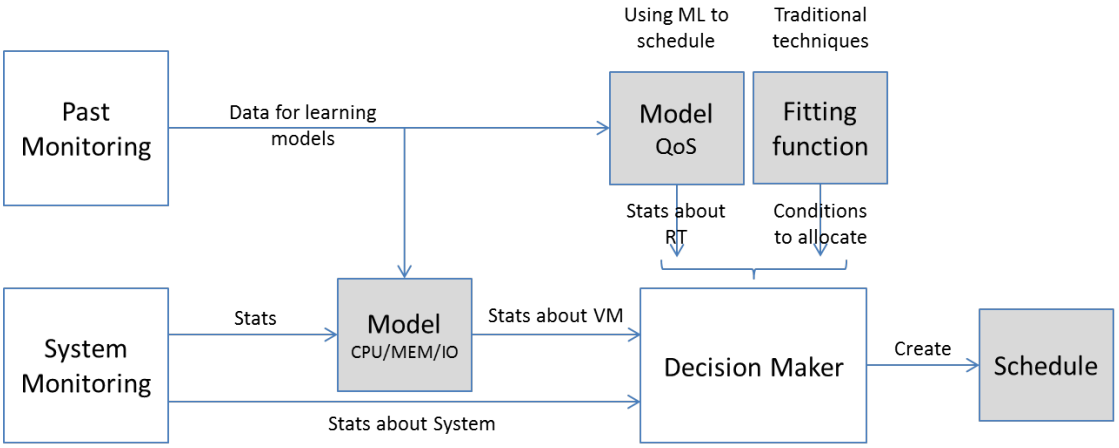


Figure 4.2: Information flow schema using models

the information must be shared but each different part of the system (cluster/datacenter/set of DCs) can partially manage itself.

Also, machine learning requires examples from the system to feed the learning algorithms and create the models. For this, previously of applying any model we run the system using the default decision makers and models, setting resources and services in several and different states (stressed, non-stressed, idle, isolated, ...), to collect as many different examples from the system, obtaining a representative sample of system behavior.

Chapter 5

Applying Predictions to Decision Making

In order to obtain an energy-efficient data center, in this chapter we present a framework that introduces an *intelligent* consolidation methodology using different techniques such as turning on/off machines and power-aware consolidation algorithms, applying machine learning techniques to deal with uncertain information while maximizing performance. For the machine learning approach, we use models learned from previous system behaviors in order to predict power consumption levels, CPU loads, and SLA timings, and improve scheduling decisions. The framework presented here is vertical, because it considers from watt consumption to workload features, and cross-disciplinary, as it uses a wide variety of techniques. We evaluate these techniques within the framework exposed in the previous chapter, covering the whole control cycle of a real scenario, using a simulation with representative heterogeneous workloads, and we measure the quality of the results according to a set of metrics focused toward our goals, besides traditional policies. The results obtained indicate that our approach is close to the optimal placement and behaves better when the level of uncertainty increases.

5.1 Introducing Energy-aware Prediction and Scheduling

Recalling the "green computing" idea explained in the Background Concepts, companies and IT infrastructures are focusing now on the need to improve energy efficiency. A large variety of power-saving methods has been presented in the last years, e.g. consolidation techniques, and "intelligent" turning on/off servers and resources.

Nevertheless, most previous proposals focus only on particular scenarios, cover only single strategies, or deal with synthetic data for some phases of the control cycle. For this reason, we propose a framework based on classical workload consolidation for reducing the power consumption of a data center executing a real dynamic workload, which covers the whole control cycle: from the acquisition of real power measures to the scheduling of the resources in the most power-efficient way according to these measures. Our approach applies some scheduling policies that reduce the number of unused machines according to the workload needs in each moment, and decide task placing and reallocation in order to compact jobs in the lowest number of machines without degrading their service level agreements. A new challenge that needs to be addressed here is providing a well-defined metric to evaluate the effectiveness of different adaptive solutions. For this, we define additional metrics in addition to power consumption to assess the quality of a given approach.

Furthermore, some scheduling information is sometimes not available or imprecise due to the user task specification or different unexpected events inherent to the system. Some decisions use information that can vary during the execution time or is heuristically obtained. When the system and the required application-level measures might be wrong or absent, we can use

predictive methods to 'model' this missing information. In order to provide a better and more *intelligent* consolidation, we propose a machine learning-based method for obtaining models of application and machine behaviors that let us predict service levels before applying changes on the system, maintaining QoS while reducing energy consumption. This work is a proof-of-concept on applying new machine learning techniques in situations where information can be missing or unclear, so for now we focus on CPU dependent workloads and CPU usage timing constraints, expecting to extend the approach towards full resource representative environments and workloads.

Related Work

Lost of works have been done in the last decade focusing on applying energy optimization techniques in multiprocessor environments (e.g. [97][33]), load balancing and performance optimization (e.g. [129]), or economical approaches (e.g. [44]). Usually all of these approaches use greedy resource allocation to distribute web workload or high performance tasks (like [47]).

We propose adding smarter scheduling policies, using machine learning techniques, to dynamically turn off idle machines and reduce the overall consumption. While some approaches like [92] or [125] use rigorous mathematical formulations to minimize wasted power or dynamically configure consolidation in virtualized server clusters, while turning on/off the spare ones. However, these works are too focused in web workloads and the decision algorithm is intended to run every 5 minutes, in contrast with our current approach that can handle heterogeneous workloads and adapt the system at every new job arrival, making better use of energy. The use of heavy mathematical calculus in the scheduling can lead to a too slow decision process for an online scheduler like the one we are looking for at this time.

Following the idea if virtualized jobs consolidation, [102] and [166] propose virtualization aware consolidation approaches. In [166] they use correlation techniques in order to predict usage, while we use machine learning to predict application power and performance, but at this moment they do not apply powering off techniques, just analyze the system. Also there have been also several proposals into QoS control (e.g. [15, 138, 141]), applying energy saving policies while meeting the problem of fulfilling SLAs [45]. Following this idea, we show how scheduling policies can take into account such problems.

Machine learning approaches have also been used to reduce power consumption in clusters. Tesauro, Kephart et al. [153, **?**] present a reinforcement learning approach to simultaneous online management of both performance and power consumption. These approaches look at learning what policies should be applied given a system status. Such policies save more than 10% on server power while keeping performance close to a desired target. Das et al. [53] present an approach using multi-agents in order to turning-off servers under low-load conditions, achieving 25% power savings without incurring SLA penalties on server farms. All these approaches use reinforcement learning in order to learn management policies from given data, while we are using, at this moment, induction learning to model the data for a given policy.

5.2 Energy-aware management

Our approach uses two different mechanisms in order to reduce the power consumption of a data center while respecting the different SLAs. One of the mechanisms that allows saving more power is turning off idle machines, which saves more than 200W in testbed machines. A complementary mechanism is trying to execute all the tasks but with the minimum amount of machines, *consolidation*, as seen in previous chapters. Therefore, scheduling takes a main role in order to achieve this power consumption reduction.

We want to turn off some idle machines in order to save power and we turn on them again if they are needed when a peak load occurs. For this purpose, our strategy is based on consolidating a set of tasks, distributed among a set of machines, into as few machines as possible without degrading excessively the execution of these jobs. Here, several scheduling policies could be applied in order to assign new jobs in the system to available machines and redistribute jobs being executed in order to make some machines idle and then turning them off [77]. Notice that

turning on machines again is not a free and instantaneous process and this overhead, which can take more than a minute, must be taken into account.

We consider several traditional scheduling policies, including; *Random* which assigns the tasks randomly (taking into account if the node fits there); *Round Robin* which assigns a task to each available node, which implies a maximization of the amount of resources to a task but also a sparse usage of the resources; *Backfilling* which tries to fill as much as possible the nodes, thus solving the former problem; *Dynamic Backfilling* which is able to move (i.e. migrate) tasks between nodes in order to provide a higher consolidation level. When tasks enter or exit the system, it checks if any tasks should be moved to other nodes according to different parameters such as the system occupation, current job performance, or expected user SLA satisfaction.

While *Dynamic Backfilling* performs well when having precise information (as shown in the evaluation), other policies are necessary when information is incomplete or imprecise. For this reason, a machine learning policy is introduced in order to predict features that will only be known in the future. This lets us anticipate the SLA degree and the power consumption before placing or moving jobs, and therefore choose a job configuration that is expected to be good.

5.2.1 Machine Learning approach

In this study, we use machine learning techniques in order to predict, from our set of machines and set of jobs, the resulting client satisfaction level of each job and power consumption before placing tasks in machines or moving tasks across machines. These predictions are then used by a move selection algorithm to choose destination machines with good resulting client satisfaction and opportunities for consolidation.

For this prediction process, we need to choose suitable predictor algorithms, computationally light but able to obtain good results once trained with data from various workloads. Also, we need to obtain a good training set (a set of data containing labeled instances from representative executions) and another test (or validation) set. If, after training, the predictors' guesses are close to the correct values on the test set, we expect that they will also be correct on future real workloads.

The machine learning aided policy implements a *Dynamic Backfilling* scheduler replacing the static decision maker, using the information provided directly by the user, and using as decision maker the results of the performance and power consumption estimators. This is, instead of fitting jobs in host machines directly from the user specifications, we estimate the impact the job will cause in the potential host machine, in performance parameters and power consumption.

In the line of *Dynamic Backfilling*, for each reschedule we attempt to empty low-used host machines fulfilling nearly fully-booked ones. Then, for each movement we estimate whether the job will interfere in the resource requirements of all other jobs in the machine, and the estimated new power consumption of this machine will compensate the possible performance degradation. This permits to obtain a more adaptive and robust system, where user or application specifications can be imprecise or change over time.

5.2.2 Relevant factors and basic assumptions

When a new job arrives, the system will try to allocate it to some host, and then perform a scheduling round in order to find a more efficient schedule. The candidate moves are of the form "move job j from its current host to host h", and the chosen one will be the one with maximum expected benefit. This benefit is the combination of two factors: the future performance of the jobs and power consumption in the resulting allocation, that we call R and C. Given a host, R_h and C_h cannot be known beforehand in general, so we will predict these values from our learned models obtaining the estimated $\hat{R}_h$ and $\hat{C}_h$.

The factor R_h indicates the *health status* of the jobs running on a machine h. This factor can be represented as a number between 0 and 100; a value close to 0 will indicate unacceptable performance, and a value close to 100 will indicate a good performance of the jobs in the machine. For this case of study, as a proof of concept we will assume that R_h depends only, for each job allocated to h, on the particular deadline constraint, indicating the SLA fulfillment.

Usually an SLA is an agreement on application resource consumption or performance guarantees (bandwidth, disk and CPU quotas, response time or throughput, time deadlines). In this paper we use a time deadline metric as SLA guarantee. The SLA fulfillment level follows a grid client satisfaction ratio, where it is fulfilled when the task completion time takes less than the deadline given by the user.

In next chapters we are including other relevant metrics into the SLA objectives, such as throughput constraints for service applications and time of response for interactive applications.

We can define a *finished job* j by a tuple

$$j = \langle UserT_j, SLAFactor_j, StartT_j, EndT_j \rangle$$

where $UserT_j$ is the user estimation of the time to complete the job, $SLAFactor_j$ is the factor over $UserT_j$ that the user is willing to accept, and $StartT_j$ and $EndT_j$ are the times in which the job was started and finished.

The performance factor R_h can be calculated in the way of

$$R_j = f(UserT_j, SLAFactor_j, StartT_j, EndT_j)$$

where function f, which is negotiated with the user, indicates the penalty for not satisfying the user's requirement, and we use it to define the fulfillment of job j, R_j (independently of the machines in which it has been executed). A very strict function f (f_{hard}) would indicate maximum loss when the SLA is not totally satisfied, while softer functions (f_{soft}) could go from 100 to 0 smoothly.

$$\begin{aligned} f_{hard} &= \begin{cases} 100 & \text{if } EndT_j - StartT_j \leq UserT_j \cdot SLAFactor_j \\ 0 & \text{otherwise} \end{cases} \\ f_{soft} &= max(100, \frac{UserT_j}{EndT_j - StartT_j} \cdot SLAFactor_j \cdot 100) \end{aligned}$$

At this stage of the work we use the softer version of f, expecting to use more elaborated functions when we dispose of more complex workloads with complex SLA requirements. For this work, the value of R_h given a machine h should be the aggregation of the values R_j for all allocated jobs on h. Supposing an initial hypothesis of fairness between the jobs on a machine, for this version of our work we take as aggregation function g the arithmetic mean of the R_j's $g(h) = \sum_j^{Jobs_h} R_j/(Num\ Jobs_h)$

The consumption factor C_h indicates the power consumption of machine h. It can be measured empirically for the training data sets, and possibly during the execution. Our experiments and common knowledge indicates that it depends mostly (but is *not* linearly proportional to) the percentage of CPU usage at h.

The global function that the system should optimize is a combination of the aggregated levels of SLA fulfillment and the total power consumption, that is of $R = g(R_1, \ldots, R_H)$ and $C = \sum_{h=1}^{H} C_h$ if we have H host machines. For the moment, we decided to choose moves that maximize R under the condition that they do not increase C; this maintains SLA accomplishment as a priority over consumption, which is the usual practice up to date. A summary of symbols is shown on Table 5.1.

5.2.3 Data sets and prediction algorithms

The values of R_h and C_h as described above are usually not known before performing the job allocation and finishing the running jobs in the machine. But we can run representative executions in order to obtain examples of $\langle jobs, machine, R_h, C_h \rangle$ configurations, and learn a model capable of predicting, from $\langle jobs, machine \rangle \Rightarrow \langle \hat{R}_h, \hat{C}_h \rangle$

To predict the power consumption $\hat{C}_h$, we found it useful to predict first the percentage of CPU usage at h. We used the Linear Regression algorithm, where the most relevant attributes resulted to be the CPU usage for each individual job in h and, with smaller weight, the number of jobs in the candidate host. This attribute choice was to be expected. Predicting power consumption is more complex that a simple linear regression, because it has a nonlinear relation with CPU usage, so we used the more sophisticated M5P algorithm. As explained in the

$UserT_j$	User estimation of time to complete job j
$SLAFactor_j$	Weighting SLA Factor in $[0, 1]$ for job j
$StartT_j$	Start time for job j
$EndT_j$	Ending time for job j
R_j	Performance obtained for job j
$f_{hard}(R_j)$	SLA fulfillment for job j, hard threshold
$f_{soft}(R_j)$	SLA fulfillment for job j, using decayment function
$R_h = g(h)$	Aggregation function for all R_j in host h
C_h	Power consumption for host h
$ExecT_j$	Execution time for job j ($EndT_j - StartT_j$)
$ExpectedT_j$	Estimated total execution time for job j

Table 5.1: Summary of symbols and functions

Background chapter, M5P builds a decision tree splits on attributes at inner nodes and applies different linear regression at the leaves. It therefore computes a piecewise linear function of the attributes, which is enough to approximate the nonlinear dependence of power consumption on (mostly) CPU usage and number of jobs.

The real problem is predicting the deadline fulfillment of a given job, R_j because the most important value $EndT_j - StartT_j$ will not be known until the job ends, and also the user estimate $UserT_j$ may be inaccurate. Using the learned model we predict $\hat{R}_j$ using as known information: the amount $UsageCPU_j$ of CPU used by the jobs on h (included the new job), time spent so far $Now - StartT_j$ and the characteristics of the machine where it is executing $AvailableCPU_h$. For this prediction we have used another linear regression function, where the most relevant values are the timing values for j and other jobs in the same machine. In this preliminary work, assuming that all the machines have identical capacity, some attributes about machine characteristics need not to be included into the learned model, but as see during the experiments of the Linear Regression, a coefficient exists directly related to the capacity of the machine.

The algorithm pseudo-code is shown in Algorithm 3. Basically it performs a dynamic back-filling strategy, using the learned model to decide whether the new allocation of the job will bring an energetic improvement given an estimated performance cost. At each scheduling round, the underused hosts are selected to be emptied, and their jobs are *virtually* allocated in nearly-full hosts. Using the model, we can estimate if this is a suitable allocation. If it is, and we consider the host can be emptied, we proceed to perform the jobs reallocation. Note that it is a greedy algorithm that is not guaranteed to find the theoretically best possible list of movements.

5.3 Simulation and Metrics

Simulation consists of the evaluation of a set of nodes which are stressed by a given workload. The system performance is evaluated according to a set of different metrics that take into account consolidation and power consumption.

The simulation is able to compare different scheduling policies. They are implemented in a modular way and can be plugged on top of other architectures, such as EMOTIVE [68] or XtreemOS [177, 49] (inside its component Application Execution Management, AEM [50]), where a set of schedulers can be introduced and selected by the administrator. XtreemOS follows a scheme of local scheduling rather than a global one, but its direct interaction with the kernel and its dynamic resource discovery via DHT can provide a way to simplify the switch on-off techniques.

5.3.1 Simulation and power models

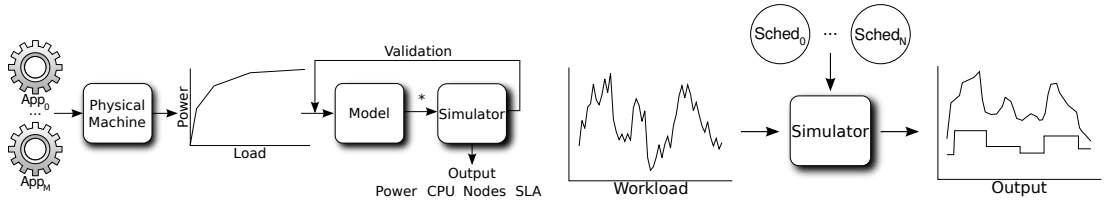
In this section we present the framework for evaluating the power efficiency of a data server which executes an heterogeneous workload. This framework tackles the whole problem from the power consumption measurement of a single machine to the execution of different applications on a data center which allows evaluating different approaches such as dynamic turn on/off or consolidation.

Algorithm 3 Machine learning move selection algorithm

```
Poll hosts for information about their jobs and status;
OH := select "Emptiable Machines" [jobs < 4];
For each Machine (oh) in OH do:
    For each Job (j) in oh do:
        CH := select "Fillable Machines" [enough CPU and mem];
        For each Machine (ch) in CH do:
            -- predict effect of moving j from oh to ch;
            predict R(oh) and R(ch) after movement;
            predict C(oh) and C(ch) after movement;
            compute global R and C after movement;
        End For
        Get ch leading to highest R among those that decrease C;
        add movement (j,oh,ch) to List_of_movements;
    End For
    If (all jobs in oh can be reallocated) then:
        proceed with the List_of_movements;
    End If
End For
```

It is based on a simulator in order to evaluate the performance of a whole data center focusing on power consumption. It allows obtaining different metrics of the modeled cluster while applying different workloads in order to optimize different policies.

Figure 5.1a shows the development cycle of the simulator used in our framework. Firstly, different applications with different typologies and profiles are executed (on a real, not simulated, machine) and their resource usage and power consumption monitored. Power usage is recorded using an external device which monitors the whole machine energy consumption. From these recordings, a model of the machine is built, which is then used to simulate a data center with many (identical) machines. Validations are applied to refine the model and the simulator. Finally, the simulator is executed to provide the experimental data described here.



(a) Simulator workflow (b) Simulator

Figure 5.1: Simulator Diagrams

This simulator is able to add and remove nodes dynamically including the boot time and load times. It allows using different scheduling techniques which can take advantage of different capabilities such as migration of running tasks between nodes. Finally, the simulator is able to read a workload and apply the different scheduling policies to output the results of executing the workload, including power consumption and other resource usage statistics, as shown in Figure 5.1b.

Energy consumption measurement

We measured the real power consumption using different workloads in a 4-CPU computer (the Intel Xeons) whose kernel includes some energy efficiency policies. The power consumption of the machine was gathered using digital methods via an ammeter. In the past, analogical methods via oscilloscope where used, as seen in [119], but similar results are obtained with the ammeter method (however, instantaneous wattage is lost; we can only measure stable workloads). The

resolution of the measurements is below 1 Watt. Figure 5.2 shows the system behavior; we can see that wattage increases with the workload (in a non constant slope), but that it is noticeable even in an idle machine, which is the main reason why we can gain by consolidation. This graph was included in the simulator, as part of the model. It is important that idle wattage level should be decreased in the industry as it is one of the most used states and it is not energy efficient, as seen in [22].

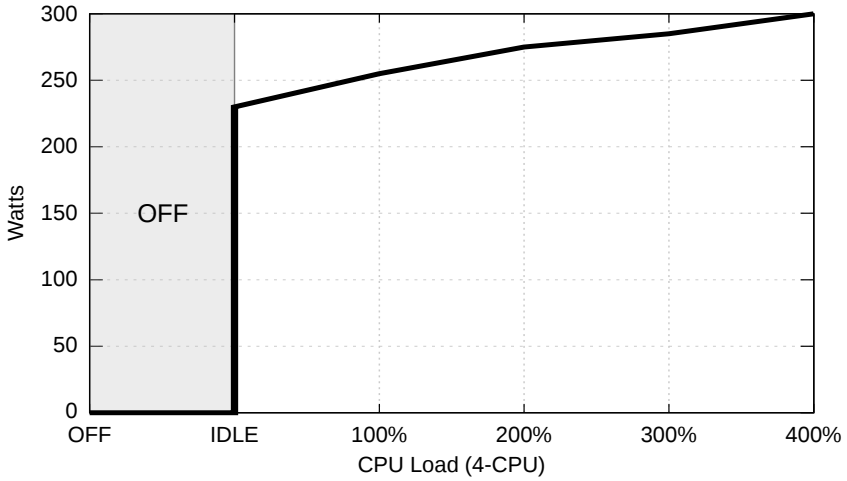


Figure 5.2: Power behavior of the target PC

Testbed Simulator and Model validation

Our testbed simulator is the Energy-Efficient Simulator EEFSIM, previously introduced in chapter 4. The EEFSIM contains the models tailored by the human experts and workload models from HPC jobs and transactional (web-service) jobs.

We have validated the model using special schedulers whose outcome we can predict separately and testing that the simulation obtains the expected relative values (such as SLA = 1.0 or similar). Moreover we tested that simulating one machine produces similar power usage values as in the real case. Future work may include fine-tuning and a more detailed validation, as some smaller overheads and other issues that can show in the real world should be modeled and introduced in the simulator.

5.3.2 Metrics

One of the key proposals of this paper is the ability to compare different techniques for efficient power usage. Currently, there is no standard approach for measuring the power efficiency or the consolidation of a datacenter. For example, [140] proposes a benchmark for measuring power efficiency on a set of different scenarios such as mobile devices or servers. However, it does not address consolidation.

In this paper, we introduce some metrics to compare adaptive solutions. To this end, the energy consumption must be evaluated precisely. This part is mandatory to be able to compare different approaches. Nevertheless, it is not enough since a given policy can decrease the efficiency energy but it can make some tasks violate their SLAs. In addition, consolidation factors are also important for measuring the scheduling policy quality as understanding what a scheduler is doing is not easy just evaluating energy or SLA fulfillment. For this purpose, we add some other metrics that would help to comprehend and measure different relevant aspects.

Working nodes The number of nodes that are executing some task. Hence, in order to allow shutting down more nodes, less working nodes are better.

Running nodes The number of nodes that are turned on. Having a lower number of these machines is one of the key issues for saving energy in order to reduce the idle machine consumption.

CPU usage The amount of CPU time that has been used.

Power consumption Total energy consumed by the nodes.

SLA fulfillment level The client satisfaction based on the task SLAs. We evaluate a service by its availability ratio, which is 100 if it is always available, and 0 if it never is. On the other hand, we will use the typical grid client satisfaction ratio, which is 100 if execution time is less than expected time and 0 if completing the task takes longer than twice the expected time. This is defined by this equation:

$$SLAFulfillment = \begin{cases} 100 & \text{if } ExecT < ExpectedT \\ 100 \cdot max\{1 - \frac{ExecT - ExpectedT}{ExpectedT}, 0\} & \text{if } ExecT \geq ExpectedT \end{cases}$$

5.4 Evaluation of the Energy-aware Schedule

In this section, we present the experimentation regarding our strategy and the different used techniques, and also the simulation and power consumption models used for evaluating them.

5.4.1 Experimental environment

The experiments will consist of the simulation of a whole datacenter with 400 nodes that will execute different workloads and will evaluate its behavior according to different metrics including power consumption.

The presented approached intends to take benefit of the variation and the heterogeneity in current datacenters. For this reason, the evaluation includes two different workloads: Grid and service oriented. The former is a Grid workload obtained from Grid5000 [155] on the week that starts on Monday first of October of 2007. The training of the ML model has been performed using the workload corresponding to the week of third of September. For evaluating the *SLA satisfaction*, SLAs have been added to the Grid jobs, specifying tolerance factors in execution times in the range 1.1. . . 2.0.

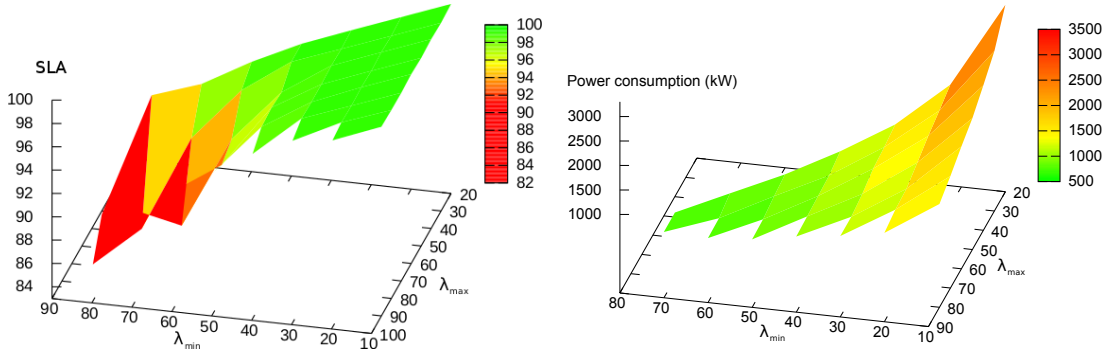
The latter workload results from the aggregation of different services based on the load of *Ask.com* [17]. These services correspond to three different profiles. One that represents a single day execution from 0:00 to 23:59 with a low usage during the night and a classical increase at the start of day. The second one follows the same behavior but it has a bigger load in the afternoon. The third uses a whole week in order to represent the weekend user decrease.

Finally, the evaluation also includes a mix of the already presented workloads in order to simulate a heterogeneous datacenter and test the functionality of the approaches with a realistic approach for current datacenters.

5.4.2 Power vs. SLA fulfillment trade-off

In our approach, one of the key decisions is determining when a node should be turned off in order to save power consumption or when to turn on it again in order to be used to fulfill the tasks SLAs. This decision is driven by means of two thresholds: the minimum *Working nodes* threshold λ_{min}, which determines when the provider can start to turn off nodes, and the maximum *Working nodes* threshold λ_{max}, which determines when the provider must turn on new nodes. Finally, in order to set a minimum working set, the minimum amount of machines $\min_{exec}$ is also specified.

The effect of these two thresholds has been tested by executing the Grid workload on top of the simulated datacenter following the *Dynamic Backfilling* policy, which is the one which makes a more aggressive consolidation without taking into account the task SLA. This allows evaluating the influence of the turning on/off thresholds by showing the SLA and the power consumption respectively.



(a) SLA satisfaction using different turn on/off thresholds

(b) Power consumption using different turn on/off thresholds

Figure 5.3: SLA and Power using different turn on/off thresholds [Source: eEnergy'10 [30]]

Figure 5.3b shows that waiting the nodes to reach a high utilization before adding new nodes (high λ_{max}) makes the power consumption smaller. In the same manner, the earlier the system shutdowns a machine (high λ_{min}), the smaller the power consumption is. It demonstrates how turning on and off machines in a dynamic way can be used to dramatically increase the energy efficiency of a consolidated datacenter.

On the other hand, SLA fulfillment decreases, as shown in Figure 5.3a, when the turn on/off mechanism is more aggressive and it shuts down more machines (in order to increase energy efficiency). Therefore, this is a trade-off between the fulfillment of the SLAs and the reduction of the power consumption, whose resolution will eventually depend on the service provider interests.

Fortunately, average threshold values give a balanced trade-off between energy and SLA. According to this, in the evaluation we will use $\lambda_{min} = 30\%$ and $\lambda_{max} = 60\%$ in order to ensure almost complete fulfillment of the SLAs while getting substantial power consumption. A next step would be to dynamically adjust these thresholds, and it remains for future work for ad-hoc modeling techniques.

5.4.3 Validation of ML models

In this section, we evaluate the accuracy of the machine learning models derived during the training process, to assess their reliability to predict future situations. Furthermore, we evaluate the performance of our overall method for scheduling and consolidating.

The role of the machine learning methods is to provide predictions of values that are unknown at the time in which they are needed. In our setting, they provide some of the inputs to the *Backfilling* and *Dynamic Backfilling* algorithms that they need to perform their scheduling, namely, anticipated power variation and SLA fulfillment resulting from a possible move. Luckily, at the validation stage, the information about actual power variation and SLA fulfillments can be read from the available datasets. We can thus evaluate predictor accuracy on a test/validation subset, disjoint from the training set.

The linear regression model to predict SLA fulfillment ratio fits with the real measurements with average accuracy close to 0.985. This very high value is explained in part by our current choice of priorities. Since we prioritize SLA fulfillment over consumption, the algorithm's choices are conservative or cautious with respect to SLA's, which are therefore almost always fully satisfied, and therefore easy to predict. In fact, we did not find situations where the predicted SLA value is 1 but the actual SLA is lower: the 0.015 fraction of prediction errors are on the side of SLA's that are predicted to fail but finally succeed. This will generally implied that our algorithms do well on the SLA side at the expense of somewhat higher than necessary power consumption.

The model for predicting CPU usage, basically using the CPU usage of all jobs allocated to it, is accurate up to 0.997, almost perfectly. CPU usage prediction is in turn used to predict the power consumption of a machine after adding a job, and we obtained a high accuracy of

	Working nodes (avg)	Running nodes (avg)	CPU usage (hours)	Power (kW)	SLA (%)
Grid workload					
Round Robin	16.11	41.37	5954.91	1696.66	85.99
Random	16.51	40.76	6017.85	1671.16	88.38
Backfilling	10.18	27.10	6022.34	1141.65	100.00
Dynamic Backfilling	9.91	26.46	6104.33	1118.86	100.00
Machine Learning DB	15.04	37.92	6022.27	1574.78	99.69
Service workload					
Round Robin	290.99	400.00	78419.97	19761.54	100.00
Random	218.46	400.00	75336.88	19784.38	100.00
Backfilling	108.79	352.88	59792.09	16257.26	100.00
Dynamic Backfilling	108.79	352.88	59748.10	16229.22	100.00
Machine Learning DB	99.61	270.50	61379.38	13673.71	100.00
Heterogeneous workload					
Round Robin	260.66	400.00	84432.96	19713.72	94.20
Random	224.08	400.00	82137.27	19763.63	88.53
Backfilling	110.85	330.19	65894.46	16304.38	99.50
Dynamic Backfilling	111.03	329.07	66020.58	16214.49	99.59
Machine Learning DB	124.20	307.89	68554.01	15110.33	98.63

Table 5.2: Scheduling results

0.98 between the model and the workload data, so the model is able to predict consumption for low loads, average loads, and high loads. Having so validated the models, we can use them to provide inputs to the ML-based scheduler. Next subsection shows the results of this and other schedulers considered.

5.4.4 Scheduling policies

This experiment evaluates the behavior and performance of the different scheduling policies using three different workloads, namely a Grid workload, a Service workload, and a Heterogeneous workload. It uses the turn on/off thresholds $\lambda_{min} = 30\%$ and $\lambda_{max} = 60\%$ derived in Section 5.4.2.

We have evaluated five scheduling algorithms: *Random* and *Round-Robin* (see Algorithm 4) do not use any user-provided information about the jobs and do not consolidate. For *Backfilling* and *DynamicBackfilling* (see Algorithm 5), the user provides for each job a figure indicating which % of a CPU capacity should suffice to satisfy the task SLA's. The algorithms trust this figure as totally reliable, and therefore will make decisions that may fit very tightly the SLA's and therefore save power. Our algorithm, *Machine Learning*, has the drawback with respect to these algorithms that it does not use any user-provided information. Therefore, a priori we should expect it to perform worse in general, as it has to pay a price for this lack of information, but the closer in performance it is to these two algorithms with privileged information, the more successful we can consider our approach. Somewhat surprisingly, we will see that it does sometimes better than the algorithms having additional information. The results are presented in Table 5.2, according to the metrics proposed in Section 5.3.

The results obtained using the Grid workload show that non-consolidating policies such as *Random* and *Round-Robin* give a poor energy efficiency while violating some SLAs: these policies give the worst results on both criteria. *Backfilling* and *Dynamic Backfilling* fulfill all SLA's with substantially lower cost. *Machine Learning* performs almost perfectly w.r.t. SLA's (as we have seen that predictions for SLA fulfillment are very accurate), but with respect to power is closer to *Random* than to the backfilling algorithms. The reason is that the user-provided figures for the tasks are very close to the real ones (and the load quite steady), so the backfilling algorithms will

Algorithm 4 λ-Round Robin algorithm

```
for each vm i:
    get_data(i);
    res_quota[i] <- get_required_resources(i);
end for
for each host j:
    res_avail[j] <- get_total_resources(j);
end for
numHosts <- calculateNumHosts(res_quota[],lambda);
c_host <- 1;
for each vm v:
    visited <- 1;
    while (not fit(res_quota[v],res_avail[c_host]) and visited <= numHosts) :
        c_host <- (c_host + 1) % numHosts;
        visited <- visited + 1;
    done
    if (visited <= numHosts) :
        assign_vm_to_host(c_host,v);
        update_resources(res_avail[c_host],v);
    else :
        assign_vm_to_host(null_host,v);
    end if
end for
```

Algorithm 5 (Dynamic)Backfilling algorithm

```
for each vm i:
    get_data(i);
    res_quota[i] <- get_required_resources(i);
end for
for each host j:
    get_data(j);
    res_avail[j] <- get_available_resources(j);
end for
order[] <- order_by_empty(hosts,res_avail[]);
for each host h in order[]:
    for each vm v in host h:
        k <- numHosts;
        l <- index_of(h,order[]);
        stay <- true;
        while (k > l and stay) :
            c_host <- order[k];
            if (fit(res_quota[v],res_avail[c_host]))
                move_vm_to_host(c_host,v);
                update_resources(res_avail[c_host],v);
                stay <- false;
            end if
            k--;
        done
    end for
end for
```

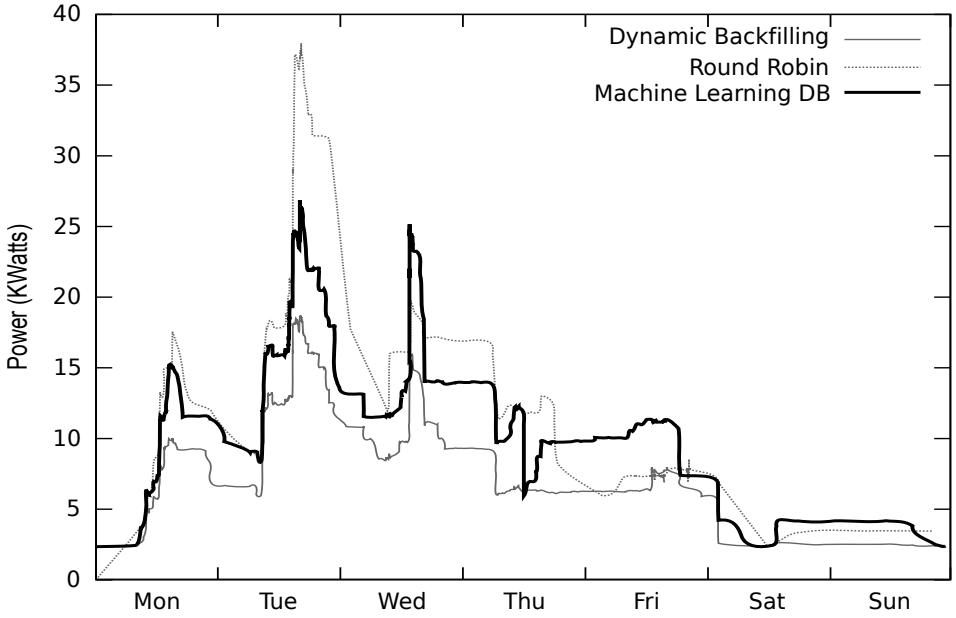


Figure 5.4: Power consumption of different schedulers with a Grid workload

take many decisions that will not violate any SLA but that look too risky to *Machine Learning*, that pays a high price in consumption for its caution.

Note that this workload makes a very variable use of the power consumption over time as it is graphically shown in Figure 5.4. This is due to the fact it makes the system creating and destroying many VMs, which implies a high variability in the number of running nodes and power consumption during time. The figure shows the power consumption pattern of the different schedulers and enforces the table results.

On the Service workload, the *Machine Learning* scheduler is the clear winner with respect to energy consumption. Note first that on this workload all the schedulers executed all the tasks, so all SLA's are fulfilled. The workload has a very variable CPU usage. This means that the user-provided estimation about the CPU to be used for the given jobs will be a large overestimation for large periods (while it was very tight on the Grid workload), and power will be unnecessarily wasted. Here is where the *Machine Learning* scheduler takes advantage because of the capability of computing somewhat conservative but adaptive estimates of the degree of SLA fulfillment, and adapt its power consumption accordingly. Thus, it is able to work better when the features of the input load are not known or the user-provided estimates are misleading, which is very often the case.

Finally, the results obtained using the Heterogeneous workload are, as expected, a mix of the two previous workloads. In this case, the overall SLA fulfillment by our algorithm is worse by about 1%, but its overall power consumption is better by about 10%. Figure 5.5 shows the evolution over time, and one can see that *machine learning* does worse with respect to SLA when the CPU utilization is higher (i.e., when the other algorithms can exploit the user-provided information they have), but much better than Random and Round Robin, which behaves very similar to Random.

5.5 Conclusions for energy-aware scheduling

In this chapter we have introduced and presented a framework that provides a vertical and intelligent consolidation methodology to deal with uncertain information keeping in mind performance and power consumption at the same time. This framework covers the whole control cycle of a real scenario with a holistic approach that requires a collaboration among researchers from different disciplines. The results obtained in this paper indicate that significant improvements can be achieved using machine learning models in order to predict application SLA timings and

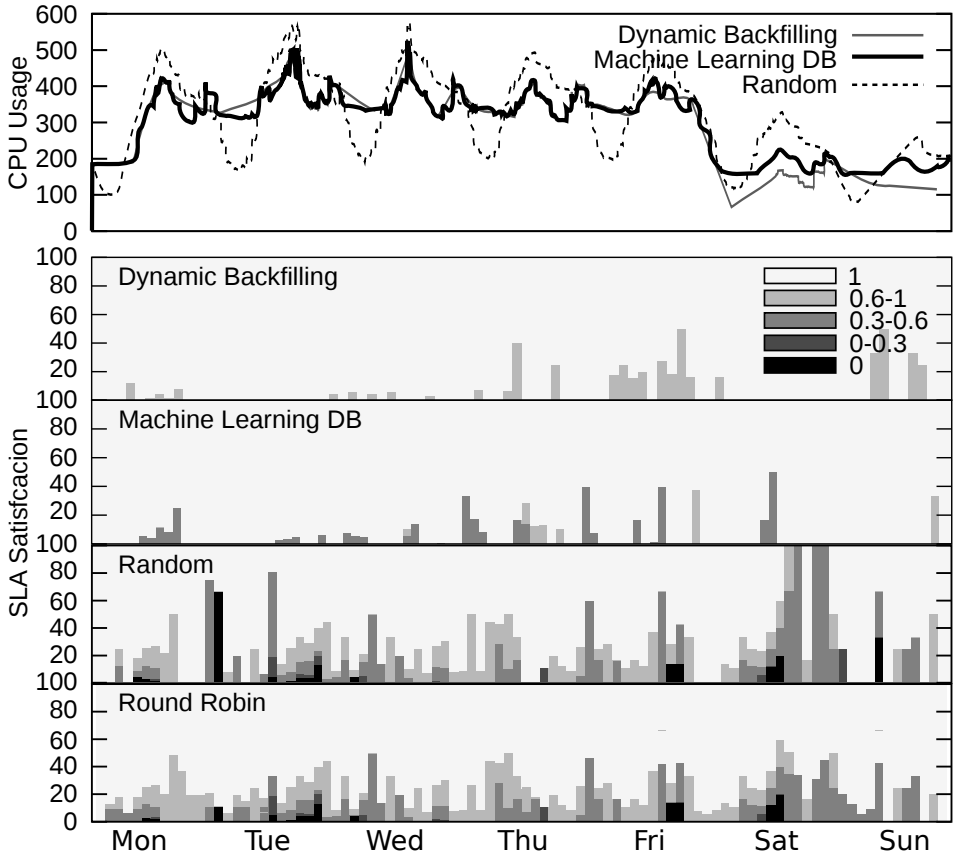


Figure 5.5: CPU usage and SLA fulfillment with heterogeneous workload; Most significant policies: Dynamic Backfilling, Machine Learning and Random

decide the movements and operations to be done within scheduling functions.

The experiments, performed using real workloads, exemplify that these techniques can offer substantial improvements in energy and performance efficiency in these scenarios. Using the Grid workload the experiments demonstrate how non-consolidation aware policies give a poor energy efficiency. Backfilling gets a good performance and its dynamic extension demonstrates power efficiency in order to reduce power consumption, but only if reliable a priori information on the tasks is available, and if the task features are steady over time. The machine learning method is close enough to these models that use external information w.r.t. SLA fulfillment (performance), and much better with respect to power consumption when the information provided by the user is not uniformly accurate. On mixed, heterogeneous workloads, it obtains noticeable reductions in power consumption at the expense of only a slight decrease in performance.

On this work, as a proof of concept, we used a greedy algorithm for scheduling (Dynamic backfilling) and we departed from basic attributes (CPU Usage, Timing SLAs), as a first approximation of a decision making methodology. Therefore, as being proved the viability of adding ML techniques to improve power management, we are now including the concept of *resource* aggregation of not only CPU but also memory and IO, and expanding the concept of SLA.

The work presented in this chapter has been published in the "international ACM eEnergy 2010 Conference" [30] (2010) and as a chapter in the book "Energy-Efficient Distributed Computing Systems" [31] (2012).

Chapter 6

Mathematical Modeling of DataCenters

In this chapter we expose how to represent a datacenter based scheduling problem, taking the advantages of the virtualization and consolidation techniques, as a mixed integer linear problem meeting energy saving, economic profit, and customer satisfaction. First we expose step by step the fundamentals for representing the DC, and how factors and variables relate among them. Then we study the trade-offs between those factors. Secondly we test the model in a framework for autonomic scheduling of jobs and web-services, optimizing the profit, taking into account revenue for task execution minus SLA violation penalties, minus energy costs. We use an exact solver based on mixed linear programming as a proof of concept but, since it is an NP-complete problem, we show that approximate solvers provide valid alternatives for finding approximately optimal schedules. Also we introduce machine learning techniques to complement unknown or unavailable data (field to be expanded in the next chapter). In particular, here we predict a priori CPU consumption by different web-services under real workloads, and estimate the web-service SLA (such as response time) given workload features, host characteristics, and contention among other web-services in the same host.

6.1 Introducing the Modeling Approach for DataCenters

In datacenter services, the goal of the manager is to maximize its revenue, executing as many hosted services as possible but constrained by the infrastructure. Overloading the infrastructure will lead to a QoS degradation and thus unfulfilled SLAs. Also high job throughput and user-satisfaction can be obtained by deploying large amount of resources, incurring in high energy costs. The aim of this work is to propose a method for adaptively remaining at the sweet spot where enough resources (hosts - hence electrical power) are deployed to achieve almost maximal throughput and user-satisfaction, while minimizing its costs.

This chapter presents a datacenter modeling focused on the three main goals: economic revenue, power saving and quality of service. This model represents a data-center as a set of elements (machines, processors and web-services in virtual machines) to be scheduled in an optimal way, setting for each element a set of constraints to be accomplished, representing the capabilities of each one and its requirements. The proposed methodology models a grid based data-center as a set of resources and as a set of virtualized web-services, each one with its energetic or resource requirements, involving a consumption cost and a rewarded execution. The problem to be solved at each scheduling round is decide what resources are assigned to each VM, while maximizing a function result of the benefit from VMs revenues, the power costs and how much loss of QoS is tolerated in order to save energy. For this reason, this approach represents the model as a mixed integer linear program, solving each scheduling using well known linear programming techniques and integer optimizers.

It is well known that solving a MILP can become a hard computational problem, also the integer optimization is often fitted within a time function, but having good constraints and a well defined problem can help to find, in a short time, solutions very close to the lower bounds of the problem. This lets the middleware specialists to understand better the system and be able to create good heuristics close to the optimal/sub-optimal solutions found with MILP solvers. In fact, this work will be really useful in order to obtain data and information towards analyze the behavior of grid datacenters and learn new management rules.

Related Work

As shown in previous chapters, here we use virtualization and consolidation as a basic strategy. Our work is based on the advantages of these technologies, in order to build the mathematical model, and in order to build upon it the machine learning mechanisms. Also we take advantage on the ability to determine when to turn on and off physical machines. The ideas for this model are based in the work done by J.S.Chase et al., presenting the MUSE framework [43], a framework for modeling and autonomically control hosting centers and its policies. They present an economical managing approach for scheduling jobs to resources, where hosts and jobs bid for resources, and elasticity is allowed taking into account penalties for not fully attended resources. The model presented here follows the same outline and basic concepts they used, with the addition of our own policies and introducing a ML contribution.

Mathematical modeling is a good way to depict an optimization problem, as it lets to detail all elements involved and the problem constraints. Petrucci et al. [125] developed a MILP formulation to dynamically configure the consolidation of services in a virtualized server cluster, with the problem of heavy calculus required per schedule. Also works like [34] present the energy-aware service allocation problem as a mathematical program, driven by an effectiveness energy utilization factor instead of economic SLA constraints as we do here. Also they indicated the hardness of solving a MILP in practice and compared the exhaustive solving with greedy algorithms. Here we follow the same methodology, by depicting the model, testing exhaustive solving solutions, and then compare with heuristic or approximate algorithms as an alternative.

6.2 A MILP Representation for DataCenters

A grid based data-center can be modeled as a set of resources, each one with a consumption cost, and a set of web-services in VMs to be executed with a set of resource requirements and a set of benefits and execution penalties. The problem to solve at each scheduling round is to decide what resources are assigned to each VM, depending always in its requirements and conditions established by the agreement between the provider and the client submitting the job (the service level agreement). The best solution will be that one that maximizes or minimizes a target function, usually a function describing the benefit of the solution.

In the proposed situation, the three elements to maximize or minimize are the power consumption, the economic benefit and the client satisfaction, so a good solution is that one assigning resources to VMs (or VMs to resources) saving the most electrical power while granting a good quality of service and having a positive benefit value by serving that clients. So the problem can be represented by a function to maximize the optimal balance between the three elements, and a set of conditions and rules in order to set VMs and resources without overloading the resources and granting viable and real solutions. At this time we consider that a VM can not be split between two or more hosts. When scheduling a VM into a host, the VM remains completely in that host, turning the problem “integral”.

6.2.1 Scheduling Approach

The solution for the problem defined here is a scheduling integer binary matrix $Hosts \times VMs$, where each position $[h, j]$ indicates whether the job j is or not in host h. A valid solution must accomplish the condition that a job must be run entirely in a unique host, so it can not be split in different hosts at this time.

Each job needs determined resources to run properly at each moment, like CPU quota, memory space and I/O access. At this time, as an example and case of study we focus on CPU resources, understanding that memory and I/O can also be represented following the same model and expanding it. With this purpose, the system must be able to observe the CPU consumption of each task, and also the available CPU for each host. A solution looking for assuring the VMs requirements must allocate VMs in hosts in a way that each VM has its CPU quota, and the host is able to give that required CPU. So the sum of the VMs CPU demands in a given host must not surpass the CPU capabilities. These conditions are the basic problem constraints. So a solution consists in a schedule where VMs are in one host only, the load of a host does not surpass the capabilities of it:

Variables:	$schedule[Hosts, VMs]$, as Integer Binary ; representing the Schedule
Parameters:	$cpus(h)$, as CPUs existing in host h
	$cons(j)$, as the CPU consumption of VM j
Constraints:	$Unique\{j \in VMs\} := \sum_{h \in Hosts} schedule[h, j] = 1$
	$Capacity\{h \in Hosts\} := \sum_{j \in VMs} schedule[h, j] \cdot cons(j) \leq cpus(h)$

Note: Consider that a VM CPU consumption can be a positive real value instead of a positive integer value. When a VM enters in CPU keeps the CPU for itself during its time quantum, but as VMs tend to block themselves waiting for IO or resting in idle/sleeping status, given a determined amount of time the expectation of CPU consumption can differ from rounded values.

6.2.2 Minimizing the power cost

Introducing the first factor, one of the main goals is to schedule properly datacenters while minimizing the power consumption. At this first stage, the search problem focuses on reducing the number of CPUs used to run the datacenter VMs by consolidating them. The idea of consolidation consists in running the commended VMs using the least resources as possible, and in our case it means running as more VMs as possible in as less hosts as possible. As shown in our previous work [70], given a host the power curve grows in a logarithmically alike way, understanding that two machines with many processors running only one each of them consumes more power that only one of those machines running two processors, keeping the second one in a low-power state or just shut down.

In order to model the CPU power consumption of a given host, the CPUs of it can be considered as on-Line or off-Line depending on the required load. New technologies and architectures not only allow to maintain multi-processor in idle states but also shut down processors and components on demand. I.e. a load of 2.50 CPU should make a host to run 3 CPUs, leaving a 0.5 CPU in idle state (consuming power) and letting a VM with a demand of 0.5 CPU to enter and take profit of this CPU waste. And when CPU load goes beyond 2.0 the host is allowed to shut down a CPU reducing the consumption waste.

This brings new conditions to be accomplished: the solution will consider that the number of CPUs given a host is a natural value, so the schedule will also consider separately the processors (CPUs) of a given host. Also, the number of active CPUs in a host will not surpass the maximum number of CPUs available on that host (this condition looks silly at first time, but must be considered when having a model with different shaped hosts in the data-center).

In order to keep the representation of the problem as much linear and clear as possible, the value to minimize is the sum of all the power consumed by the scheduled solution and its active CPUs. This is, for each host, look at how much CPUs are running, and check the power consumption according to the host characteristic power curve. I.e. the function to minimize for the host represented, consisting in a 4 CPUs Xeon machine following its power curve, would be $pr_1 \cdot 267.8 + pr_2 \cdot 17.7 + pr_3 \cdot 17.0 + pr_4 \cdot 15.4$ (In this case coefficients are kWatts/hour). Then, the sum of the power consumption of all the data-center is the sum of all power measures. This representation of the problem requires a last condition in order to make it reliable, as CPUs will be started up and shut down in order, so the first CPU to be active in the representation model will be pr_1, then pr_2 and successively.

The resulting model minimizes the consumed power taking into account the consolidation goals, like filling a half-empty host is better that starting a stopped one:

Variables:	$schedule[Hosts, VMs]$, as Integer Binary ; representing the Schedule
	$pr_i[Hosts]$, as Integer Binary ; representing the use for each host of its i-essim CPU
Parameters:	pwr_i, as the power consumed by an i-essim CPU
	$cpus(h)$, as CPUs existing in host h
	$cons(j)$, as the CPU consumption of VM j
Minimize:	$\sum_{h \in Hosts, i \in cpus(h)} pr_i[h] \cdot pwr_i$
Constraints:	$Procesors\{h \in Hosts, i \in cpus(h)\} := pr_i[h] \geq pr_{i+1}[h]$
	$Unique\{j \in VMs\} := \sum_{h \in Hosts} schedule[h, j] = 1$
	$MaxCPU\{h \in hosts\} := \sum_{i \in cpus(h)} pr_i[h] \leq cpus(h)$
	$Capacity\{h \in Hosts\} := \sum_{j \in VMs} schedule[h, j] \cdot cons(j) \leq \sum_{i \in cpus(h)} pr_i[h]$

6.2.3 Maximizing the profit

Once having modeled system from a power consumption point of view, the economic factor can be introduced. Successfully accomplished VMs are rewarded with revenue, so clients pay the data-center provider for running their applications on it. Given this obvious fact, VMs can be translated to money according the data-center pricing or service level agreements. Power consumption can be represented as kWatt/hour, tons of CO2, or also money per kWatt/hour, evaluating the power consumption by the cost of buying the required electricity. In this manner, revenues and power costs can be included in the same equation: $Benefit = Revenue - Costs$.

When VMs and power have a fixed value, and VM revenue is above power cost, benefit will always imply running the most applications as possible while consolidating, so power waste is minimized. Unfortunately, consolidation strategies have a great handicap: migration costs. Changing a VM from a host to another implies that during this process the VM is stopped or replicated, and this can make the SLA fail due to broken deadlines or interruptions of service, and also extra CPU load while moving the VM. For this reason, migration is penalized with an economical cost referring to client-provider SLAs or to time and resource wasting [69].

At this time, the model attempts to maximize the benefit of the data-center, as the revenue for all tasks minus the power cost, and minus a penalty for each migration done towards the previous schedule (excluding finished and new-coming VMs). That penalty can be considered, i.e., as the nonpayment for the migration time, an insurance for the unfulfilled SLA, or predict the risk of an SLA failure and its consequences. As an example, we will consider the nonpayment for that migration time, but keeping in mind all the other options:

Variables:	$schedule[Hosts, VMs]$, as Integer Binary ; representing the Schedule
	$pr_i[Hosts]$, as Integer Binary ; representing the use for each host of its i-essim CPU
Parameters:	pwr_i, as the power consumed by an i-essim CPU
	$cpus(h)$, as CPUs existing in host h
	$cons(j)$, as the CPU consumption of VM j
Functions:	$migr(schedule_{old}, schedule)$, as $\frac{1}{2} \cdot (schedule_{old} \oplus schedule)$ (w.o. leaving or new VMs)
Maximize:	$+ \sum_{j,h \in VMs, Hosts} schedule[h, j] \cdot$ Revenue VM/Hour
	$- \sum_{i,h \in cpus(h), Hosts} (pr_i[h] \cdot pwr_i) \cdot$ power price
	$- migr(schedule_{old}, schedule) \cdot time_{migration} \cdot$ Revenue VM/Hour
Constraints:	$Procesors\{i, h \in cpus(h), Hosts\} := pr_i[h] \geq pr_{i+1}[h]$
	$Unique\{j \in VMs\} := \sum_{h \in Hosts} schedule[h, j] \leq 1$
	$MaxCPU\{h \in hosts\} := \sum_{i \in cpus(h)} pr_i[h] \leq cpus(h)$
	$Capacity\{h \in Hosts\} := \sum_{j \in VMs} schedule[h, j] \cdot cons(j) \leq \sum_{i \in cpus(h)} pr_i[h]$

Note that the impact of migration can affect CPU loads and many other relevant factors on the system, depending on the relation of the migration time and re-scheduling time.

6.2.4 Quality of Service as a factor

Often systems can be enough flexible to allow some tolerance to Quality of Service, and that means that VMs are not strictly tight to a fixed QoS, and sometimes this QoS can be relaxed letting the system to certain overload in order to improve consolidation and reduce power consumption. By relaxing the QoS, some penalization can be applied specified in the SLA, so the schedule is able to alter the demands of each VM by attending at the economic consequences.

In order to define the level of accomplishment of the VM goals and SLA conditions, the concept *Health* is defined. The health is an index indicating how well is performing a VM, and often depends on the amount of the required resources are received. A value of 1 means that the VM is performing optimally, and 0 that the VM is not running nor progressing in its execution.

When turning the VMs CPU requirement into variables, in a range between a maximum CPU (original required) and minimum CPU (SLA failure assured), we can estimate the health level using knowledge from the system, heuristics, or prediction [30] comparing the offered CPU with the maximum required CPU. This health value can be used to establish the penalty to be subtracted to the revenue, or by default use a fixed value explicitly indicated in the SLA.

The function to be optimized includes now the new factor, by scaling the revenue with the health function, adding the before fixed VM requirements as variables, and establishing a range for that variable. Changes to be introduced are reflected in the following parts of the problem:

Variables:	$jcpu[VMs]$, as Integer ; representing the CPU usage of VM j
Parameters:	$consmin(j)$, as the minimum required CPU consumption of VM j
	$consmax(j)$, as the maximum required CPU consumption of VM j
Functions:	$health(j)$, level of QoS of the VM j result of $jcpu[j] \sim \langle consmin(j), consmax(j) \rangle$
Constraints:	$MarginCPU\{j \in VMs\} := 0 < consmin(j) \leq jcpu[j] \leq consmax(j)$
	$Capacity\{h \in Hosts\} := \sum_{j \in VMs} schedule[h,j] \cdot jcpu[j] \leq \sum_{i \in cpus(h)} pr_i[h]$
Maximize:	$+ \sum_{j,h \in VMs,Hosts}(schedule[h,j]) \cdot$ Revenue VM/Hour
	$- \sum_{j \in VMs}(1 - health(j)) \cdot$ QoS agreed penalty
	$- \sum_{i,h \in cpus(h),Hosts}(pr_i[h] \cdot pwr_i) \cdot$ power price
	$- migr(schedule_{old}, schedule) \cdot time_{migration} \cdot$ Revenue VM/Hour

At this moment, note that the problem as written as shown loses its linearity as *scheduling* and $jcpu$ are being multiplied as being both variables of the same problem, and this requires to re-write some details in order to obtain again a linear problem.

Having the variable schedule as a binary values matrix, the $Capacity$ constraint can be understood as

$$Capacity\{h \in Hosts\} := \sum_{j \in VMs}[\text{ if } (schedule[h,j] = 1)\ jcpu[j] \text{ else } 0] \leq \sum_{i \in cpus(h)} pr_i[h]$$

For this, a change of variables can be performed and rewrite the constraint as

$$\begin{aligned} Capacity\{h \in Hosts\} &:= \textstyle\sum_{j \in VMs} quota[h,j] \leq \sum_{i \in cpus(h)} pr_i[h] \\ quota[h,j] &= \text{ if } (schedule[h,j] = 1)\ jcpu[j] \text{ else } 0 \end{aligned}$$

This *quota* condition formulated as a set of linear constraints in the following way

$$\begin{aligned} quota[h,j] &\geq schedule[h,j] \\ quota[h,j] &\leq schedule[h,j] \cdot BigConst_1 \\ quota[h,j] - jcpu[j] &\leq (1 - schedule[h,j]) \\ jcpu[j] - quota[h,j] &\leq (1 - schedule[h,j]) \cdot BigConst_2 \end{aligned}$$

being $BigConst_i$ constant values always big enough to surpass the sum of the VM requirement ranges, just assuring that the inequalities of conditions always are valid letting the model to perform the target condition. Also remember that $jcpu[j]$ is an integer greater than zero.

The final integer linear program is as follows:

Variables:	$schedule[Hosts, VMs]$, as Integer Binary ; representing the Schedule
	$quota[Hosts, VMs]$, as Integer ; representing CPU quota for each VM in each host
	$pr_i[Hosts]$, as Integer Binary ; representing the use for each host of its i-essim CPU
	$jcpu[VMs]$, as Integer ; representing the CPU usage of VM j
Parameters:	pwr_i, as the power consumed by an i-essim CPU
	$cpus(h)$, as CPUs existing in host h
	$consmin(j)$, as the minimum required CPU consumption of VM j
	$consmax(j)$, as the maximum required CPU consumption of VM j
Functions:	$migr(schedule_{old}, schedule)$, as $\frac{1}{2} \cdot (schedule_{old} \oplus schedule)$ (w.o. leaving or new VMs)
	$health(j)$, level of QoS of the VM j result of $jcpu[j] \sim \langle consmin(j), consmax(j) \rangle$
Maximize:	$+ \sum_{j,h \in VMs,Hosts}(schedule[h,j]) \cdot$ Revenue VM/Hour
	$- \sum_{j \in VMs}(1 - health(j)) \cdot$ QoS agreed penalty
	$- \sum_{i,h \in cpus(h),Hosts}(pr_i[h] \cdot pwr_i) \cdot$ power price
	$- migr(schedule_{old}, schedule) \cdot time_{migration} \cdot$ Revenue VM/Hour
Constraints:	$Procesors\{i, h \in cpus(h), Hosts\} := pr_i[h] \geq pr_{i+1}[h]$
	$Unique\{j \in VMs\} := \sum_{h \in Hosts} schedule[h,j] \leq 1$
	$MaxCPU\{h \in hosts\} := \sum_{i \in cpus(h)} pr_i[h] \leq cpus(h)$
	$Capacity\{h \in Hosts\} := \sum_{j \in VMs} quota[h,j] \leq \sum_{i \in cpus(h)} pr_i[h]$
	$MarginCPU\{j \in VMs\} := 0 < consmin(j) \leq jcpu[j] \leq consmax(j)$
	$QoSAux1\{j, h \in VMs, hosts\} := quota[h,j] \geq schedule[h,j]$
	$QoSAux2\{j, h \in VMs, hosts\} := quota[h,j] \leq schedule[h,j] \cdot BigConst_1$
	$QoSAux3\{j, h \in VMs, hosts\} := quota[h,j] - jcpu[j] \leq (1 - schedule[h,j])$
	$QoSAux4\{j, h \in VMs, hosts\} := jcpu[j] - quota[h,j] \leq (1 - schedule[h,j]) \cdot BigConst_2$

6.3 Studying the Behavior of the Model

In this section the method is evaluated, exposing the scenario used in order to test the approach. Also all the decisions taken in order to set up a representative testbed are exposed, building a good scenario to focus on the method while letting the addition of new elements or variables.

6.3.1 Programming and Simulation Environment

The experiments performed to test this approach have been done simulating a real workload and also simulating a datacenter formed by different sized testbed machines. In this occasion, as the important thing to be evaluated and tested is the methodology and algorithms for scheduling and making decisions, a real data-center and a scheduling mechanisms have been recreated in R [80], extracting the behavior formulas and data-center working modules directly from the cloud simulator EEFSIM, explained in previous chapter 4.

For this approach, the modules and formulas have been implemented in R, using workloads and data-center configurations generated from real loads and examples. The example simulated datacenter is presented in Table 6.1.

The used workload corresponds to a transactional workload on application web-servers, obtained from a real web-applications workload. These web-applications simulate customers who wants to run their services on top of the the provider. The behavior of these applications deployed corresponds with the one of SPECweb2009 e-Commerce application [148] which is used

Number of Hosts	CPU	Memory
20	4 @ 3GHz	4 GB
10	2 @ 3GHz	4 GB
10	1 @ 3GHz	4 GB

Table 6.1: Properties of the simulated datacenter

as web-based application and its model has been obtained by stressing this application (deployed on a Tomcat v5.5 with an hybrid architecture) with different input loads and with different processing units. The details of the workload are shown in Table 6.2.

Type	SLA type	#VMs	Description	Mean duration
Web	Performance	10	Monday	86400”
Web	Performance	10	Tuesday	86400”
Web	Performance	10	Wednesday	86400”
Web	Performance	10	Thursday	86400”
Web	Performance	10	Friday	86400”
Web	Performance	5	Saturday	86400”
Web	Performance	5	Sunday	86400”
Web	Performance	20	One week	604800”

Table 6.2: Workload details

This modeling, as proposed in [87], focuses on the response time high-level metric and, relating this metric with both incoming users load and CPU usage of the server. The modeling details include an *stationary* response time (RT), when the incoming load causes a CPU utilization less than 60%; a *slightly increased* RT when this CPU utilization is between 60% and 80%; and a *polynomial* behaved RT when the server is overloaded.

The power consumption is also determined in EEFSIM, where the real power consumption is measured using different workloads in a 4-CPU computer whose kernel includes some energy efficiency policies. As seen in previously the wattage increases with the CPU requirements on a physical machine, but this increment is lower with each extra processor used on a machine. This is the reason why consolidation optimizes the power consumption. Also, in the experiments of this approach the turning on and off of used and idle machines is considered to reduce the consumption, as seen in works like [70, 30].

In order to set the economic values of each involved element, the EEFSIM and its related works (as seen below) established that providers behave as a cloud provider similar to Amazon EC2, where users will rent some VMs in order to run their tasks. The pricing system for the VMs is similar to the one EC2 uses and medium instances with high CPU load are assumed, which have a cost of 0.17 euro/hour (EC2 pricing in Europe). Also the electricity pricing used is the Spanish one, that is 0.09 euro/KWh [60]. The VM migration process can be performed in several ways: as stopping the service and resuming it after full VM data is copied, or creating a copy into the physical destination before deleting the original and then change the load flow. Anyway, this process may cause in some occasions a interruption of service or just degrade the user experience for a short period of time, and as example of a simple valid SLA in this model is proposed a *migration penalty* consisting in a economic compensation to the client (i.e. the execution during a migration period becomes for free). For the present approach, the maximum migration time is set to 5 minutes, while each scheduling round is set to 1 hour, according to the variability of the performed workloads.

As a parameter defining the Quality of Service at low level, the concept *health* is defined as the ratio between obtained CPU and the required CPU in function of the client load. The maximum loss of QoS permitted can be determined in each VM SLA, but as an example and case of study here a range $[0.8, 1.0]$ is determined as acceptable offered CPU, and a penalization of $(1 - \frac{CPU_{VM}}{OfferedCPU_{VM}}) * 0.17 euro/hour$ (revenue of VM per hour) is applied in order to add a benefit factor. These parameters can vary in function of each VM SLA adjusting the minimum

accepted health factor, the way of calculating the health factor, and the revenue of the VM.

Finally, in order to find good scheduling results implementing the here described integer linear programs, different popular implementations of the simplex algorithm [52] have been used. For the here exposed results shown in next subsections the GNU Linear Programming Kit (glpk) has been used, but other options have been run in order to find enhanced simplex algorithms like lpsolver or CPLEX [85]. Here we will not discuss a comparison between them, as lpsolver often find better solutions than glpk with the same running time and CPLEX is able to find optimal solutions in seconds instead of minutes like the other ones. The glpk has been selected here just because is the easiest to apply and experiment with, without a license requirement (i.e. CPLEX requires a limited payment license to be run).

6.3.2 Experiments

In this subsection the model is run focusing on each of its factors (power, economic, qos) one by one. The integer linear program can be compared in an *only-power* version, looking for minimize the power consumption; in an *economic versus power* version, looking for increase the revenue minus the power economic cost; and in the full version, including the tolerance to some loss of QoS in favor to the power saving.

As expected, the experiments reveal that the power model is the one that performs the maximum number of migrations as its priority is to reduce the consumption and in a natural way the scheduler consolidates the VMs at each round. This model only contemplates options that reduce the number of processors used and machine on-line, and this brings the policy of migrating VMs as many times as required. Without any restriction referring to migration penalization, this model is the one that performs best in an energetic way. The "economic vs power" and full models do not optimize as good in energy, mainly because they include restrictions at applying this migration policy. The power consumption is shown in Figure 6.1.

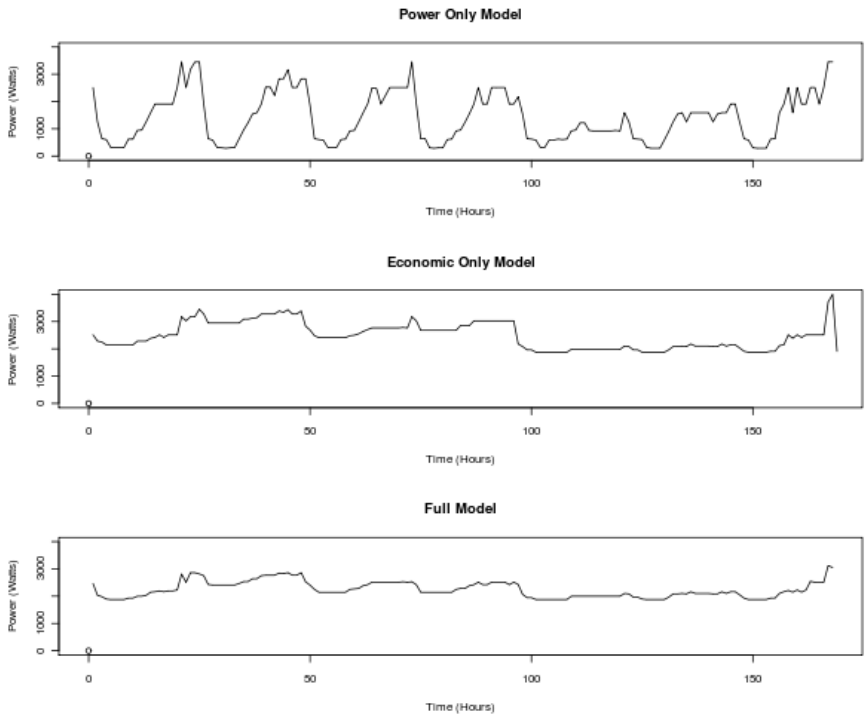


Figure 6.1: Power consumption in for each model

The "economic vs power" model and the full model include the policy of reducing the number of migrations by penalizing them. This makes that their power saving is reduced compared with the power only model, but here the schedule is reducing the number of times a VM can be

temporally "out of service". As Figure 6.1 shows, the "economic vs power" model and full model power consumption is similar, also QoS tends to improve the economic only model, as lets the reduction of resource quotas to improve power saving.

When introducing the explained migration penalty the optimization changes as the economic based models improve the revenue brought by maintaining machines running without being migrated. As seen in Figure 6.2, power model degrades in revenue as migrations are done without restrictions. The model tolerating some QoS loss maintains similar revenues than the economic model reducing the number of migrations by reducing the quota of resources, also penalized by the explained *health rule*. This temporary reduction of resources in order to reduce the migration needs is shown in Figure 6.3, where clearly the full model nearly avoids migration.

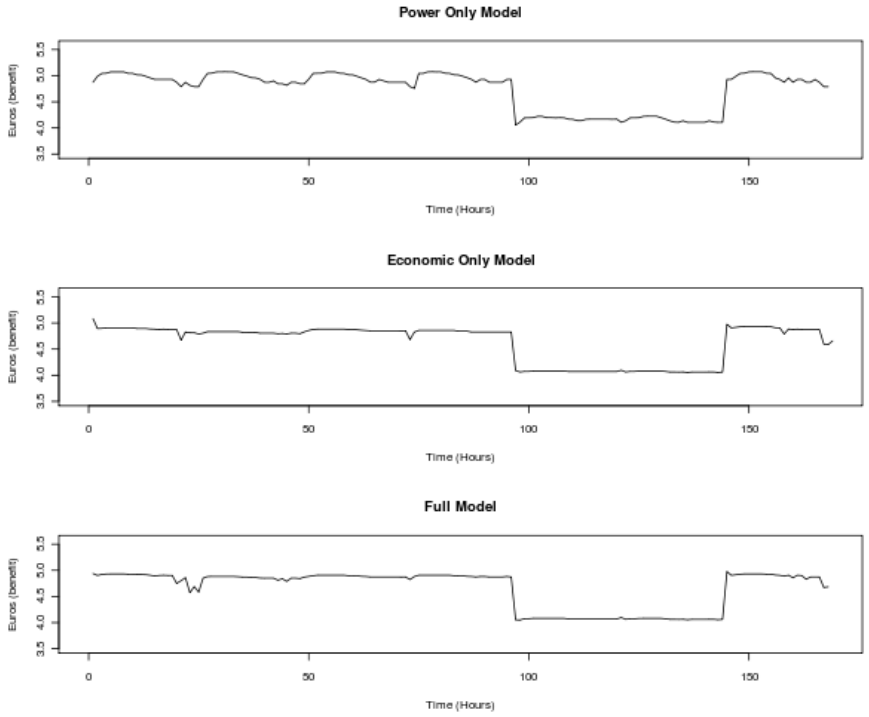


Figure 6.2: Benefit obtained from each model

It is important to remark the fact that the reason for penalizing migrations is that each migration brings chances of interrupting the service during a brief period of time or any other drawback of this technology like having duplicated services or VMs. For this reason economic and QoS models should be preferred than power-only models, as they attempt to reduce the number of chances of service interruption. For future models, whether knowing the exact migration specifications, this chances and exact SLA penalization could be added to the optimization model, so the decision of migrating a VM versus do not do it would be better weighted assuming a more accurate probability of interruption of service.

Table 6.3 summarizes the consumption, benefit and number of migrations performed by each model. In the table a Round Robin scheduling algorithm has been added in order to compare with a naive scheduler. This Round Robin algorithm performs all allocations always assuring that the VM fits into the target machine.

The coefficients applied in the problem weighting each task execution, the power price, the migration penalty and the QoS penalty define the policy to be followed in order to optimize. This case of study assumed a policy that obviously can change when prices change, and when technology changes. If migration technology is improved reducing the maximum migration time, the penalty of migration can be reduced so in some occasions moving VMs will be better than sacrificing VMs performance. Furthermore, a modeling of each kind of VM could provide a characterization of the probability of "out of service" when migrating a specific VM and adjust

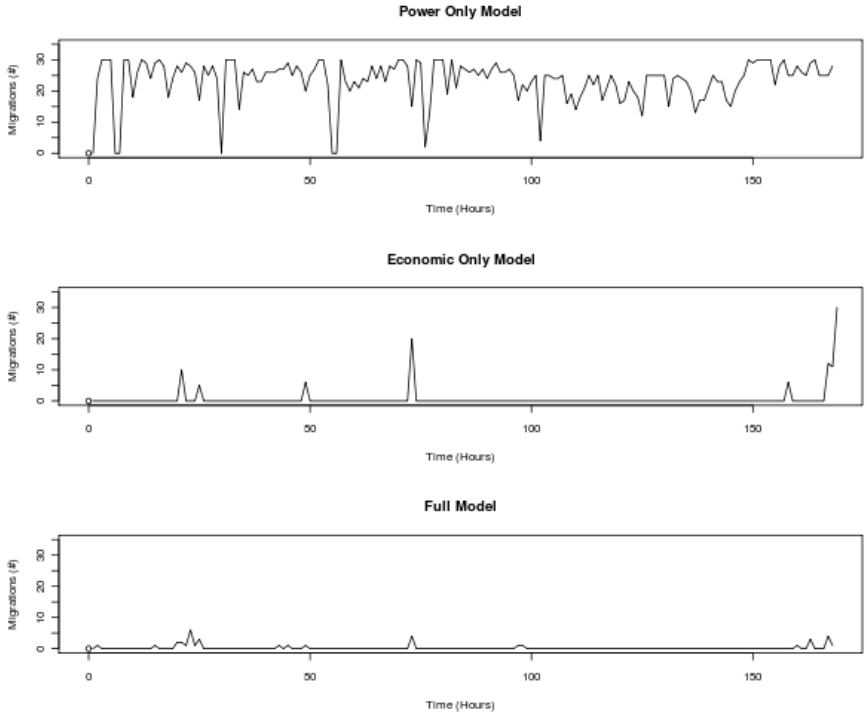


Figure 6.3: Number of migrations for each model

Model	Power cons.(watts)	Benefit (euro)	Migrations (#)
Round Robin	1687717.0	598.1375	4712
Power Only	236398.3	794.5513	3944
Economic Oriented	419813.1	782.7054	100
QoS relaxation	376801.8	780.8093	35

Table 6.3: Comparative between models

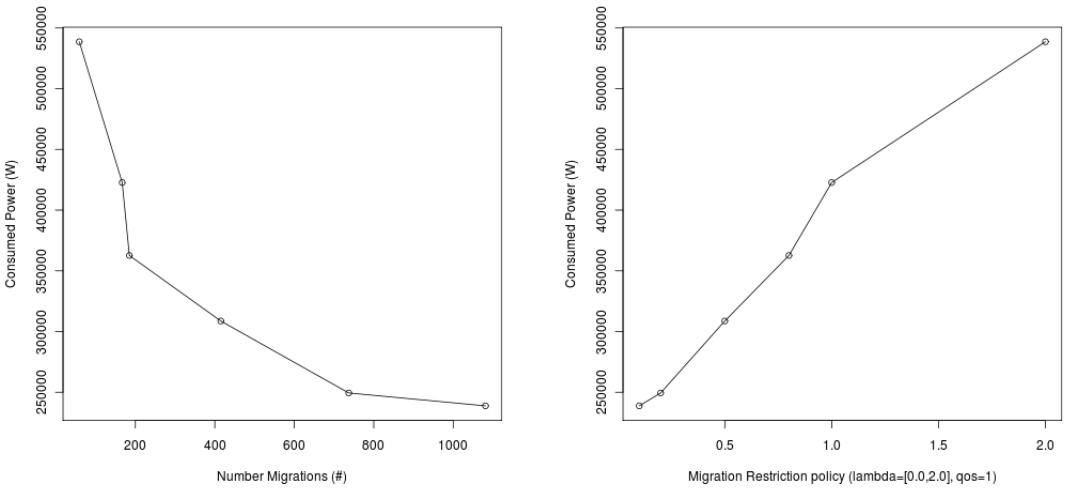
the migration penalty depending on the VM to be moved. This also would provide an accurate selection of which VMs are more candidates to be moved when the situation requires it. The same occurs with the QoS threshold, being modified for each VM and circumstance, so some VMs are more or less able to be economized depending on their execution time. Here a *VM migration vs VM compression* situation is faced, depending entirely of the policy applied at each moment.

Integer linear programming can be used directly as a scheduling algorithm in some occasions, when engineers and designers know that the situations to control will be solvable in short time (seconds or minutes). But unfortunately, simplex algorithm can become exponential, and the integer optimization usually is exhaustive, and in some situations finding a good enough solution requires excessive search time. For this case of study the average time in order to find a good solution: an average of 0.2447% difference respect the simplex relaxed lower bound [min 0.0%, mean 0.2447%, max 7.0530%], with a mean of 14 seconds time search for scheduling often 30 VMs x 40 machines (with a maximum search time of 2 minutes, not often reached). Next section will show that when expanding and adding more complex constraints to the model, this computing time will increase significantly, and alternatives will be required as an alternative to the exhaustive solving.

6.3.3 Impact of Policies and Trade-offs

Finally by looking at the proposed model, each element included in the benefit function can be seen as an element of a policy system. Denoting Benefit with B, Revenue with R, Migrations with M, Loss of QoS with Q and Power with P as: $B = R - M - Q - P$, the elements M and Q represent the policy of the system in front of the possibility of migrations and task resources compression. The impact of each policy can be measured by testing them using multipliers in order to check how much vary the power and real benefit while varying the enforcement of the migration penalty or the loss of QoS penalty. Setting $\tilde{B} = R - (\lambda M + \varepsilon Q) - P$ and a policy $\Pi = \langle \lambda, \varepsilon \rangle$ indicating how each element will be weighted in the MILP solver, some tests are run to obtain the relation between each migration and loss of QoS for each power saving unit. A first set of executions test the migration policy by varying its weight (λ) and forbidding the loss of QoS ($\varepsilon = \infty$), while other set of executions test the tolerance to QoS loss by forbidding the migrations ($\lambda = \infty$) and varying the weights of the QoS loss penalty (ε) and letting no minimum bound to the QoS loss.

In the case of the migration policy test, it is obvious that the migration technique is always applied in order to reduce the number of running processors in the whole system, so as shown in Figure 6.4a the number of migrations is reversely related to the consumed power. This can be understood as "migrations relief power consumption, while paying a price", and it is the relation between the cost per kWh and the penalty for each migration. If the price of the amount of power a migration can save is bigger than the cost of performing this migration, do it is better. So policies based in migration can modify the migration penalty in order to adjust the number of migrations to be performed in front of a determined amount of power to be saved, as seen in Figure 6.4b.



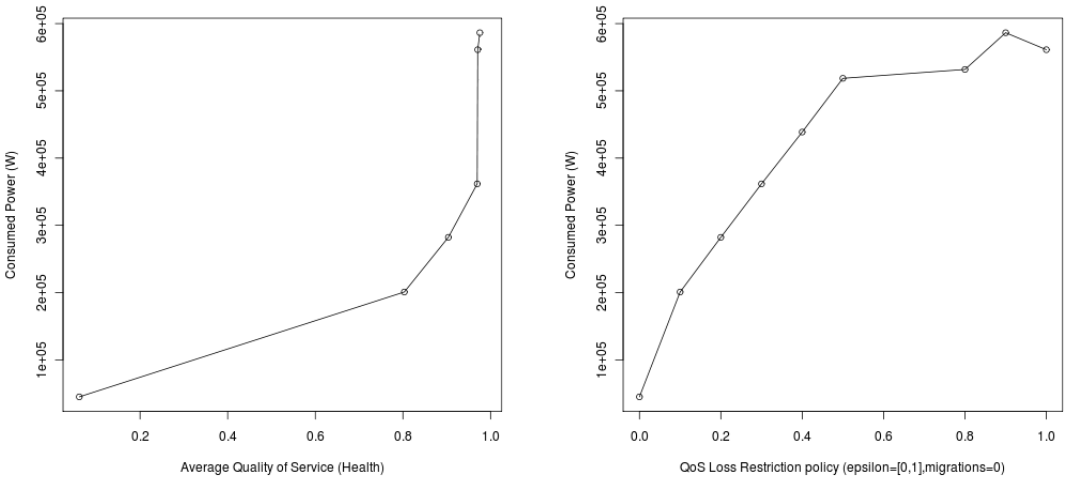
(a) Power vs Number of Migrations

(b) Power vs Migration Restriction

Figure 6.4: Power versus migrations and migration restriction policy

Finding the best fitting linear regression from the power vs. migration results, the generated model matches with the form: $\hat{Power} \simeq MinimumRequiredPower + (maxPossibleMigrations - \#Migrations) * MachineBasePower$. This means that in average for each migration performed, the power of having on-line one machine is saved, demonstrating that the method performs first that migrations that improve most the power saving. In this case, performing a migration that allows to shut down a machine is preferred to perform a migration that just saves a few watts. From Figure 6.4a we can distinguish three parts, the first one where the policy optimizes only with migrations that directly shut-down machines (0-200 migrations), the second one where migrations shut-down machines and reduce the number of running processors (200-800 migrations), and the third part where saturation comes and no for more migrations done the power will be easily reduced due to reaching the minimum required power.

In the case of Power versus Loss of QoS it can be observed that the power is related to the level of Quality of Service (in terms of resource occupation) and the restrictions to be applied on QoS loss, as seen in Figures 6.5a and 6.5b.



(a) Power vs QoS level

(b) Power vs QoS Loss Restriction

Figure 6.5: Power versus QoS (health) and QoS Loss restriction policy

Note that the policy values λ and ε are directly related to the number of migrations and QoS respectively. So in tuning the model establishing the weight of each condition (or policy or price), a trade-off between the provider revenue per application and the penalties for each migration and loss of QoS. The MILP model presented here is complemented by finding these penalization weights by applying the policy that minimizes the power consumption while granting a good QoS level and reducing the possibilities of service outage caused by migrations. Also with this information the model can be adjusted if business policies decide not to allow more than X migrations per time unit, or not to set a determined average of QoS allowed (or also a minimum QoS level without restricting it in the MILP constraints).

6.4 Discussion on MILP Modeling of DataCenters

In order to optimize the scheduling and management of datacenters several factors must be kept in mind, like the economic ones (obtaining revenues from task executions and resource usage), bringing good service quality to the customers and clients, and also the energy-related factors required to run infrastructures. This optimization problem can be modeled as an integer linear program, so MILP methods can help in order to solve it, or also help to find other accurate models and solving methods.

Taking the advantages of virtualization and consolidation techniques, this model can address policies that focuses on energy-saving goals, like using migration to turn-off unnecessary machines, or lend tasks fewer resources in order to save power. The results obtained show that the MILP model allows to find optimal (or near-optimal) solutions bringing good QoS levels (by giving tasks the required resources), and each new technique introduced in form of rules and constraints allows the scheduler new options to expand the model.

Also, from the obtained results an expected fact can be observed: in order to maximize economic benefit the usually most "sacrified" factor is the cheapest one. In this case it is power consumption. When the schedules attempts to make a decision that maximizes revenue while not degrading QoS, migrating as minimum as possible and consuming low power, as power consumption is economically weaker than the revenue obtained by the rest of the system, it receives less attention and so benefit increases response to higher power consumptions.

A way to adjust this energy consumption is to play with policies, adjusting weight to power-saving techniques in order to relax their usage penalties, so here the system would risk some economical benefit or QoS in order to save some energy. Also policies decide which kind of risks are to be taken between chances of service outage caused by migrations or chances of QoS loss by reducing the given resources to tasks. Future work is addressed to research in the way of decision makers and modeling the kind of risks to be taken at each moment, so static policies can turn into dynamic depending on the context, the load and the interests of providers and clients.

6.5 Introducing Machine Learning into the Model

In this section we present some experiments done to compare the MILP solving with some generic approximate algorithms, in order to prepare the research for the next chapter, where we will apply machine learning to model and predict component behavior, to be applied as part of the fitting functions in our mathematical modeling. Many of the concepts introduced here will be more completely explained and detailed in the next Chapter 7.

A cloud can be viewed as a set of VMs or tasks to be distributed along a set of resources, so the main decisions to make are what resources are given to what VMs, assuring that each VM receives enough resources to be executed properly with respect to global goals and customer requirements. In order to make this management more efficient, here we employ methods from Machine Learning. This is a subfield of the data mining area in charge of modeling systems from real examples of their past behavior. These models can then be used to predict future behaviors. The advantage of Machine Learning, compared to explicit expert modeling, is that it applies when systems are complex enough that no human expert can explore all relevant model possibilities, or in domains when no experts exist, or when the system is so changing over time that models have to be rebuilt autonomously over time.

6.5.1 Web-Services and Prediction

Each VM has its own behavior and resource requirements. Often these requirements are not known in advance, so the system manages the amount of resources at each scheduling round taking as valid values the previous monitored demand. In other situations the user providing the VM (customer) provides a guess on average or maximum requirements, and either the system trusts the (often inaccurate) customer advice, or else overbooks resources. In this work we focus on web services, as their requirements are more sensitive towards volume of load than other high-performance tasks, and tend to change unpredictably in time. The methodology proposed in this work includes learning to predict, for each kind of VM entering the system, the amount of required resources as a function of the kind of load it is receiving.

When a new kind of VM arrives we train a model mapping load to resources for that particular kind. A load is specified as a vector of load attributes determined beforehand (such as requests per second, bytes transferred per request or processing time per request). Given such a load, the model provides a vector describing the predicted usage of a number of resources (CPU, memory, bandwidth...). Kinds of VMs could be web services running on specific software packages like Apache v.X, Tomcat v.Y, with attached modules like PHP or MySQL DB services. Each execution of the VM provides pairs of ⟨workload attributes, resources used⟩ that can be used for (further) training the model describing this kind of VM.

Once in the scheduling phase, the goal is to assign as few resources as possible to running VMs keeping user satisfaction but reducing power usage. We assume in this work that the notion of user-satisfaction is captured by SLAs for each VM, that is an agreement between customer and provider about the quality of service that the provider assures the VM will offer externally. For a web service, and for this case of study, a relevant SLA term is the maximum response time the service will provide per request. A commonly used form of SLA function for response time RT

$$SLA(VM_i) = max(min(1 - \alpha\frac{RT - RT_0}{RT_0}, 1), 0)$$

where RT_0 is the response time desired by the customer, and the SLA level goes from 1 to 0 when the provided RT surpasses the desired time, with 0 fulfillment at β times RT_0 (where $\alpha = \frac{1}{\beta - 1}$).

For our purposes, this SLA fulfillment function has the advantage of being piecewise linear, and so can be easily integrated into a *linear* programming model.

This response time factor can be obtained *a posteriori*, by monitoring clients requests and times for responses, but often the scheduler is interested in predict this information *a priori*. For a given application and system a behavior model can also be learned from a training run, adding different kinds of stress to the execution, in order to learn the relation between amount of load, amount of required resources, amount of given resources and response time. This learned function allows the scheduler to predict response times by trying different amounts of resource allocations.

6.5.2 Adapting the Mathematical Model

The introduction of predictors into the explained mathematical model are done in the fitting functions and the input for resource parameters. This is, instead of providing as input the amount of CPU a virtualized web-service will require, we will provide the input about load for it and predict from it the required CPU for the web-service; also we will predict for each placement the response time for the web-service (and thus the SLA fulfillment), and add this SLA reward to the optimization function:

Maximize:

$Profit = \sum f_{revenue}(VM_i, SLA(VM_i)) - f_{powercost}(Power) - f_{migpenalty}(migrations)$

Subject To:

(1) $\forall i \in J : f_{MaxRes}(i) = f_{CPU}(Load_i)$

(2) $\forall i \in J : \hat{RT}_i = f_{RT}(Load_i, ReqRes_i, GivenRes_i)$

(3) $\forall i \in J : SLA(i) = f_{SLA}(\hat{RT}_i, RT_{i,0}, \alpha)$

Here we faced the possibility of using an exact solver and different heuristics and ad-hoc solvers. We used GLPK [65] MILP solver and two classical approximate algorithms (First-fit and ordered Best-fit, as seen in Algorithms 6,7), also the ad-hoc λ-Round Robin seen in previous chapters, in order to compare optimal values and cost in time. Besides the fact that they are faster than exact solvers, an additional advantage of such heuristics is that they can deal with possibly non-linear constraints, letting to deal with non-linearity in the model in next works.

Algorithm 6 Descending First-Fit algorithm

```
for each vm i:
    get_data(i);
    res_quota[i] <- get_required_resources(i);
end for
for each host j:
    res_avail[j] <- get_total_resources(j);
end for
order[] <- order_by_demand(vms,res_quota[],desc);
for each vm v in order[]:
    for each host h and #processors:
        if ( fit(v,h,p,res_quota[v],res_avail[h]) ) :
            assign_vm_to_host(c_host,v);
            update_resources(res_avail[c_host],v);
            continue next vm;
        end if
    end for
end for
```

Algorithm 7 Descending Best-Fit algorithm

```
for each vm i:
    get_data(i);
    res_quota[i] <- get_required_resources(i);
end for
for each host j:
    res_avail[j] <- get_total_resources(j);
end for
order[] <- order_by_demand(vms,res_quota[],desc);
for each vm v in order[]:
    best_profit <- 0;
    c_host <- 0;
    for each host h and #processors:
        profit <- profit(v,h,p,res_quota[v],res_avail[h]);
        if (profit > best_profit) :
            best_profit <- profit;
            c_host <- h;
        end if
    end for
    assign_vm_to_host(c_host,v);
    update_resources(res_avail[c_host],v);
end for
```

6.5.3 Experiments

Environment and Workload

The experiments to test this application of the model are done with the simulated environment explained before, using the HPC machines environment (using 40 Intel Xeon 4 Core simulated PMs). Also we introduce here the the LiBCN Workload 2010 [26], as explained also in chapter 4. As we are using a simulated environment to test the algorithms, we do not reproduce the exact load of the LiBCN'10 over a real VM environment yet. But we use the indicative load values to determine the performance of each virtualized web-service, also substituting some of the human-expert placed models by our obtained machine learning models (CPU and RT). The following stress experiments represent a run of these workload pieces during 4 days (monday to thursday).

As a parameter defining the Quality of Service, the response time at the data-center output is used. As a case of study a penalization of $SLA(VM) \cdot Revenue(VM)$ (revenue of VM per hour) is applied in order to add a profit factor. The VMs on workload have as RT_0 the values of $0.004s$ or 0.008 (each VM can have different SLA terms), as experiments with the Xeon test machine shown that it is a reasonable response value obtained by the web service without stress or important interferences. The initial α parameter is set to 1 (SLA fulfillment is 0 if RT exceeds $2RT_0$). Besides these basic settings, VM pricing, power pricing, and the SLAα parameter have been set to other values in some of the experiments.

Learning Process and Validation

The first function to be learned is Load $\sim$ CPU for a given web-service, as for this first approach we address CPU prediction and leave the full study of memory and IO models for the next chapter. The value to be predicted is the CPU consumed by the web service inside a VM, and in the workbench scenario built to collect data from the VM, CPU is the critical resource disposing of enough memory and bandwidth, so the relevant data describing the usage of CPU is reduced to requests per second, average bytes per request and average processing time per request.

The chosen learning method is a regression tree algorithm *M5P* explained in chapter 2, also following the selection methodology also explained there. The model obtained mean squared error is around 9.9%CPU during the model selection test and 12%CPU for the validation test, each one with around 3000 instances (two different subsets of the workload). This amount of

accuracy should suffice for useful prediction.

The second model to be learned is Load $\sim$ Response Time. The objective is to learn how the web service VMs RT behaves in front of stress, so we can predict RTs for different possible VM schedules. After a feature selection process, the most relevant attributes selected to predict the RT for this situation are the load attributes, the total used CPU from physical machine, the given CPU to the VM and the required CPU from the web service VM. Note that the memory attributes do not appear this time, as memory becomes a non-critical resource for the experiments done with the Xeon machine, while in previous experiments using a non-HPC architecture (Celeron M, single core @ 2Ghz, 512Mb RAM) RAM memory became a critical resource and it was present during the prediction process.

As a learning method, we simply used linear regression because it gave reasonably good results and is conveniently linear to be plugged in constraint (2). The selected algorithm is the linear regression method. With this model, the mean squared error obtained from the predictions is around $2.374979 \cdot 10^{-}5s$ (stdev $0.004s$), each one with around 3000 instances (another two different subsets of the workload).

MILP and Algorithms testing

After performing some complete experiments, we found that given our scenario 4 minutes are enough to find the optimal solution, but still leaving part of the search space without exploring (with no better solution). So for current experiments, we set a time limit of 4 minutes for the MILP solver. Further, for these experiments the λ for the Round Robin is set to 30 because the study in Chapter 5 seemed to indicate it is optimal for settings similar to ours. Also, the best-fit algorithm uses as input the VMs ordered by descending minimum CPU required. Table 6.4 show the result for the run of all algorithms, and its statistic information from running each method 10 times.

		Avg QoS	Energy	Profit	Migs	Hosts
FF	mean	0.6047	294.5	210.349	1488	934
	stdev	0.0060	2.807	2.47682	16	9.404
	max	0.6176	298.4	215.592	1513	947
	min	0.5975	289.9	207.181	1465	919
λ-RR	mean	0.7136	355.1	240.232	523	1228
	stdev	0.0059	2.946	2.34982	10	9.555
	max	0.7232	359.8	244.184	536	1245
	min	0.7054	351.7	236.839	506	1214
BF	mean	0.8649	204.4	320.363	1688	663
	stdev	0.0013	1.677	0.52711	23	5.926
	max	0.8669	206.8	321.123	1717	671
	min	0.8631	201.7	319.562	1652	653
MILP Solver	mean	0.8772	173.8	327.406	1988	586
	stdev	0.0018	3.887	0.78141	20	14.792
	max	0.8782	179.2	328.767	2012	605
	min	0.8740	169.8	326.849	1963	571

Power Cost = 0.09e/Kwh, VM Revenue = 0.17e, MaxRT = $2 \cdot RT_0$
[Energy = kWh; Profit = Euro]

Table 6.4: Statistical analysis for each algorithm and metric.

As can be seen here, the exhaustive (sub)-optimal solution obtains higher profit than the other methods, as its solution is (with the required time) complete and exhaustive. The ordered best-fit solution, however, is very close - often by less than 1%. Furthermore, it is far faster – it requires only 4 seconds instead of 4 minutes in this example (in some occasions +4 hours!). Additionally, as discussed before, it can potentially accommodate non-linear constraints better than MILP. It is thus a strong candidate to replace the exhaustive search method in future work. Figure 6.6 shows dynamically the comparative of power and SLA fulfillment level over time for

the different used schedulers.

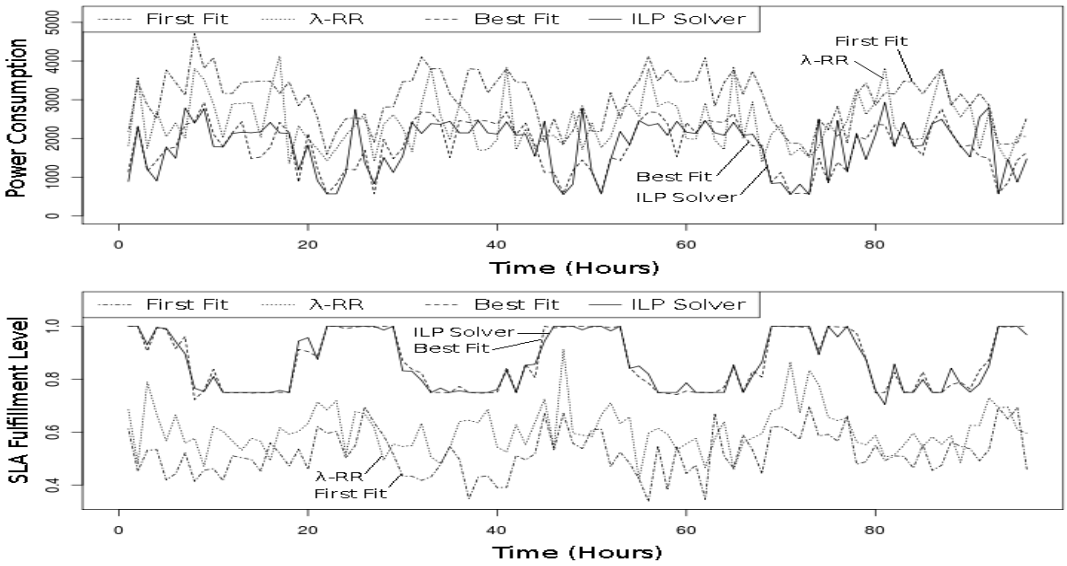


Figure 6.6: Power and SLA Comparative on the Schedulers

Method	Energy	Migs	Profit	AvgQoS	RangeQoS	UsedCPU	UsedHosts
FirstFit	290.2	1421	177.29	0.519	[0.524,1]	3523	921
λ-RR	358.9	520	241.04	0.716	[0.001,1]	3917	1237
Ordered BF	203.1	991	304.88	0.860	[0.316,1]	2250	658
MILP Solver	194.9	232	321.31	0.877	[0.346,1]	1702	662

Parameters: Power Cost = 0.09 euro/kWh, VM Revenue = 0.17 euro, MaxRT = $2 \cdot RT_0$
Migration Penalty = 0.014 euro; [Energy = kWh; Profit = Euro]

Table 6.5: Scheduling Comparative between techniques applying Migration Penalties

€/kWh	Energy	Migs	Profit	AvgQoS	RangeQoS	T.Spent	GapLB	CPUs	Hosts
0.00	365.5	195	353.55	0.883	[0.368,1]	94.88	0.047	4600	1150
0.01	242.8	225	340.00	0.879	[0.351,1]	115.18	0.055	1798	846
0.09	194.9	232	321.31	0.877	[0.346,1]	191.98	0.213	1702	662
0.14	170.7	330	312.52	0.878	[0.355,1]	216.12	0.273	1619	571
0.45	150.2	445	264.77	0.870	[0.310,1]	230.41	0.637	1492	498
0.90	137.2	550	199.36	0.857	[0.229,1]	234.02	0.966	1380	455
1.80	128.0	600	81.18	0.840	[0.190,1]	239.26	1.574	1327	422
3.60	120.1	776	-136.81	0.815	[0.155,1]	240.01	1.964	1285	392
7.20	115.3	889	-559.54	0.776	[0.101,1]	235.17	2.647	1274	374
14.40	110.0	837	-1344.66	0.737	[0.065,1]	232.42	2.951	1248	357
28.80	110.4	913	-2933.12	0.738	[0.085,1]	229.43	3.560	1251	356

Parameters: Power Cost = variable, VM Revenue = 0.17 euro, MaxRT = $2 \cdot RT_0$
Migration Penalty = 0.014 euro; [Energy = kWh; Profit = Euro; Time Spent = Seconds]

Table 6.6: Scheduling MILP Solver with different electric costs and migration penalty

Table 6.5 shows the results for the exhaustive solver and the other algorithms, seeing that the ordered BestFit obtains closer results to the exhaustive solver, that given small lower bound

gaps, we can assume it is extremely close to the optimal results. Further, table 6.6 shows how when the power cost increases, the model increases the number of migrations in order to reduce the energy consumption, also in consequence profit degrades.

6.6 Conclusions on Introducing Machine Learning on Data-Center Modeling

Nowadays optimizing the management of data-centers to make them efficient, not only in economic values but also in power consumption, requires the automation of several systems such as VM scheduling and resource management. And automation needs knowledge about the system to act in an "intelligent" way. As shown here machine learning can provide this knowledge and intelligence, as well as adaptivity.

Compared to previous works, we have shown here that it is possible to model VMs and system behaviors towards resources and SLA metrics in an automatic manner through machine learning, and apply them to full-scale datacenter models, letting schedulers more adjusted estimation functions a priori, to make better their decisions for each system and kind of VM.

Also, by focusing the data-center model towards factors to be optimized (economics, service quality to clients, and energy-related ones), a mathematical model can be built and solved by different methods. Allocation problems like the presented VM×host scheduling are NP-hard problems, so solving them with exact methods can be intractable given (not so) high sizes or problem dimensions. Experiments have shown that our model can find optimal solutions according to the introduced policies, but paying a high cost in computation time. For this reason, heuristics and approximate algorithms can be applied over the same model, obtaining, in the case of the Best Fit, solutions enough close to the optimal in extremely low time.

In next chapters we will include the study of scenarios where the CPU is not the only critical resource, so the ML methods have to deal with a more complex space of attributes, identifying different situations. The effects of the memory behavior of web server platforms will be studied to model and predict that factors affecting the memory occupation. Also the impact of input/output resources like network. Complementing the learned models revealing new hidden factors will make the model easier to handle and solve. Finally, new heuristics and approximate algorithms will be studied to solve the model without applying exhaustive solvers with high cost in time.

The works presented in this chapter have been published in the "IEEE International Conference on GRID Computing 2011" [25] (2011) an the UPC-LSI technical report *UPC-LSI-10-21-R* [24] (2010).

Chapter 7

Modeling DataCenter Resources using Machine Learning

As shown in the lately previous chapter, machine learning can be applied to discover information and improve DC management thanks to modeling and prediction. In this chapter we show the modeling of basic cloud resources (not only CPU, but also Memory and network IO), predicting resource requirements from context information like workload. Also we predict the QoS from resource planning to help scheduling algorithms make better decisions about job and resource allocation, aiming for a balance between throughput, quality of service, and power consumption. The approach is tested first with real data and web-services on a datacenter simulator, and further validated in a real execution on a reduced scale cluster running the OpenNebula [147] virtualization platform.

7.1 Introducing the DataCenter Scenario

In Chapter 6 we showed how the datacenter can be modeled using a mathematical program, including all the parameters affecting the system, all the internal variables that affect the output, the constraints that define the problem, and the output variables that provide the solution to our resource allocation problem. Our work in this chapter goes substantially beyond that, since then we considered only one resource (CPU usage) while here we model all the main resources on the datacenter. This makes the problem multidimensional, and it remained to be shown whether ML techniques can capture in a useful way the subtle interactions among resources. I.e, when a task runs out of memory and resorts to swapping, CPU consumption will dramatically decrease (for no apparent reason, if looking only at CPU consumption). If several tasks in the same physical machine compete for bandwidth, again each of it will slow down in CPU, etc.

In order to make decisions matching services and resources, as managers we may use low-level measurements (resource, power, and operating system monitors) and high-level data (user behavior and service performance such as up-time, response time and availability). Here we extend the methodology presented in previous chapters to, learn high-level information from low-level information and drive decision-making algorithms for virtualized job schedulers, without much expert knowledge or real-time supervision. All in all, the problem to solve is to decide the effects of resource allocations on hosts using consolidation towards QoS and power consumption.

In this chapter we focus on Intel Xeon HPC architectures, as a state of the art architectures looking for high performance, also we tested some non HPC architectures like Intel Celeron out of scene (results not shown here as not relevants for our current purposes), but with validating results as the obtained with the Xeon processors. In order to test the approach after validating

each models, we compare a set of ad-hoc and off-the-shelf algorithms for scheduling, with and without using predictive ML functions. Finally, we validate the algorithms and models on a real datacenter using the virtualization platform OpenNebula.

Related Work

As seen on Background Chapter 2, there are relevant previous works on autonomic computing and datacenter management using machine learning techniques. As many use Reinforcement Learning (e.g. [164][154][152][88]), to manage resource allocation and power consumption, others focus on status prediction using supervised learning or fuzzy logic (e.g. [57][10]). Almost all the current approaches, as far as we know, are oriented towards learning the consequences of using policies on determined states of the cloud or its elements. Here we focus on learning component behaviors as a way to improve the decision-making process in resource allocation and web-service job placement, instead of learning policies for components or global systems.

7.2 Infrastructure and Monitoring

In this chapter we focus on a set of techniques for modeling Cloud resources, having enough information to manage and schedule jobs on host machines in the most proper way. From each job (web-services in this case) we want to discover how much resources it will demand to the cloud, given the characteristics of its clients. Also we want to discover how giving more or less resources affects each job taking into account its load and machine hosting it, so we can decide how much resources we give to each one and where we allocate it. How services work on the cloud is explained in the following subsection.

7.2.1 Service DataCenter Architecture

As seen in chapter 2, in commercial datacenters, the web-service owners (datacenter customers) pay the resource providers on a usage-basis. The resource provider's goal is to maximize its profit, by fulfilling the QoS of each virtualized web-service and reduce the cost of running resources. For this, consolidation is a technique consisting on filling PMs with most VMs as possible and shutting down empty PMs, reducing energetic consumption.

Typical measures of quality of service on web sites are the average response time for web queries, the web site up-time and the ratio of satisfactory replies. The RT (focused as quality measure in this case of study) is affected directly by the amount of resources given to the VM and the amount of requests and clients received by the web-service. Also up-time can be considered as "acceptable RT" or "timed out request" from the client point of view. If a service receives insufficient resources than the required for handling a given load, the reply to the user web query will be slower; but over-granting resources to it will not imply a quicker reply.

The web-services this work focuses on are web applications, with a typical stack formed by the Operating System, the web server, the app environment, and the data-base server, like e.g. GNU/linux OS + apache server + PHP + MySQL. Each VM contains a replica system with this infrastructure, so customers add their services on the web server. Also we focus on the basic resources found in a datacenter (CPU, memory and input/output network).

7.2.2 Service Level Agreements

Quality of Service as perceived by the client is a highly complex issue, involving technological and psychological factors. Response Time, the amount of time required by our DCs to reply to a request, is a typical object in service level agreements, the contracts where providers and customers agree on a common notion of QoS. Here we make the simplifying assumption that SLA fulfillment is strictly a function of RT; further factors such as up-time rate could be added to our methodology, as long as they are measurable from monitored data. To be specific, here we measure the RT on the datacenter domain and network, not at the client side since he may use unpredictable thinking time and may have a slow machine or connection at their end.

We recall the common "RT to QoS function" in SLAs that we have used in previous chapters, setting a threshold α and a desired response time RT_0, and set SLA fulfillment level to

$$SLA(VM_i) = max(min(1 - \alpha\frac{RT - RT_0}{RT_0}, 1), 0)$$

that is, the SLA is fully satisfied up to response time RT_0, totally violated if it exceeds $\alpha \cdot RT_0$, and degrades linearly in between. We use this function for simplicity in our experiments, but this is a nonessential choice.

7.2.3 Information Monitoring

At scheduling time, we are interested on obtaining as much system information as possible, so knowing the requirements of each VM, availability of each PM, and RTs obtained from current loads and scheduling. Monitors get the load and resources from hosting machines and VMs, obtaining the following set of attributes per time unit: timestamps; number of requests; average response times; average requested bytes; and resource usage and bandwidth (BWD). Table 7.1 summarizes all these attributes.

time	Timestamp (timeunit) for current measure.
requests	Number of requests for the current time unit.
rtpr	Average response time per request for the current time unit.
bytespr	Average bytes per request for the current time unit.
cpuvm	Average CPU demanded by the VM for the current time unit.
memvm	Average memory demanded by the VM for the current time unit.
invm	Amount of input bandwidth used by the VM for the current time unit.
outvm	Amount of output bandwidth used by the VM for the current time unit.
cpupm	Average CPU occupied in the PM for the current time unit.
mempm	Average memory allocated in the PM for the current time unit.
inpm	Amount of input bandwidth used by the PM for the current time unit.
outpm	Amount of output bandwidth used by the PM for the current time unit.
rcpuvm	Average CPU given to the VM in the PM for the current time unit.
rmemvm	Average memory given to the VM in the PM for the current time unit.
cpugw	Average CPU occupied in the network gateway for the current time unit.
memgw	Average memory allocated in the network gateway for the current time unit.
ingw	Amount of input bandwidth used by the network gateway for the current time unit.
outgw	Amount of output bandwidth used by the network gateway for the current time unit.

Table 7.1: Attributes obtained from the monitoring agents

This information can be grouped in two classes: Load (#requests, response time per request, bytes per request), and resources (CPU, memory, and bandwidth, both at the PM and VM levels, and both requested and granted). Further, other useful information or attributes can be derived from the obtained from monitors, like the time elapsed from last time unit (previous measures and current measures); any load or difference of load respect the last time unit; any resource or difference of resources respect the last time unit; and any aggregation of resources on VMs, PMs or gateways.

7.2.4 Modeling and Prediction

When making decisions in this context, often the required information 1) is not available, 2) is available but highly uncertain, or 3) cannot be read because of privacy issues. Examples of the three cases occur when reading from both PMs and VMs, and information coming from VMs is extremely delicate to handle and interpret. First, observed resource usage often differs a lot if monitored at the PM level and at the VM level. VM apparent CPU usage is affected by its stress level and the virtualization software overhead. Secondly, monitors on the PM may read information of consumption relative to available resources, e.g. if a VM is running alone in a PM, consuming 25% effective CPU, but having CPU quota $\sim 100\%$, as the virtualization agent

has all the CPU for itself or it runs its own VM-inside IDLE process. And third, pinning the VM to read information from its internal system log could be against the customer privacy, as property of the data and code inside the VM.

In order to solve these lacks of information and uncertainty, we employ here machine learning methods, basing the work on the hypothesis presented in Section 2.3.2, where we understand that for each situation, an expert can carefully obtain a model or tuning better that any ML-learned model, but ML can obtain semi-automatically a model as good or better that a generic model built without intensive expert knowledge. The advantage of ML over explicit expert modeling is when systems are complex enough that no human expert can explore all relevant possibilities, when no experts exist, or when system changes over time so models must be rebuilt periodically or reactively to changes. This is the main reason we use these ML modeling and prediction techniques, to obtain accurate models of the datacenter we are managing, and predict information and behaviors to manage it properly.

7.2.5 Framework Schema

The first contribution of this chapter is to model the VM and PM behaviors (CPU, Memory and IO) from the amount of load received, to be predicted on-line, boosting the decision making algorithm (here the PM×VM scheduler) with extra information. By learning a model of the function $f(load) \rightarrow \mathrm{E}[CPU, MEM, IO]$, lectures from inside the VM can be replaced, and predict the estimated effective resources required by a VM depending only on its received load without interferences of stress on the VM or occupation on the PM or network.

The second contribution is the prediction of the QoS variables (this is, the RT). When the scheduler has to make decisions on where to place each VM depending on the expected resource usage, minimization of power consumption will consolidate VMs in PMs. Giving each VM always the maximum resources would not consolidate resources as much as could be, and giving each VM less than the minimum required given the load would degrade the RT. By learning a function expecting the RT from placing a VM in a PM with a given occupation $f(status, resources) \rightarrow \mathrm{E}[RT]$, scheduler can consolidate VMs without risking the RT in excess, and grant resources playing safe.

As seen in previous Figure 4.2, showing the information flow and elements composing our decision making schema, we substitute all the human-expert models by the semi-automatically learned ones. From past monitoring information we obtain a model for load versus CPU, MEM and I/O consumption for each VM/web-service (or each kind of). Also we obtain a model for load and context vs RT. Then the decision maker, powered by current on-line monitoring information plus model predictions, creates the next schedule for VMs, using some classic fitting function like "always give maximum resources" or using the RT prediction by playing with tentative resource grants.

7.3 Resource Modeling and Learning

The following subsections explain each modeling and Figure 8.1 shows the numeric results for each one. Note that to obtain accurate data from the resource usage all measures are taken on a VM running alone in a test run in *void*, without other VMs or jobs in the PM. The learned models must include only the VM behavior with the virtualization overheads.

We used the machine learning methodology and algorithms seen in previous chapters and explained in the Background chapter, for all our learning tasks. We tested different regression methods (LinReg, M5P, k-NN, RepTree, ...) for each model, with tuned parameters, selecting the simplest model among the ones that bring the best or almost-best results.

Further, the experiments to test this approach have been performed obtaining data from real workloads applied to real hosting machines, and then using this information in a simulated a datacenter in order to check the scalability of the method. All the following experiments and data about behaviors and response times, have been obtained from a full recreation of the LiBCN'10 workload [26] on the datacenter provided by RDLab-UPC [139], with Intel Xeon 4 Core machines running at 3Ghz and with 16Gb RAM, running jobs of kind [Apache v2 + PHP + MySQL v5] in a virtualized environment [Ubuntu Linux 10.10 Server Edition + VirtualBox v3.1.8]. The

workload have been properly scaled on some experiments according to the workload instructions in order to recreate different load volumes.

7.3.1 CPU Prediction

CPU is the main element in resource brokerage. To serve incoming requests, the VM running a web-service will demand CPU depending mainly on the amount of requests. From the monitors we can obtain, for example, the current number of requests per time unit $E[requests]$, the average time per request $E[timepr]$ (in a no-stress situation), and the average number of bytes exchanged per request, $E[bytespr]$. The function to be learned, one for each type of PM in the datacenter, maps $E[requests]$, $E[bytespr]$, and $E[timepr]$ to some figure $E[cpuvm]$ denoting the expected percentage of the PM CPU that will be used by the VM in these circumstances. Other predictive variables can be added to the function, but these are the ones we found significant in our specific experiments.

The expected number of requests $E[requests]$ can be obtained by observing the number of requests of the last time units, or knowing a priori the usual amount of requests given the week day and time hour, among other techniques; the expected bytes per request $E[bytespr]$ can be obtained or modeled by having the averages of bytes per request; and the CPU time per request $E[timepr]$ can be obtained also by measuring the average CPU time spent per request on a VM running alone in a test run without interferences of other VMs or jobs. All the other variables have been dismissed as they will not be available at the time of using this model, they have been considered independent to this specific problem, or they showed very low relevance.

By modeling the amount of CPU given the load (without other VMs or jobs in the Physical Machine) we obtain the natural CPU demand function of the VM and its contained web-services. This model will also include the CPU overhead provided by the virtualization platform. This predicted information will be necessary to learn the same behavior under stress, where VMs will compete for the same CPU resources of the PM. The function to be learned is:

$$f_{cpuvm}^{learned}(E[requests], E[bytespr], E[timepr]) \rightarrow E[cpuvm]$$

In order to learn this function, we used a M5P algorithm [76]. The relation between requests, bytes per requests and time per requests seemed to be non-linear, as two of their parameters are averages laying over the third. But the algorithm M5P is able to approximate this non-linear relation by parts. Figure 7.1 shows the model selection test and the validation test, also Table 7.2 shows the details of the CPU training result:

	Learning Results
Method	M5P ($M \in [50, 70]$)
Training Dataset	3968 instances
Model Selection	Random split (66%,34%)
Validation Dataset	7528 Instances
Error / StDev	MAE = 2.53% CPU; $\sigma = 4.511$
Data Range	[2.37,100.0] % CPU

Table 7.2: Learning Load vs CPU function

The tree of regressions obtained shows that *requests* and *timepr* are the most relevant variables by driving the tree. It explains how requests and the CPU time per requests increase the amount of CPU required by the VM, something we expected and very reasonable.

7.3.2 Memory Modeling

The second resource to model is the Memory allocated to the VM. Unlike other resources, memory consumption has... memory, that is, the memory used at any given moment cannot be determined or even approximated from the currently observable measures, but strongly depends on the past evolution. This is because because virtualization software, operating systems, the

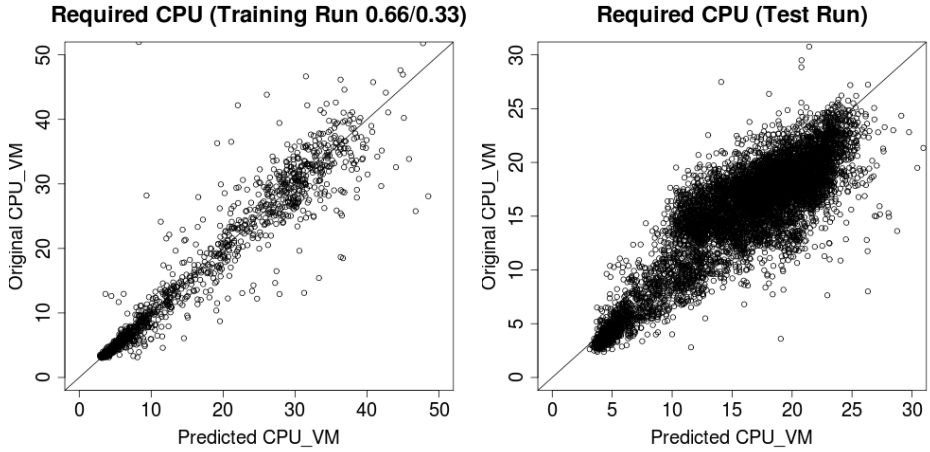


Figure 7.1: Prediction of VM CPU demand for Xeon-4Core

Java Virtual Machine, application servers, and databases typically will initially request large chunks of memory for buffering and caching, which is retained until some limits are reached, when memory space is reorganized by flushing caches, freeing unused elements, and general garbage collection (Figure 7.2 shows the memory reorganization on a apache web server given different amounts of memory limit.). These changes are known to be nightmarishly difficult to model explicitly. At this stage, we propose to learn to predict the amount of used memory $memvm_t$ used at the t-th measurement as a function of $memvm_{t-1}$ and the variable describing the load, say $E[requests]$, $E[bytespr]$, $E[timepr]$.

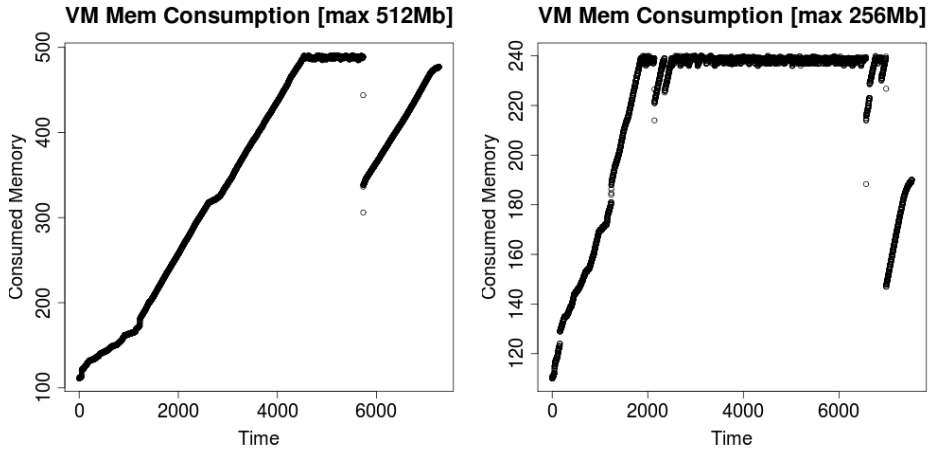


Figure 7.2: Typical VM Memory Behavior, with memory Flushing and Garbage Collection

Knowing how web servers usually work, each new client (web user) makes the web server and application server to allocate memory for his transactions, and also caches grow with each new request until reaching the memory limit when memory allocated for old clients and unused cached elements is freed. In order to predict memory we should know how much memory we had previously allocated for the VM, and the expected amount of load incoming. So it can be learned as a function of the elements representing load ($E[requests]$, $E[bytespr]$, $E[timepr]$, $E[mbps]$, ...) and the previous memory status ($memvm_{t-1}$) and the time t of last memory measure. Being $memvm_{t-1}$ the amount of memory used at the previous observation, and ΔT the amount of time elapsed since then, the function to learn at this moment is:

$$f_{mem}^{learned}(E[requests], E[bytespr], E[timepr], E[mbps], memvm_{t-1}, \Delta T) \rightarrow E[memvm]$$

First experiments trying to find a function that fits the sample data, using regression methods like LinReg or M5Ps showed models incompatible between different time intervals ΔT, as each model fitted the average of ΔT, and new measures with extreme time intervals missed the target, and M5P tended to create a branch for each seen ΔT. After some observations and conclusions, we discovered that the relation of all our input data is not linear, as $memvm_{t-1}$ and ΔT followed a polynomial relation, as the weight of $memvm_{t-1}$ could vary depending on the age of the measure ΔT.

In order to confirm this suspicion, adding a new attribute $memvm_{t-1} \cdot \Delta T$ on the datasets, linear regression method fits adequately independently of the ΔT factor and correcting previous experiments without this attribute. Figure 7.3 shows the result of applying a Linear Regression as a memory learner, also the details of the training and testing are shown in table 7.3.

	Learning Results
Method	Linear Regression
Training Dataset	108 instances
Model Selection	Random split (66%,34%)
Validation Dataset	243 Instances
Error / StDev	MAE = 4.3963 MBytes; σ = 8.3404
Data Range	[124.2,488.4] MBytes

Table 7.3: Learning Load vs MEM function

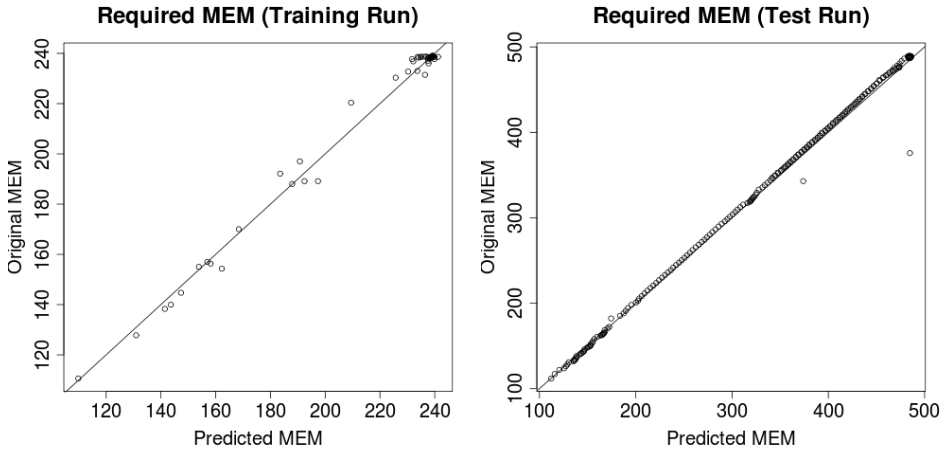


Figure 7.3: Prediction of MEM VM demand (TR: $\Delta T \in [10s, 10min, 1h]$; TS: $\Delta T = 5min$)

An interesting thing is that the function obtained in the regression shows that the expected VM Memory is almost always the previous memory value plus an increase depending on the load:

$$E[memvm_t] = \alpha \cdot memvm_{t-1} + (1 - \alpha) \cdot f(Load_t, \Delta T, memvm_{t-1})$$

with ΔT being the real time between measurements t and $t-1$, and α about 0.9 in our experiments. We observed that linear regression of this form gave reasonable results. Obviously, more complex models with nonlinear dependencies and longer memory are up for research.

7.3.3 Bandwidth Prediction

The final resource we model is the network bandwidth used by each VM. VM's in a PM usually share the network interface, which implies that all VMs dump and receive data directly from the same physical interface. PM traffic is then the sum of all VMs network packets plus some extra PM network control packets. This traffic also includes the data transfer from external file system servers, something common in commercial DCs, so the usage of disk requires network.

Knowing the volume of data the host will transmit up and down over the network is also an important issue to take into account. Low load imply the network is usable and VMs can deliver the responses to requests at the moment, and high loads can saturate networks and requests or responses can not be delivered to the VMs. We then would like to learn a function returning the expected number of incoming and out-coming packets at a PM as a function of the sums of load parameters $E[requests]$, $E[bytespr]$, $E[timepr]$ of the VM's allocated to it. The function to be learned, using the appropriate time aggregation of data, is:

$$f_{bwd}^{learned}(E[requests], E[bytespr], E[timepr]) \rightarrow \langle E[\#PktIn], E[\#PktOut] \rangle$$

After experimenting with several different models (LinReg, decision trees, ...), we again selected the tree-of-models M5P algorithm because of its results. Figure 7.4 shows the model selection test and the validation test, also the details of the M5P usage are shown in table 7.4.

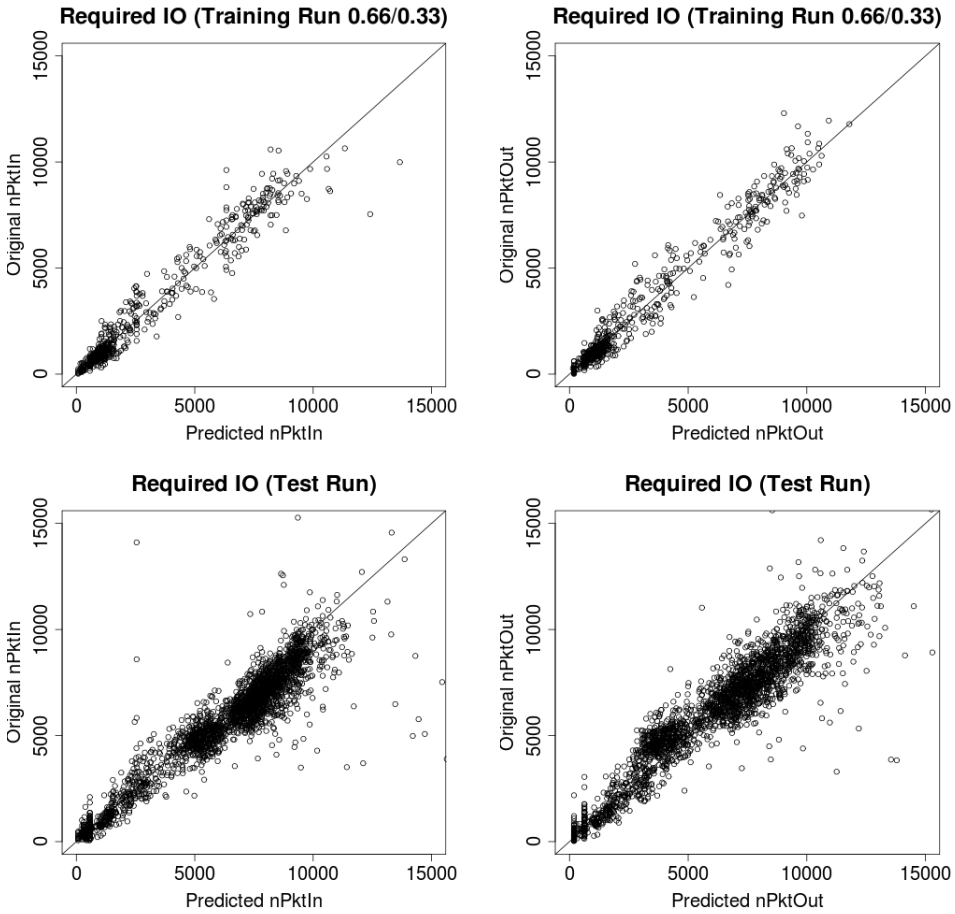


Figure 7.4: Prediction of PM Bandwidth demand

	Learning Results	
Method	M5P ($M \in [10, 15]$)	
Training Dataset	1623 instances	
Model Selection	Random split (66%,34%)	
Validation Dataset	2423 Instances	
Error / StDev	IN: MAE = 926 Pkts; σ = 1726	OUT: MAE = 893 Pkts; σ = 1807
Data Range	IN: [56,31190] #Packets	OUT: [25,41410] #Packets

Table 7.4: Learning Load vs IO function

7.3.4 SLA Prediction

Finally, we need to predict the expected SLA fulfillment levels given a placement of VM's in PM's and resource allocation within each PM. Our decision making method (allocator) is based on a fitting function that predicts the degree SLA fulfillment of a VM (a figure within 0 and 1) from its load parameters and its context, i.e., the features of the PM where it is currently or tentatively placed, the load parameters of the VM in the same PM, and the amount of physical resources currently allocated to each VM.

$$f_{RT}^{learned}(Load, VM\_Required, VM\_Granted) \rightarrow E[RT]$$

As explained, we made the simplifying assumption for the moment that SLA fulfillment depends exclusively on response time, RT, via a known and simple function. The task is thus to learn a function relating a high-level measure, response time, to low-level measures, such as number of requests, bytes per request, CPU, memory, and bandwidth used by each VM. Most importantly, we are predicting the response time *if we (hypothetically)* placed VMs in a particular candidate way that the scheduling algorithm is currently considering among others. Therefore, we do not really have most low level measures: we will instead use the predictions by the respective learned models discussed before.

For this prediction stage, we use again the M5P method, since simple linear regressions were incapable of representing the relations between resources and RT. The details of the M5P usage are shown in Table 7.5, and prediction results are shown in Figure 7.5.

	Learning Results
Method	M5P ($M \in [10, 15]$)
Training Dataset	38040 instances
Model Selection	Random split (66%,34%)
Validation Dataset	15216 Instances
Error / StDev	MAE = 9.9ms; σ = 35.4ms
Data Range	$[0, 2.78]$s, $\widehat{RT}$ 17ms

Table 7.5: Learning Load,Resources vs RT function

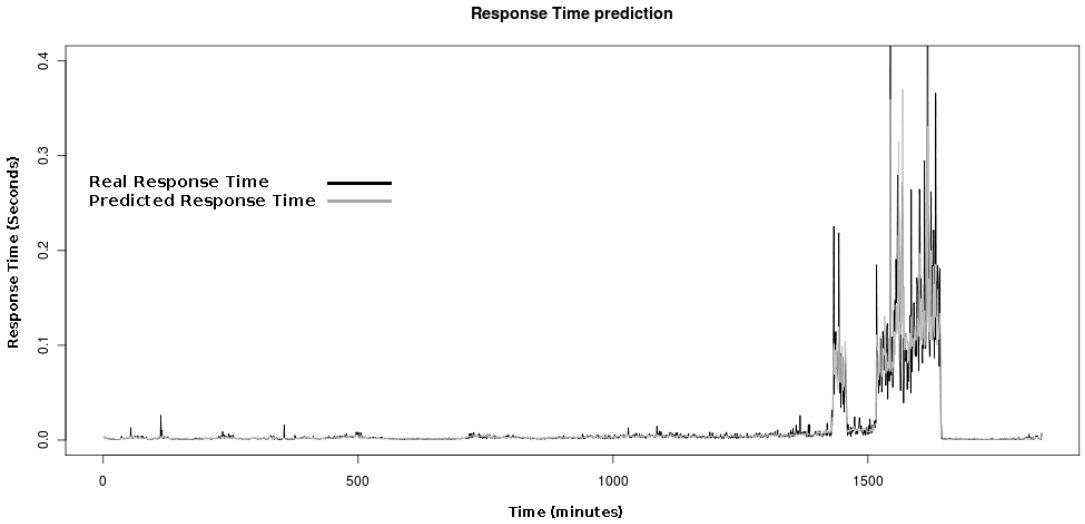


Figure 7.5: Prediction of Response Time, in a non-stress and stress situation. Stress begins around instance 1400, where RT begins to increase

7.4 Managing and Scheduling DataCenters

7.4.1 Scheduling Algorithms

Following the schema explained in previous chapters, the datacenter benefit optimization problem can be formulated as a Mixed Integer Program, which can be made linear if the elements in it are themselves linear. In short, the function to be maximized is the sum of:

- the income from customers from executed jobs,
- minus the penalties paid for SLA violation, typically a function of the SLA fulfillment level as described above,
- and minus the power costs, which we can take as the sum of power consumed by all machines during the times are turned on, times the cost of a power

The function reflects the trade-off we have been discussing so far: one would like to have as many machines turned on as possible in order to run as many customer jobs as possible without violating any SLA, but at the same time to do this with as few machines as possible to reduce power costs. The unknowns of the program describe which tasks are allocated to each PM, and how resources of each PM machine are split up among the tasks allocated to it. Constraints in the program link these variables with the high level values (degree of SLA fulfillment, power consumption). The point of our methodology is that the functions linking the former to the latter are, in many cases, learned via ML rather than statically decided when writing up the program as explained in previous Chapter 6 and [25]. Here we expand the previous introduced machine learning elements considering several resources instead of CPU only.

Such Mixed Integer Programs can be in theory solved exactly via exhaustive solvers, but due to its exponential cost in the number of variables and constraints, this becomes unfeasible for realistic settings. In previous Chapter 6 we saw that approximate algorithms and heuristics can obtain results close to an exhaustive search solution. Thus we use here approximate or heuristic, faster algorithms: the generic for bin packing problems, Ordered First-Fit and Best-Fit algorithms [163] (as seen in previous chapters as Algorithms 6,7). We also use two the BackFilling and the λ-Round Robin algorithms (as seen in previous chapters as Algorithms 4,5), specialized for load-balancing via consolidation.

All such algorithms use as an oracle some "fitting function" used to evaluate how well a VM "will fit" into a PM which has already been assigned some VMs. We propose to substitute the conventional fitting functions by the learned function mapping tasks descriptions and assigned functions to response times. This predicted time in turn is monetized, via the SLA fulfillment function, into a predicted SLA economic penalty, hence into its effect on the function to be maximized.

7.4.2 Environment Description

We have performed different test to demonstrate how ML can match or improve approximate and ad-hoc algorithms using explicit knowledge, and to validate the models on real machines. The experiments have been performed using real workloads for the model learning process, an analytic simulator to compare the different ML-augmented algorithms, and real hosting machines for the model validation.

The testbed used for experimentation is similar to the seen previously in Chapters 6 and 5, but here we do not only take models from real systems but also use them to validate the whole approach. As a testbed we used the previously described HPC machine environment (Intel Xeon 4Core), first to tune the R version of the previously seen cloud simulator EEFSIM to test the ML-augmented algorithms; and then we deployed our OpenNebula + Scheduling middleware over the set of 10 Xeon 4Core machines, as explained previously. The simulator allowed us to test different configurations and models before validating the model on a real datacenter, also recreate big scenarios easily. The workload used corresponds to the Li-BCN Workload 2010 [26].

To price each element involved on our minimization function, we established that providers behave as a cloud provider similar to Amazon EC2, where users will rent some VMs in order

to run their tasks. The pricing system for the VMs is similar to the one EC2 uses and medium instances with high CPU load are assumed. We fixed their cost to 0.17 euro/hour (current EC2 pricing in Europe), and the power cost to 0.09 euro/KWh (representative of prices with most cloud-providing companies).

As a parameter defining the QoS, we used the response time at the datacenter exit gateway. The jobs on workload have as RT_0 the values $\in [0.4, 1.2]$s (each job can have different SLA terms), as experiments on our datacenter showed that it is a reasonable response value obtained by the web service without stress or interferences. The initial α parameter is set to 1 (SLA fulfillment is 0 if $RT \geq 2RT_0$).

7.4.3 ML-augmented scheduling algorithms

In the following experiments we compare the λ-RR and BackFilling algorithms, also First Fit and Best Fit algorithms, these two last ones with and without ML added functions. Their basic versions require expert information on the models and the fitting function. E.g. checking if a VM fits in a PM requires getting the VM CPU usage and add at least +20% of virtualization overhead, while the ML version uses the VM CPU as a parameter of the RT prediction function without pre-known factors. The same happens with the memory, as instead of multiplying the VM memory per 2 as memory caching overhead, the predictor uses this value without extra knowledge. Further, each version with machine learning uses the learned function RT as a fitting function $E[RT_{vm}] \geq RT_{0,vm}$ or profit function $SLA(E[RT_{vm}], RT_{0,vm})$, while the others use as fitting function $cpupm_h + cpuvm_{vm} \leq MaxCPU_h$ and $mempm_h + memvm_{vm} \leq MaxMEM_h$.

	Benefit (euro)	Energy (wh)	Avg.QoS	Migrations	Avg.PMs/h
λ-RoundRobin	33.94	2114	0.6671	33	9.416
BackFilling	31.32	1032	0.6631	369	6.541
First-Fit	28.77	1874	0.5966	139	6.542
First-Fit + ML	29.99	1414	0.6032	153	5.000
Best-Fit	29.85	778	0.5695	119	2.625
Best-Fit + ML	31.771	1442	0.6510	218	4.625

Table 7.6: Comparative of algorithms from the relevant business model values

These comparative experiments are simulator-based, running 20 VMs containing web-services in a datacenter of 20 machines with 2 or 4 cores, for a 24 hours workload, and scheduling rounds of 1 hour. Data monitoring and statistics are taken each 10 minutes, and the measures used to perform each schedule are the ones taken previously to the scheduling round flank. Table 7.6 shows the results.

From the results we observe that the versions using the learned model perform similar or better than the versions including expert knowledge, and they approach relatively well to the ad-hoc expert algorithms, backfilling and λ-RR, using the optimal configurations for this kind of datacenter, calculated in [70]. While ML version of the approximated algorithms are better than their expert-knowledge versions, the Best Fit + ML approach is close to the ad-hoc expert algorithms in QoS and benefit.

7.4.4 Validation on Real Machines

After the initial experimental check on the simulated datacenter, we move to validating and testing the method in a real environment. The set-up consists in a small workbench composed by 5 Intel Xeon 4core machines, 3 as datacenter nodes, 1 as gateway and 1 attacking machine reproducing client requests, in a different datacenter than the previous training experiments. The virtualization environment is the same as in ML training and testing (Oracle VirtualBox), and as virtualization middleware framework we use OpenNebula, replacing the default scheduler by our own, having implemented on it our policies and algorithms. We introduce on the system 10 VMs, each one containing a replica of the LiBCN10 imageboard website. Also for each VM, an attacker is launched replicating the load of a whole day scaled by 100-300 times to reproduce

heavy load (using different days and scaled different for each VM in order to create some diversity on traffic). We ensured that all of CPU, memory, and bandwidth overload occur separately and in combination in our benchmarks, to test the full spectrum of prediction models.

We used physical machines with the same architecture than those used for the training, so that we could import the learned models for CPU, memory, and I/O. The Response Time model had to be learned again, as the network environment and topology were different and response time certainly depends on them. We observed that linear regression, in this case, seemed to perform significantly worse than before. We trained a nearest neighbor model, which recovered the previous performance. Let us recall that the contribution we want to emphasize is not the particular models but the methodology: this episode suggests that, methodologically, it is probably a good idea to fix on any particular model kind, and that upon a new environment or system changes, several model kinds should be always tested.

For this validation experiments we run Best-Fit against its ML-augmented version in this reduced environment. Figure 7.6 presents the results. We can see that best-fit considers that all VMs will fit in CPU and Memory (virtualized and physically) in one machine, which degrades RT. The ML approach, instead, is able to detect from low-level measures situations where RT would not be achieved (because of CPU competition, but also because of memory exhaustion and network/disk competition), hence migrating sufficient VMs to other machines where, for example, network interfaces not so loaded.

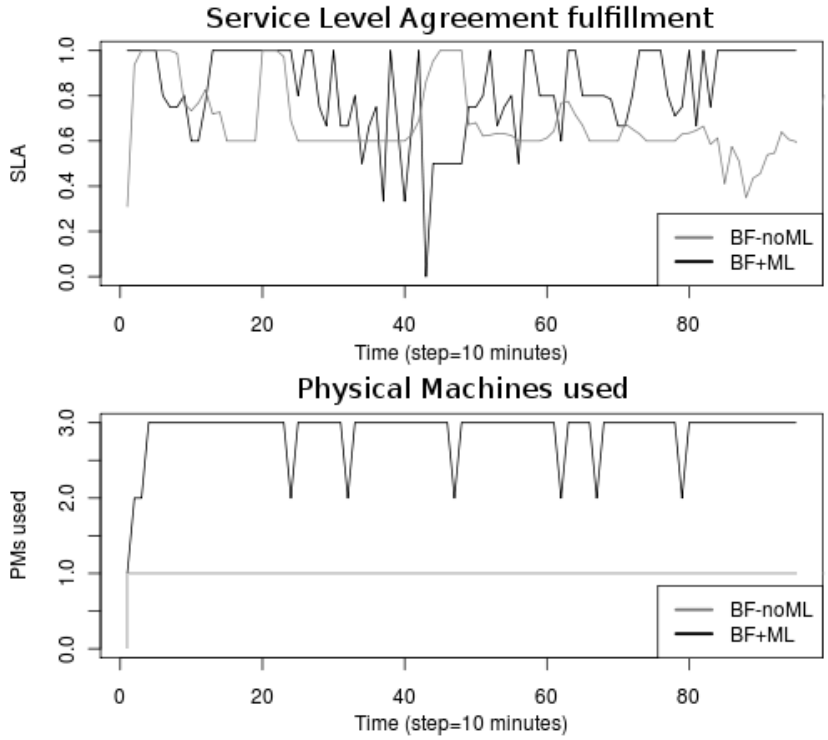


Figure 7.6: BF-noML against BF+ML SLA (based on response time) and machines used

We observe that the ML-augmented versions of Best-fit and First-fit improve the final figure (revenue) over their non-ML counterpart, and gets closer to the revenue of ad-hoc algorithms such as λ-RR, which needs to be parametrized and tuned by the administrators. Not reported here is the fact that these ML-augmented versions can automatically adapt to changes in task execution prices, SLA penalties, and power price as shown on [25]. Adapting the ad-hoc algorithms to these changes requires expert (human) intervention, and is simply unfeasible in the highly changing scenarios envisioned for the future, where virtual resources, SLA penalties, and power prices will interactively and constantly be in negotiation, for example by means of auctions and automatic agents. In fact, the ML hypothesis can be verified, as for any *fixed* parametrization of λ-RR, there will be some scenario where a real-time, on-line learned ML-model will perform better.

7.5 Conclusions on Modeling DataCenter Resources using Machine Learning

In this chapter we presented a methodology for modeling cloud computing resources of a web-service based datacenter using machine learning, obtaining good predictors to empower and drive decision-making algorithms for virtualized job schedulers, without the intervention of much expert knowledge. Using these models, CPU, memory, input/output and web-service response times can be predicted, so classic scheduling generic algorithms such as First-fit and Best-fit can use predictions to make more accurate decisions driven by goal functions (SLA depending on response times). This is a more economical and sustainable solution than resource over-provisioning, which is still the predominant one in practice.

Models and schedulers have been trained in a real datacenter using as input real web-service traces, tested in a simulation environment to compare behaviors in large scale datacenters, and latter tested in short scale in a real cluster, using the OpenNebula virtualization platform as testbed. We observe that the ML-augmented generic algorithms behave often equal or better than ad-hoc with expert tuning. Response time and quality of service is better maintained on some stress situations when it is possible, by consolidating and de-consolidating by predicting the required computing resources and the resulting RT for a given schedule.

Further, a reason to obtain learned models, far away from applying them into autonomic systems, is that it has been useful to examine them. From the models we can observe the patterns for a resource or the whole system, relevant system variables, characteristics of specific situations. All of this using modeling methods that provide readable model, able to be easily understood by system operators, resource designers and datacenter architects, so the human personnel can also learn from the system and improve it thanks to machine learning.

The current experiments are performed using Intel Xeon architectures, focusing on high performance. An effect seen on these experiments, is that the resource that easily saturates and becomes bottleneck is network, as checked on machines during experiments and also seen on the created models, where IO attributes become most important. CPU and memory became here hard-to-deploy resources, and although putting several virtual machines in a single physical one did not fulfill CPU and memory, response time degraded due to competition for network usage. As commented in the chapter introduction, before choosing Xeon we performed some experiments (not reported) with Intel Celeron processors, and saw the same effect but with memory, as the usage of memory became bottleneck as seen in monitors and learned models.

In the next chapter we will focus on scalability and on hierarchically modeling the cloud system as a set of datacenters where services can not only move between machines but among locations around the world. We will introduce DC to DC network elements, like the service time DC-client as another SLA object, and forcing the services to go near their demand. Further we will turn to low-powered architectures such as Intel Atom processors, where in contrast with Xeon, CPU may become the bottleneck and models will learn about it.

The work presented in this chapter has been published in the “ACM Symposium on Applied Computing 2013” [27] (2013).

Chapter 8

Extending to Multi-DataCenter Systems

The Cloud is not only a collection of individual datacenters, but a collaborating multi-datacenter infrastructure distributed along the world. Continuing from previous chapters, we need to expand the consolidation strategy from a single DCs to a full network of DCs, taking advantage of local energy prices, and assuring quality of service and proximity to clients and customers. (De)consolidation and priming proximity to clients become two main strategies to allocate resources and properly place these web-services in the multi-DC network. In this chapter we expand the previously mathematical model, including multi-DC and VM transportation variables.

8.1 Introducing the Multi-DataCenter Management using Machine Learning

Proximity of resource datacenters to datacenter customers and clients is usually translated into quality of service, as response times are an important Service Level Object (SLO) as part of service level agreements. So service placement in a spread DC network is extremely important when assuring a minimal quality of service. Naturally, providers want in turn to optimize the use of the resources they have deployed with their own metrics. Because the volume, heterogeneity, and complexity to be managed, this has become today a hard optimization problem. It is even harder for a typically provider who owns a multi-DC system, usually distributed through the world, and must balance response times, data and task location, and energy consumption. All of this management and access tasks being with total transparent view of the infrastructure to their customers.

In this chapter we model the multi-DC system as a mathematical optimization problem, to schedule virtualized web-services inside DCs and across DC networks, using energy consumption, resource allocation and QoS as decisive factors. As seen in previous chapters we use machine learning to predict the effect of VM placement moves. These techniques for local DC management previously studied, are now scaled to multi-DC systems having into account the new relevant factors like proximity service-client, migration overheads, and modularity between inter-DC relations and information. The main contribution here is thus the hierarchical extension of those techniques to the multi-DC scenario, where latencies and energy prices vary among DCs.

The problem is to decide the best placement for each VM depending on the energy costs, request processing time, network latencies, and delays between clients and VMs, the virtualization and VM migration overheads, and resource allocation to each VM, while consolidating and not degrading the QoS. To test the approach we prepared ML models of CPU, memory, bandwidth, response times and SLA fulfillments from a real system and workloads. We compare a previously studied method based on the best-fit heuristics algorithm for scheduling in previous chapters,

with and without using predictive ML functions to test each prediction and the benefits of using learned models. And finally we study the performance of our formulation using energy, latencies, and QoS factors on a real DC environment using the OpenNebula [147] virtualization platform.

Related Work

Load distribution among datacenter networks is also an important aspect to be dealt with. Virtualization technology allows an easy management and migration of jobs among hosting machines and DCs [108], and orchestrating this management on DCs and multi-DC networks is currently a challenging topic (see [73][16]). Works like [176] present within the Sandpipe framework strategies and key points to migrate virtualized jobs mitigating hotspots on hosting systems. Also other works focus explicitly on balancing load by following renewable energies (as seen in previous sections on this chapter), like [100] and [71], where optimization focus on move load where renewable energy is at each moment. Here we apply virtualization on our system to migrate the load across a worldwide distributed multi-DC network, including a policy optimizing DC-user proximity versus migration costs vs energy consumption, but letting room to introduce other policies like "follow the sun/wind" into the energy cost computation.

8.2 The Multi-DataCenter Scenario

Business Model for Multi-DCs

Companies offering computational power or web hosting (e.g. Amazon [8]) base their business on offering customers resources from their multi-DC system for running web-services (often virtualized). Customers pay the provider according an SLA, generally focused on QoS objects toward the web-service clients. Usually this QoS depends on the amount of resources (CPU, Memory, IO...) granted to each VM (hence to the web-service), but these resources have a cost in running them (energy, maintenance, cooling, ...). The provider goal is to ensure the agreed QoS for the VMs, while minimizing the costs by reducing the resource usage.

A typical QoS measure on web sites is response time, and this response time is affected by the time to process a client request and dispatch it, also the proximity of the service to the client. The time to process and dispatch is affected by the resources dedicated to the VM and the load towards the web-service (number of requests competing for it) at that same time. A VM receiving insufficient resources will be slower to reply to user requests, but over-granting resources past a certain point will not necessarily speed-up the replies. Proximity to the client depends on the localization of the client requesting the web-service, the placement of the required VM holding the web-service, and the connection between the DC with the client. Here we consider that client requests going to non-local DCs can pass through our inter-DC network, while the client is connected through his/her local DC [7].

Finally, thanks to VM migration, web-services can be moved following the best benefit. When the benefit function involves green energy (with really low cost or none), involves the usage of the web-service in regular office time schedule, or a factor that moves across the world on a given pattern (i.e. daylight), the resulting strategies are the kind of "follow the X" strategy. On green energy aware systems, VMs follow the sun, or the solar and wind production, minimizing the usage of "brown" energy. Other systems, like ours, use a "follow the load" policy according to clients requests, moving VMs next to clients to provide good response times, but balancing it with the costs of energy and process QoS.

Collecting Information

In cloud-like architectures oriented to multi-DC infrastructures, middlewares are in charge of managing VMs and PMs in an autonomic way, following policies and techniques based in the MAPE schema [43]. We rely on such middlewares both for collecting high- and low-level data (monitoring), and for managing VMs and PM resources (executing). The typical middleware schema shown in previous chapters (see again Figure 2.2) can be expanded to multi-DC infrastructures, where the physical components can be distributed among different s.

When we are scheduling a VM in a given DC we are interested in each VM requirements and each PM resource availabilities, also to be aware of current loads to each VM and resulting RTs. We can monitor the load to each VM by measuring the number of requests, the average response time per request and the average bytes per request, also how much CPU, Memory and Bandwidth is used in each PM, and how those are being shared among the VMs. Using machine learning methods we want to 1) anticipate the VM requirements given an expected incoming load, 2) reduce overhead of PM monitors, when observations can be replaced by estimations, and 3) predict an expected RT and QoS given tentative placements, making better scheduling decisions to maximize QoS.

Handling all this information becomes difficult the larger, more distributed, and loaded the system becomes. For this reason multi-DC systems management tend to decentralize, allowing each DC to administer their PMs and VMs, transferring VMs across DCs only when required. Here we propose allowing each DC to deal with its VMs and resources (as shown on Chapter 7), bringing to the global scheduler information about the offered or tentative host where each VM may be placed for each DC. This modularity lets the global scheduling to drive the multi-DC by load sources, energy costs, and also predicting QoS using the provided host information for such DC; and after locating the VM into a DC, the local DC will decide if it reallocates properly the VM inside it or (de)consolidates intra-DC.

Service Level Agreements

As a service level agreement we use the response time function, previously explained in Chapter 7. This time, when computing RT, we should differentiate the time to process the request and produce a reply; and the time to transmit the reply to the client (a VM in a DC near the client will bring lower latency than in a farer DC). The resulting RT is the addition of the time to produce RTp and the time to transmit RTt.

8.3 Mathematical Approach and Models

The following mathematical model represents the system to be optimized:

Maximize:

$$\begin{aligned} Profit &= \sum^{i\in \mathrm{VM}} f_{revenue}(SLA[i]) - \sum^{i\in \mathrm{VM}} f_{penalty}(Migr[i], Migl[i], ImgSize[i]) \\ &- \sum^{h\in \mathrm{PM}} f_{energycost}(Energy[h]) \end{aligned}$$

Output:

$Schedule[PM, VM]$, Integer Binary ; the Schedule

Parameters:

$Resources$[PM], resources $\langle CPU, MEM, BWD\rangle$ per host

$Load$[VM, $Locs$], requests, bytes/req, ... per VM and source

$pSched$[PM, VM], previous schedule

$LatHL$[PM, $Locs$], latency between hosts and sources

$LatHH$[PM, PM], latency between two hosts

$ImgSize$[VM], size for the image for current VM

RT_{0i} and α_i, RT_0 and α for VM i to fully satisfy its SLA

Constraints:

1. $\forall i \in \text{VM} : \sum^{h \in \text{PM}} Schedule[h,i] = 1$
2. $\forall h \in \text{PM} : \sum^{i \in \text{VM}} GivenRes[i] \cdot Schedule[h,i] \leq Resources[h]$
3. $\forall h \in \text{PM} : Energy[h] = f_{Energy}(\sum^{i \in Schedule[h,*]} GivenRes[i])$
4.1. $\forall i \in \text{VM} : Migr[i] = \lceil \sum^{h \in \text{PM}} (Schedule[h,i] \oplus pSched[h,i]) \rceil^1$
4.2. $\forall i \in \text{VM} : Migl[i] = \sum^{h_1,h_2 \in \text{PM}^2} Schedule[h_1,i] \cdot pSched[h_2,i] \cdot LatHH[h_1,h_2]$
5.1. $\forall i \in \text{VM} : ReqRes[i] = f_{ReqRes}(VM_i, Load[i,*])$
5.2. $\forall i \in \text{VM} : GivenRes[i] = f_{Occup}(ReqRes[i], Schedule[i,h])$
6.1. $\forall i \in \text{VM} : RT_p[i] = f_{RT}(Load[i,*], ReqRes[i], GivRes[i])$
6.2. $\forall \langle i,l \rangle \in \langle \text{VM}, L \rangle : RT_t[i,l] = \sum^{h \in \text{PM}} LatHL[h,l] \cdot Schedule[h,i]$
6.3. $\forall \langle i,l \rangle \in \langle \text{VM}, L \rangle : RT[i,l] = RT_p[i] + RT_t[i,l]$
7. $\forall i \in \text{VM} : SLA[i] = f_{SLA}(RT[i,*], RT_{0i}, \alpha_i)$

Goal Function: In short, the function to be maximized is the sum of:

- income from customers for executed VMs according to the SLA
- minus the penalties paid for SLA violation when migrating
- minus the energy costs, as the sum of energy consumed by all on-line machines

The function reflects the trade-off we have been discussing so far: one would like to have as many machines turned on as possible in order to run as many customer jobs as possible without violating any SLA, but at the same time to do this with as few machines as possible to reduce power costs. The unknowns of the program describe which tasks are allocated to each PM, and how resources of each PM machine are split up among the tasks allocated to it. Constraints in the program link these variables with the high level values (degree of SLA fulfillment, power consumption). The point of our methodology is that the functions linking the former to the latter are, in many cases, learned via ML rather than decided when writing up the program.

Target Variable: The schedule, containing which PM must hold each VM.

Problem Parameters: Host Resources: CPU, Memory, and Bandwidth characteristics per PM; Job Load: Amount of load (number of requests, average bytes per request, average CPU process time per request, etc) for each different topological load source; Previous Schedule: Useful to count migrations; Latencies: latency between each load source and each PM (PMs in the same DC will have the same values), also latency between two hosts; Image Size: size of VM images, to calculate the time of transportation; Basic Response Time and QoS Tolerance Margin: The two parameters in the SLA describing its fulfillment according to the resulting RT.

Problem Constraints: 1) We assure a VM involved in this scheduling round is finally placed in one and only host. 2) The resources granted to the set of jobs allocated in one host must not exceed the amount of resources the host has available. 3) For each host we set the power consumed by all its granted resources. 4) For each job we set whether it is being migrated or not, and its latency between origin and destination. 5) Resources required and granted to a job given its tentative placement. 6) Response Time (production RT) given the load, required and granted resources, also the transport RT for each location. 7) SLA fulfillment for each job, from the RT obtained, the basic RT and tolerance margins agreed with the customer. This function can be used over each request or over the average RT (weighting the different load sources).

Multi-core computers energetic consumption depends primarily on their CPU usage, and once on-line, increasing load does not make consumption to grow linearly. E.g. in a Intel Atom 4-Core machine (the ones used here), the energy consumption (in Watts per hour) when all CPUs are 29.1, 30.4, 31.3, and 31.8 when 1, 2, 3, and 4 CPUs are active. This implies that two such

machines using one processor each consume much more energy than a single machine executing the same work on two (or even four) and shutting down the second machine. This explains the potential for power saving by consolidation. Further, usually in DCs, for each 2 watts consumed an extra watt is required for cooling functions, another reason to reduce energy consumption.

To calculate the migration penalty we must assume that during migration (freeze VM, transport the image, restore VM) the SLA fulfillment may become 0, so we compute the migration time as time the SLA may be 0. Finally, to determine required and given resources, we can get information from the monitors, or use the ML predictors to be explained in the next subsection. Also to determine the RT and SLA, in a reactive system we can try to obtain it statistically from the previous executions, while we are doing it proactively using our learned models.

8.3.1 Adaptive Models

When making decisions we often find that the required information 1) is not available, 2) is highly uncertain, 3) cannot be read because of privacy issues, or 4) obtaining it interfere with the system. Examples of this occur when reading from both PMs and VMs, and information coming from VMs is extremely delicate to handle and interpret. Observed resource usage can be altered by the observation window, the spawn of time between samples, or the stress of the same PM. Overheads of virtualization also add noise to the resource observation, independently of the load received by each VM. Further pinning the VM to read information from its internal system log could be against the customer privacy. Furthermore monitors can add also overhead to the PM altering the VMs performance (e.g. during experiments we observed sampling monitors peak up to 50% of an Atom CPU thread). We apply all the knowledge and techniques developed and described previously on Chapter 7 about datacenter model learning, employing methods from machine learning, creating system models automatically from real examples of observed past behavior.

Here we predict all elements that could be obtained through monitoring considered relevant to decide VM placement according to maximize benefit. From load characteristic of each web-service and its clients (Requests per Time Unit, average Bytes per Request, average Computing Time per Request in no-stress context), we learn and predict the resources demanded by the VM to produce replies (CPU, Memory and IO network traffic). Being the VMs not like HPC jobs, the total used PM CPU is not the sum of the VM CPU peaks and can not be computed just as its sum, so we learn also the PM CPU usage for a set of VMs (the total occupation of the PM in CPU terms). We assume the PM Memory usage to be the sum of VM real allocated memory, being memory not as shifty as CPU. Also network IO can be considered the sum of VM IO. With these predicted values, plus information from the gateway element (queue sizes for not replied requests towards each VM), we can learn and predict the variables involved in profit: Response Time and/or SLA fulfillment level, by using the load information (current or expected for next time unit), the predicted resources given it, the predicted context (PM CPU,MEM,IO), and the queues information.

As most of the functions that model resources and response times are piecewise linear or polynomial functions approximated by piecewise linear functions, we studied linear regressions and M5P regression trees among other discarded regression techniques (i.e. RepTrees). For SLA prediction, being hard to fit a regression function given the observed examples, we used a K-Nearest Neighbor technique to predict a SLA for each VM, by comparing each situation with previous seen before. Table 8.1 shows the details for each predicted element validation.

Once seen the results for RT and SLA prediction, we decided to use the SLA predictor instead of the RT (and then compute the SLA with it) to drive the VM placement.

8.3.2 Scheduling Algorithms

As our model becomes easily integral-linear we could use a MILP solver, but as seen in previous chapters, using solvers like GLPK [65] or more specialized solvers like GUROBI [75] required several minutes to schedule 10 jobs among 40 candidate hosts. Now, by having more complex functions (SLA function becomes a K-NN method, visiting several times all $examples \times variables$ for each tentative solution), MILP methods become intractable if we want to get schedules at least

	ML Method	Correl.	MAE	Err-StDev	Train/Val	Date Range
Predict VM CPU	M5P ($M = 4$)	0.854	4.41%$_{CPU}$	4.03%$_{CPU}$	959/648	$[0, 400]$ %$_{CPU}$
Predict VM MEM	Linear Reg.	0.994	26.85 MB	93.30 MB	959/1324	$[256, 1024]$ MB
Predict VM IN	M5P ($M = 2$)	0.804	1.77 KB	4.01 KB	319/108	$[0, 33]$ KB
Predict VM OUT	M5P ($M = 2$)	0.777	25.55 KB	22.06 KB	319/108	$[0, 141]$ KB
Predict PM CPU	M5P ($M = 4$)	0.909	14.45%$_{CPU}$	7.70%$_{CPU}$	477/95	$[25, 400]$ %$_{CPU}$
Predict VM RT	M5P ($M = 4$)	0.865	0.234 s	1.279 s	1887/364	$[0, 19.35]$ s
Predict VM SLA	K-NN ($K = 4$)	0.985	0.0611	0.0815	1887/364	$[0.0, 1.0]$

Table 8.1: Learning details for each predicted element and selected method. All training processes are done using random split of instances (66/34)

once per hour. We use again the alternative approximate greedy algorithms, as seen in previous chapters (Chapter 7), the classic Ordered Best Fit to solve the program. The profit function is the responsible of computing the SLA, energy, migration and latency factors, computing the profit for each tentative placement. Also the local DC schedule will compute it using local values (fixed watt-price and low migration latencies), while the general will include them all, also each host becomes the representative host for each DC (if, as said, we don't want to use all hosts of all DCs in a centralized schedule).

8.4 Experiments and Studies over the Multi-DC

To test our approach, we have performed experiments to 1) see how using the learned models we can schedule by consolidating while taking care of QoS, without introducing direct expert knowledge to the best-fitting algorithm; and 2) how the model can schedule across DCs based on client proximity and energy cost saving, distribute the VMs by following the load source against following the cheapest energy price against consolidating and keeping the QoS. These tests have been performed replicating real web-service workloads on real hosting machines.

8.4.1 Environment Description

Instead of using HPC-designed machines, where to fulfill any resource requirement becomes adding physical resources to increment the computational power, we have based the experiments on low-energy consumption machines (Intel Atom 4 Core), where resource management is critical in order to hold as most load as possible without degrading QoS. PMs run Oracle VBox virtualization platform, and each VM runs a web-service software stack (Apache, PHP, MySQL). The workload used corresponds to the Li-BCN Workload [26] previously described.

Our scenario, as a case of use, is composed of 4 DCs in different continents (e.g. Brisbane, Australia; Bangaluru, India; Barcelona, Spain; Boston, Massachussets), connected by high-speed network (network costs are not part of this work, but are kept as future work). For each DC there is an amount of clients accessing to the services according to their local workload. [Note: as we do not dispose of DCs in all 4 places, we performed the experiments in our local DC, splitting it and setting up the corresponding network latencies and delays between pretended DCs and clients].

To price each element involved in the system, we established that providers behave as a cloud provider similar to Amazon EC2, where customers rent VMs in order to run their web-services (0.17 euro per VMs). For energy costs, we obtained the energy cost (euros per kWh) for the different places where we have a DC placed, so the cost of running a PM will depend on the DC where it is placed. Also, the migration costs depend on the latency and bandwidth between DC connections. We took as example the intercontinental network of the Verizon network company [165] to obtain latencies between locations and assumed a fixed bandwidth of 10 Gbps.

The RT, as a QoS measure in our SLA, is measured from the arrival of a request to the exit of the reply for it through the Internet Service Provider (ISP). As SLA parameters, we set as RT_0 the values 0.1s, as experiments on our system showed that it is a reasonable response value obtained by the web service without stress, and the α parameter is set to 10 (SLA fulfillment is 0 if $RT \geq 10RT_0$).

8.4.2 Intra-DC Comparatives

First experiments are to check the benefits of driving an intra-DC scheduling for VMs using the learned models. As seen in previous chapter 7, best-fit performs better among greedy classical ad-hoc and heuristics, and here we check it against the environment. We compare the best-fit algorithm, checking if a VM has room in the PM given the recent required resources (last 10 minutes), and optimizing just power and latency to clients; the best-fit algorithm with resource overbooking (BF-OB), reserving the double required resources for each VM, as an example of how we could reserve more resources to assure unexpected load peaks; and the ML enhanced best-fit, where the fitting function becomes the SLA prediction, having into account the predicted CPU, MEM and IO required for each VM. The goal is to see how the learned models drive the best-fit into a consolidate/de-consolidate strategy given the SLA, without introducing much expert knowledge.

We set up 4 PMs with OpenNebula and VirtualBox, holding 5 VMs, a PM acting as gateway and DC manager, and 4 machines generating LiBCN'10 scaled load towards the 5 VMs. Figure 8.1 shows the results of running the workload for 24 hours, with a scheduling round of 10 minutes.

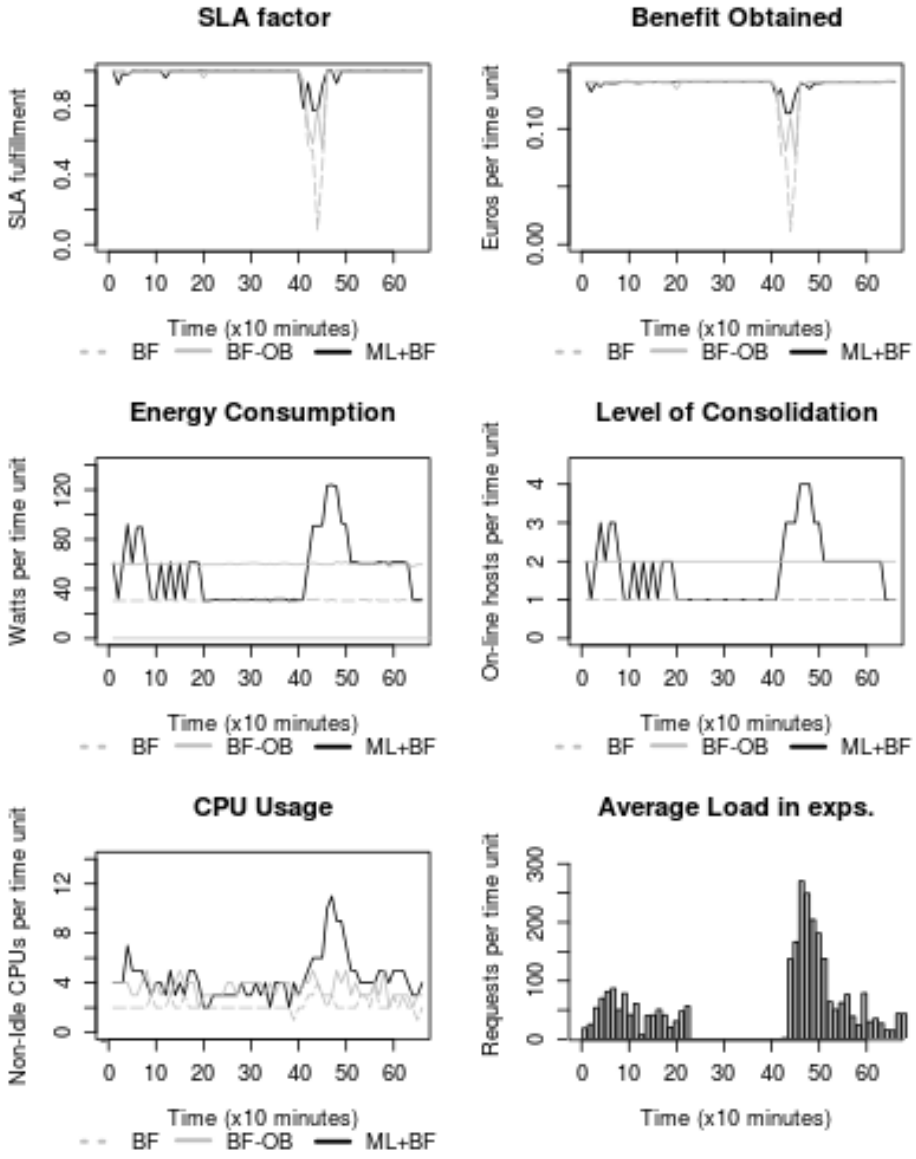


Figure 8.1: Results and Factors for Intra-DC Scheduling

The Best Fit algorithm with ML enhancement (de)consolidates constantly to adapt VMs to the load level, while Best Fit without ML considers that given the monitored data it is not required to do so, and uses less CPUs and less PMs risking the SLA. So the ML approach learns to detect situations where SLA fulfillment may not be achieved (because of CPU competition, memory exhaustion and/or IO competition), hence migrating sufficient VMs to other machines with better contexts. The drawback of de-consolidate is the extra energy utilization, but while SLA revenue pays for the energy and migration costs, the Best-Fit with ML will usually choose to pay energy to maintain QoS.

Not reported here, for space reasons, is the fact that these ML-augmented versions can automatically adapt to changes in task execution prices, SLA penalties, and power prices. Adapting the ad-hoc algorithms to these changes requires expert (human) intervention, and is simply unfeasible in the highly changing scenarios envisioned for the future, where virtual resources, SLA penalties, and power prices will interactively and constantly be in negotiation, for example by means of auctions and automatic agents.

8.4.3 Inter-DC Comparatives

After checking the learned consolidation best-fit strategy we study the inter-DC scenario, where VMs can be placed in multiple DCs, each one with different energy prices and different latencies between them and clients. Issues with multi-DC systems are that often the best placement according to SLA requires pay more for energy, or migration penalties make better a different placement, consolidating and moving VMs differently to a single fixed factor, but as the combination of all the factors.

As a case of study, here we set a PM per DC, as the average on-line PM for that DC (as the local scheduler will arrange local PMs to a correct SLA fulfillment level, this PM will represent an available on-line machine to host an incoming VM). Each DC has an access point for clients (an ISP) machine collecting all the requests sent into any VM on our system coming from the region the DC is placed, and requests going to a VM on a non-local DC will be served through our network, suffering the latency between the local DC and the remote DC. We apply our workload upon each VM from each ISP, each place being scaled different and having each region different time-zone and load time pattern. Table 8.2 shows the used prices and latencies.

	Euro/Wh	LatBRS	LatBNG	LatBCN	LatBST
Brisbane (BRS)	0.1314 Wh	0	265	390	255
Bangaluru (BNG)	0.1218 Wh	265	0	250	380
Barcelona (BCN)	0.1513 Wh	390	250	0	90
Boston (BST)	0.1120 Wh	255	380	90	0

Table 8.2: Prices and Latencies table (Latencies in ms [10Gbps line])

Follow the Load and Consolidation

First of all we perform a "sanity check", looking at the movements of VM without adding SLA or Energy factors yet (the simple "follow the load" policy), having a driving SLA function using only the request latency. Given this, the best-fit tries to put each VM as close to the major load source as possible. Figure 8.2 shows the movement of a single VM being driven only by this kind of SLA, without any resource competition or energy awareness. The VM follows the main source load to reduce the average latency to its globally distributed clients.

After following the load we also checked the "energy consumption" factor. When the scheduler becomes energy aware it tends to consolidate, having also into account client proximity and migration costs (it will consolidate to the place where it is easier to move the VMs, versus the energetic cheapest place to do it). Results of this tests are as expected, as VMs are consolidated always in just one DC, generally the one nearest to the clients. Results are not reported.

Benefit of De-locating Load

Next, we study the differences between a scenario with a single DC (in an averaged location for energy costs and latencies), where all VMs are held receiving all the load, and an scenario where

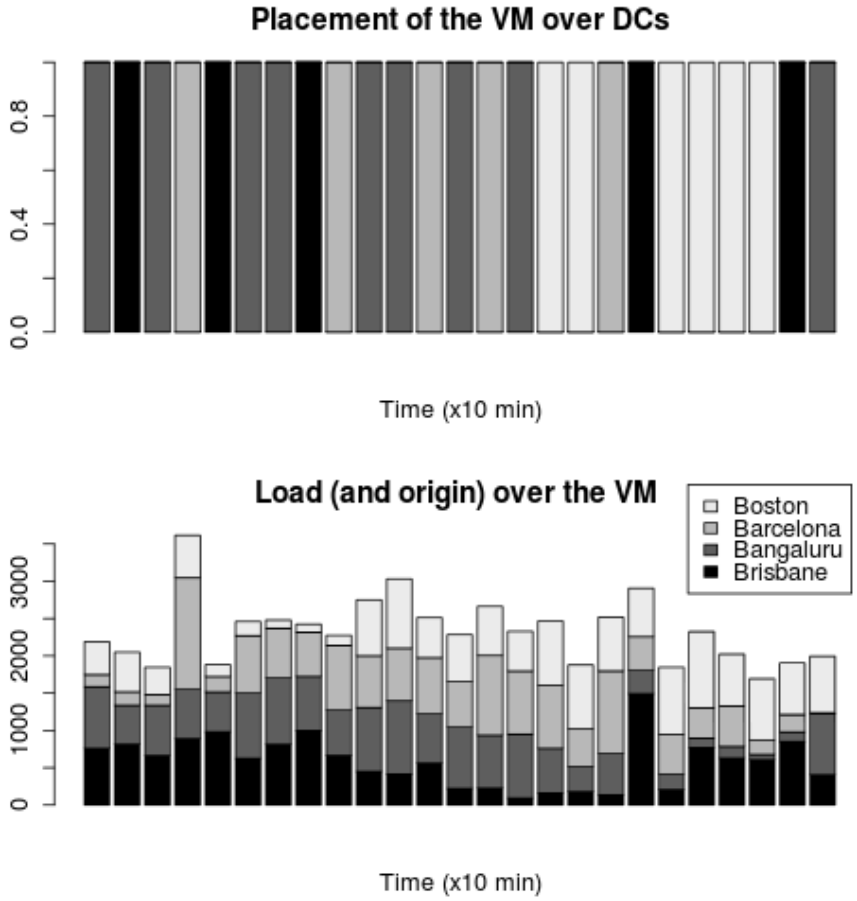


Figure 8.2: VM placement following the Load for Inter-DC Scheduling

this DC can de-locate VMs (migrate VMs to neighbor DCs temporarily) when it is overloaded. Despite having worse latencies and migration overheads when de-locating, SLA fulfillment increases from an average SLA of 0.8115/1 to an SLA of 0.8871/1 per VM doing this. This would translate, in the current experiment, to an average net benefit increase of 0.348 euro/VM a day.

In this experiment, the migration to another DC incurs in a latency increase of 0.09 to 0.39 seconds, but happens at the time when the load was so severe on the VM that its response time had degraded to about these 0.09 seconds over the desired 0.1 seconds. We observe that for lower SLA increments it prefers to consolidate in the local DC. Obviously the de-location threshold will depend on the RT_0 values and inter-DC latencies, but it is clear that the method is able to decide when de-locating VMs is worth it or not.

Full Inter-DC Scheduling

Once checking latency and energy factors, and observing the de-location benefit from a DC point of view, we perform the complete scheduling of the multi-DC system. From experiments, as seen in Figure 8.3, we have seen several details:

1. When having heavy load, the scheduler distributes VMs across DCs, (de)consolidating as doing with intra-DC hosts (assuming that all hosts in the DC are represented by the current PM, also as seen previously, if the DC has available machines, they can be included in the scheduling process with the representative-full PM of that DC). Thus, the SLA fulfillment and its revenue is still the most important factor as it drives (de)consolidation [see high load moments, or when SLA is lower than 1].

2. When SLA is not compromised, energy energy consumption pushes for consolidation [see

low load moments] into the DC with most cheap energy.

3. When attempting to move a VM do not suppose any improvement in SLA or energy saving, the VM stays in its DC or is consolidated to the nearest DC in latency.

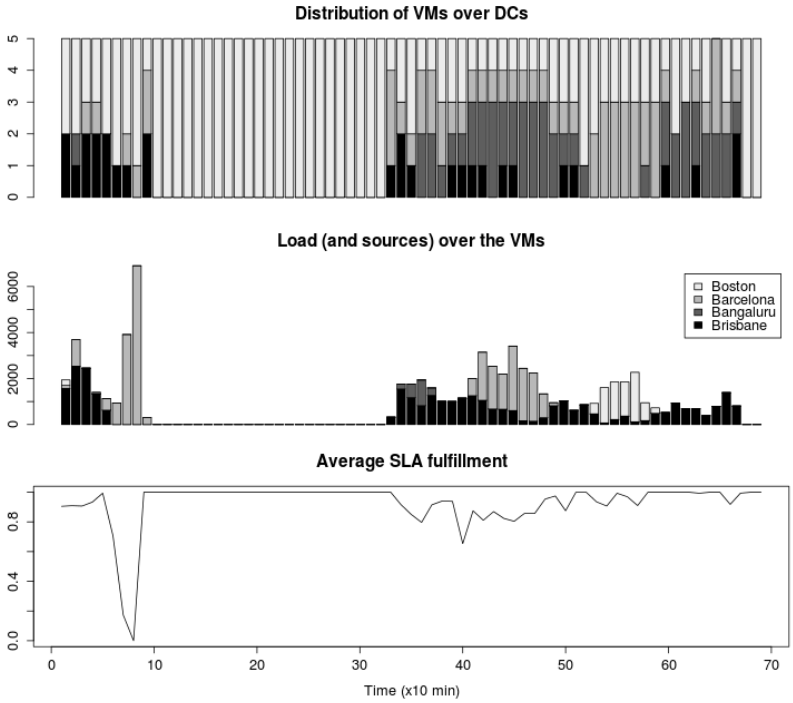


Figure 8.3: Results and Factors for Inter-DC Scheduling

Again, de-consolidation effects are seen when load (requests) increases or requests become more expensive to answer, we are able to improve the SLA thanks to de-consolidation, countering the migration penalties also enforcing the reduction of service-client latencies. Note that the workload generator produced a flash-crowd effect in the workload in minutes 70-90, for about 15 minutes, which clearly exceeds the capacity of the system. We kept this part of the workload in the test for realism.

Benefit of Inter-DC Scheduling

The fundamental question we address is whether inter-DC optimization is better than intra-DC optimization, that is, whether distributing the VMs depending on the global load and (de)consolidating at multi-DC level is better than keeping moving VMs within their own DCs only. Here we compare two scenarios: 1) The static global multi-DC network, where the VMs for each DC stay fixed without moving across DCs, where clients around the world can access to every version but each web-service stays always in the same DC near its potential clients or customer selected DC; in other words, DCs do not cooperate by exchanging VM's, but just by redirecting the load they receive and is intended for VM's located somewhere else. And 2) the dynamic multi-DC scenario we propose, where VMs may migrate among DCs to improve global benefit.

The benefit of the dynamic approach is basically the capability of moving the VMs towards the place where the energy is cheaper and/or available. As seen already in Figure 8.3, when load is low, the algorithm chooses to consolidate in DCs with lowest energy cost. For higher loads, de-consolidation becomes necessary, and energy price is still one of the factors to choose where de-consolidation occurs, together with latencies, SLAs, and migration overheads. At this stage, we chose for realism to use actual electricity prices for the four locations we have considered, which are relatively similar. As energy costs rise and markets become more heterogeneous and competitive, one should anticipate larger variations of energy prices across the world, and the

benefit of inter-DC optimization priming energy consumption should be more obvious. This is particularly so as renewable sources such as solar energy become more widespread, because of their hour-to-hour variability and its very low cost once the production infrastructure is in place.

Figure 8.4 shows the comparative among the static context and the dynamic, when wanting to consolidate VMs among DCs.

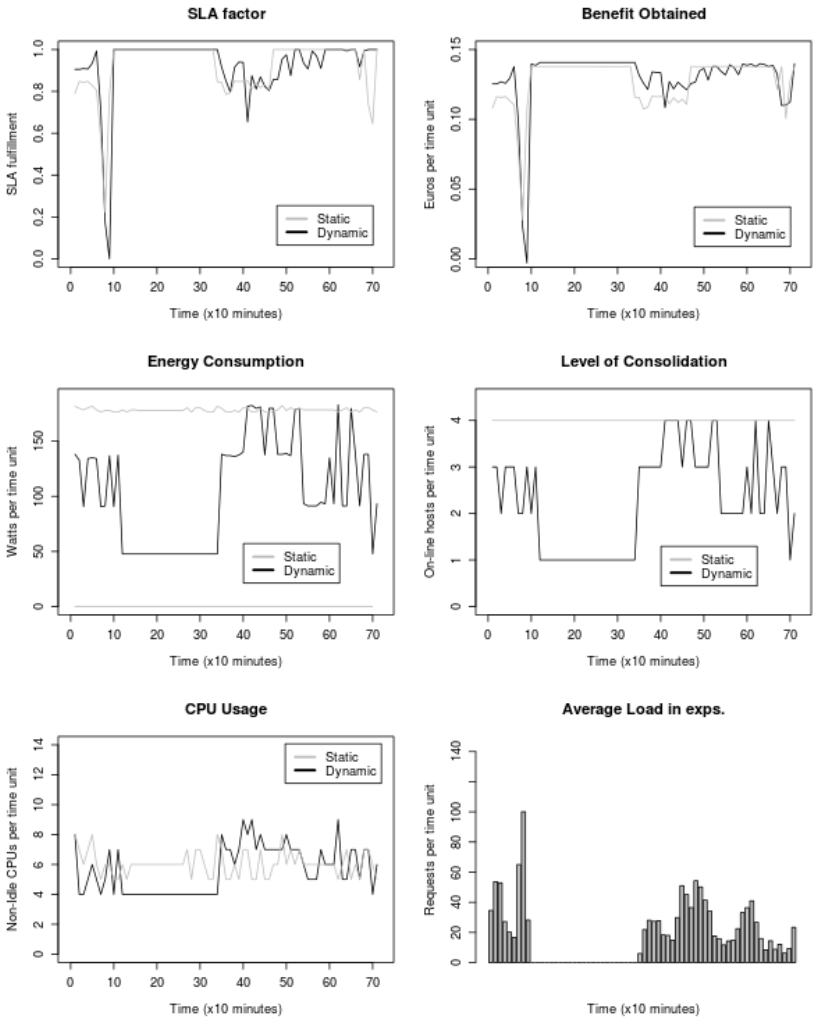


Figure 8.4: Comparative Static vs Dynamic Inter-DC for 5 VMs

The large savings in energy is largely due to our experimental limitation (one PM per DC), which leaves no room for intra-DC energy savings by consolidation. One can observe, though, that even in this restricted setting the algorithm manages to slightly improve global average SLA and revenue *while* reducing energy costs.

	Avg Euro/h	Avg Watt/h	Avg SLA
Static-Global	0.745	175.9	0.921
Dynamic	0.757	102.0	0.930

Table 8.3: Comparative of results for the multi-DC per 5 VMs

Previous studies [70] showed that consolidation can achieve a power consumption reduction of more than 30% without counting the energy saving on cooling overheads (which may cause around a 1.5 increase in power consumption). So while maintaining SLA stable, we are able to improve energy consumption in a 42% by (de)consolidating in an inter-DC way, and further

improve benefit by a 2% a day, for VMs that can not be consolidated in their local DCs.

Trade-Offs for QoS and Energy Costs

Finally, trade-offs between Quality of Service and energy costs depend in the amount of load the VMs are receiving. Figure 8.5 shows the relation of the 3 variables from the observations of the given scenario ("load" is represented by amount of requests per time unit, as the most significant attribute of load). Given the amount of load, as we want to improve the SLA fulfillment we are forced to consume more energy. For each level of load he can infer a characteristic function SLA vs Energy, so we can choose how much energy we want to spend to achieve a desired level of QoS. With low load we do not require to spend much energy as QoS will be always around [0.9,1.0], and in some situations, given the huge amount of load per time unit overwhelming a PM, we will be unable to achieve the maximum SLA level despite consuming the maximum energy available. So, once knowing the load to handle, the datacenter manager must decide how much consolidation applies depending on the SLA fulfillment levels he/she desires to obtain.

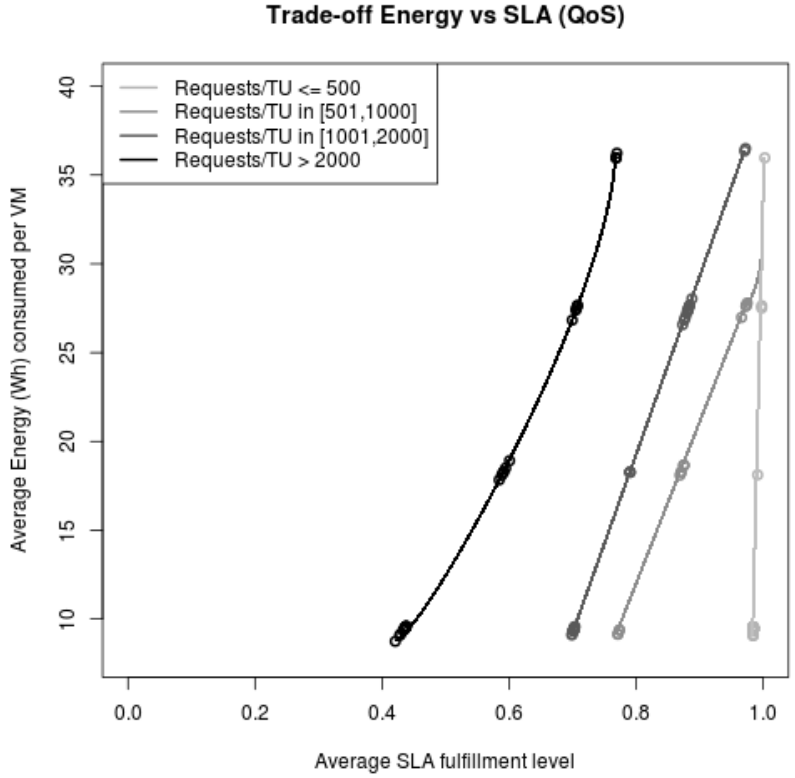


Figure 8.5: Relation of the SLA vs Energy vs Load

8.5 Conclusions for Multi-DC Modeling and Managing

Optimizing the schedule and management of multi-DC systems requires balancing several factors, like economic revenues and operational costs, depending on Quality of Service and also the energy to run the required infrastructures. As seen in previous chapters, this problem can be modeled as a mathematical problem, solved using heuristics or approximate algorithms. Also it can be improved using machine learning models to resolve uncertain or unavailable information, making decisions adapted to the environment without requiring much expert knowledge, and providing the operators and DC architects knowledge of the learned behaviors.

Taking advantage of virtualization technology, we presented a model to solve a multi-DC scheduling for web-service, using learned models to enhance its decisions towards enhancing quality of service, keeping in mind energy consumption and transmission time factors. Experiments showed that the ML models can provide the required information to (de)consolidate across DCs according to load towards web-services. Also we have seen that the model is able to schedule according to proximity to clients, and improve the energy costs per location without degrading the quality of service.

Important issues must be treated in the future work, like how we decide which VMs are excluded from inter-DC scheduling or which PMs are offered as host candidates for scheduling, affecting directly to scalability of the method. This can provide us information about how many PMs/VMs we can manage per scheduling round, and deciding whether solving the model using exhaustive solvers, being exponential in time, or the approximate algorithms, that grow linear in time with the number of candidate PMs also linear in time with the number of VMs to be (or not) migrated.

From this work on, more operational costs like networking costs and management of bandwidths according to punctual requirements can be studied, also the inclusion of green energy availability into the schemes to reduce not only energy costs but also environmental impact of computation. In the next section, an approach of DC placement is shown, where DCs are depicted with high detail, presenting all the elements involved in their construction based on solar and wind capacity of production for each tentative location. Also a model to schedule load among DC following the renewables, totally compatible and attachable to our presented mathematical DC model.

The work presented in this chapter will be published in the "Workshop on Power-aware Algorithms, Systems, and Architectures 2013" [29] (2013).

Chapter 9

A Green Approach for Placing DataCenters

In this chapter we do not address directly the management of existing datacenters, but the design and placement of it, seeking to quantify green datacenter building costs for high-performance computing cloud service providers. A set of suboptimal choices for allocating jobs and load can hardly be compensated for once the infrastructure has been deployed, as a bad decision on placing infrastructures has extremely more impact than a placing incorrectly a VM on a scheduling round. We first propose an optimization framework for intelligently selecting the best geographical locations for the provider's green datacenters. We characterize areas around the world as potential locations for green datacenters, also we illustrate the location selection tradeoffs by quantifying the minimum cost of achieving different amounts of renewable power at different levels of confidence. Further we design and implement a cloud middleware capable of migrating VMs across the green DCs to follow the availability of renewable energy.

9.1 Introducing Green DataCenter Placement

In the following work, we seek to demonstrate that green cloud services can be built at low cost. For this case of study, it focuses on high performance computing cloud services, such as Amazon's EC2 Cluster Instances. HPC applications typically do not involve user interactivity, allowing the cloud provider to place its datacenters at the best world-wide geographical locations for renewable energy generation, regardless of how remote they are. Moreover, these applications often run for long periods of time, allowing the provider to migrate them across locations to follow the availability of renewable energy.

With these observations in mind, here it is proposes a cost-centric optimization mathematical model for intelligently selecting the best geographical locations for the provider's green datacenters. The capital costs of green datacenters include purchasing and installing the solar and/or wind plant, land acquisition, datacenter construction, and bringing enough network bandwidth and grid (or "brown") electricity to the datacenter. The operational costs include paying for brown electricity and for the system administration staff. Many of these costs depend heavily on geographical location. For example, the size of a solar installation and the amount of energy spent in cooling depend on how exposed the location is to sun light. Further, based on the framework, we characterize areas around the world as potential locations for green datacenters. The characterization includes potential for solar and wind energy, average temperature, brown electricity prices, average datacenter Power Usage Efficiency (or simply PUE), and proximity to network backbones and brown electricity transmission lines.

Furthermore, we illustrate the location selection tradeoffs by quantifying the minimum cost of building green HPC cloud services of different sizes. This study explores a large parameter space covering different amounts of required renewable power at different levels of confidence. The benefit of provisioning batteries at the green datacenters is also explored. Finally, as solar and wind energy are intermittent, a scheduling algorithm for cloud middleware is designed and

implemented, capable of migrating virtual machines across the green datacenters to follow the availability of renewables. The middleware enables service providers to maximize the amount of renewable energy they consume during periods of less-than-peak load.

Related Work

At this time no other works have considered the placement of exclusively green powered datacenters at global scale, having in mind all the relevant costs involved on datacenter deployments [2, 123, 149]. The idea of revising the usage of grid while maximizing the use of off-grid renewable energy is a hot topic nowadays. As seen in the Background Chapter, works like [150] focuses on renewable energy policies on wind powered DCs. Other works like [71] and [103] consider load management depending on green energy availability, and works like [94] study policies on the energy prices for datacenters near green energy sources. Also works like [99, 104, 182] study algorithms for geographical load optimizing the usage of renewable energy depending on availability and costs.

Here, a simple implementation of such policies over a well known datacenter virtualization platform, using as a goal the maximum usage of self-produced green energy, is presented.

9.2 Green Energy DataCenter Placement

The goal here is to efficiently select locations for one or more datacenters, so that the overall cost is minimized, and together the datacenters can always provide a given level of green energy with a given level of confidence. To that end, here the most important parameters in the selection process are defined. Based on these parameters, a cost model and optimization problem can be formulated.

Domain and Parameters of the Problem

Table 9.1 lists the entire set of parameters for the mathematical model. They range from inputs provided by the user to parameters that we seek to instantiate via optimization. Among the more interesting parameters are those related to costs, green energy generation, and confidence for green energy generation.

Symbol	Meaning	Unit
	List of Parameters	
$totalCapacity$	desired minimum total power capacity of DC network for computing	kW
$minAvailability$	desired minimum availability for DC network	#DC
$minGreen$	desired minimum percentage of green energy	%
$confidence$	desired minimum confidence of achieving $minGreen$	%
$\alpha(d,t)$	solar energy production factor at d during t	[0,1]
$max\alpha(d)$	maximum solar energy production factor at d	[0,1]
$\beta(d,t)$	wind energy production factor at d during t	[0,1]
$max\beta(d)$	maximum wind energy production factor at d	[0,1]
$PUE(d,t)$,	PUE at d during t	[1,1.69]
$maxPUE(d)$	maximum PUE at d	[1,1.69]
$priceLand(d)$	land price at location d	$\$/m^2$
$areaDC$	land area needed per kW DC capacity	m^2/kW
$areaSolar$	land area needed per kW solar energy production	m^2/kW
$areaWind$	land area needed per kW wind energy production	m^2/kW
$costLineNet(d)$	cost to layout optical fiber at location d to closest network backgone	\$
$costLinePow(d)$	cost to layout power line at location d to closest power plant	\$
$priceEnergy(d)$	brown energy (electricity) price at location d	\$/kWh
$priceBuildDC(c)$	price of building a DC with c power capacity	\$/kW
$priceBuildSolar$	price of building a solar energy plant	\$/kW
$priceBuildWind$	price of building a wind energy plant	\$/kW
$priceMaintain$	price to maintain DC	\$/kW
$priceOp$	price of operation of IT equipment	\$/serv

$priceServer$	price of a server	\$/serv
$priceSwitch$	price of a network switch	\$/switch
$serverPower$	server power consumption	W/srv
$switchPower$	network switch power consumption	W/switch
$serversPerSwitch$	number of servers per switch	srv/switch
$priceNetBandServer$	cost of external network bandwidth per server	\$/srv
$priceBatt$	price of batteries	\$/kWh
$battEff$	efficiency for charging batteries	%
$nearestPPCap(d)$	size of the nearest power plant to location d	kW
List of Variables		
$placed(d)$	a DC is placed at location d	[0,1]
$capacity(d)$	maximum power capacity for computing of DC at d	kW
$solarCap(d)$	solar power capacity at location d	kW
$windCap(d)$	wind power capacity at location d	kW
$battCap(d)$	size of batteries needed at location d	kWh
$compPow(d,t)$	computing power demand for DC at d during t	kW
$extraGreenPow(d,t)$	average unneeded green power produced at d during t	kW
$battChargePow(d,t)$	average power needed to charge batteries at d during t	kW
$netChargePow(d,t)$	average power net metered into the grid at d during t	kW
$solarPow(d,t)$	average solar power produced at d during t	kW
$windPow(d,t)$	average wind power produced at d during t	kW
$brownPow(d,t)$	average brown power needed at d during t	kW
$battDisPow(d,t)$	average power drawn from batteries at d during t	kW
$netDisPow(d,t)$	average net metered power from grid at d during t	kW
$migratePow(d,t)$	$max(compPow(d,t) - compPow(d,t-1), 0)$	
$greenPow(d,t)$	$solarPow(d,t) + windPow(d,t) + battChargePow(d,t)$ $-battDisPow(d,t) + netChargePow(d,t) - netDisPow(d,t)$	
$powDemand(d,t)$	$(computePow(d,t) + migratePow(d,t)) \cdot PUE(d,t)$	
$powAvail(d,t)$	$greenPow(d,t) + brownPow(d,t)$	
$landCost(d)$	$priceLand(d) \cdot (capacity(d) \cdot areaDC + solarCap(d) \cdot areaSolar$ $+windCap(d) \cdot areaWind)$	
$buildCost(d)$	$capacity(d) \cdot maxPUE(d) \cdot priceBuildDC(capacity(d))$ $+solarCap(d) \cdot priceBuildSolar + windCap(d) \cdot priceBuildWind$	
$maintainCost(d)$	$capacity(d) \cdot maxPUE(d) \cdot priceMaintain$	
$numServers(d)$	$capacity(d)/(serverPower + switchPower/serversPerSwitch)$	
$serverCost(d)$	$numServers(d) \cdot priceServer$	
$switchCost(d)$	$(numServers(d)/serversPerSwitch) \cdot priceSwitch$	
$networkCost(d)$	$numServers(d) \cdot priceNetBandServer$	
$operatorCost(d)$	$numServers(d) \cdot priceOperator$	
$battCost(d)$	$battCap(d) \cdot priceBatt$	
$brownEnerCost(d)$	$\sum_{t \in T} brownPow(d,t) \cdot t \cdot priceEnergy(d)$	
$CAP\_ind(d)$	$costLinePow(d) + costLineNet(d)$	
$CAP\_dep(d)$	$landCost(d) + buildCost(d) + serverCost(d) + switchCost(d) + battCost(d)$	
$OP(d)$	$operatorCost(d) + maintainCost(d) + networkCost(d) + brownEnerCost(d)$	

Table 9.1: Parameters and Variables for the Placement Mathematical Problem

Explanation of Costs

The overall cost of a network of datacenters can be broken down into capital (CAPEX) and operational (OPEX) expenditures. The CAPEX costs are those investments made upfront and depreciated over the lifetime of the datacenters. CAPEX can be further divided into capital costs that are independent of (CAP_ind) and those that are dependent on (CAP_dep) the number of servers to be hosted.

The CAP_ind costs relate to bringing electricity and external networking to the datacenters. Although the amount of electricity and external bandwidth depends on the number of servers, the base cost of laying out any transmission line and/or optical fiber dominates. These costs for each location can be estimated from the distance between the location and 1) the closest

transmission line or power plant; and 2) the closest network backbone.

The *CAP_dep* costs relate to land acquisition, datacenter construction, construction of the green-energy plants, purchasing and installing the power delivery, backup, and cooling infrastructures, and purchasing servers and internal networking equipment to be hosted by the datacenters. The land price varies according to location, whereas the other costs do not to a first approximation. All of these costs depend on the level of redundancy that will be built into each datacenter. The construction cost is typically estimated as a function of the maximum power to be consumed by the datacenter. This maximum power is that required by the maximum number of servers and networking gear when running at 100% utilization times the maximum expected PUE of the datacenter. The PUE is computed by dividing the overall power consumption by the power consumption of the computational equipment. The PUE is higher when temperature and/or humidity are high, since cooling consumes more energy under those conditions.

The OPEX costs are those incurred during the operation of the datacenters. These costs relate to the maintenance and administration of the equipment, external network bandwidth use, and the electricity required to run the datacenters. There is also a cost for water that currently is not considered, but it can be easily added to the model. Electricity cost can be computed based on the IT equipment's energy consumption, the PUE, the amount of green energy generated locally, and the electricity price. This cost may vary with location.

Finally, lower taxes and one-time incentives are another important component of the cost of a datacenter. For example, some states in the US lower taxes on datacenters, as they generate employment and wealth around them. This component depends on the nature of the savings and applies to each cost component in a different way. Although this component is not considered further, it is also easy to include it in our framework.

Green-Energy Generation

Two key factors that affect the cost and benefit of building green-energy generation systems include *efficiency* and *capacity factor*. Efficiency refers to the percentage of sunlight energy and wind energy that is transformed into electricity for solar and wind energy plants, respectively. The efficiency of today's most affordable PV technology (multi-crystalline silicon) hovers around 15% and around 50% for wind (where the theoretic limit wind efficiency to 59% due to rotational generators).

Capacity factor refers to the percentage of the maximum theoretical energy production (e.g., 24 hours of maximum sunlight every day for a solar energy system) that is actually produced. Capacity factors vary depending on location and weather. For example, Berlin (Germany), New York (NY-US), Canberra (Australia), and Phoenix (AZ-US) have solar capacity factors of roughly 13.5%, 16.4%, 20.2%, and 22.9%, respectively. (For wind it would be 3.4%, 18.9%, 8.4%, 3.4%.)

In this model, we combine the efficiency and capacity factor of solar and wind generation at a location into α and β respectively, being α and β the ratio of power production versus the maximum power capable to be produced.

Confidence for Green-Energy Generation

Generation of green energy is dependent on weather conditions. Thus, to achieve a minimum fraction of green energy used at all times may be very expensive since it would require significant over-provisioning of the green power plants. Here instead a minimum percentage of green energy is target with a given confidence for each time segment (e.g., 1 hour) within a longer time span (e.g., 1 year).

Let $greenCap$ be the maximum power capacity of a green energy plant, $greenProd(t)$ be the average green power produced by the plant during time period t, $P(greenProd(t) \geq x)$ be the probability of $greenProd(t) \geq x$, and $powerDemand(t)$ is the total power demand of a datacenter during t, then to meet a target minimum fraction of green power used at the datacenter, $minGreenFrac$, with a confidence of $conf$, means that

$$\forall t \in T, P(\frac{greenProd(GP,t)}{powerDemand(t)} \geq minGreenFrac) \geq conf \tag{9.1}$$

Granting Availability for DataCenters

We model the availability of the network of datacenters as:

$$Availability = \sum_{i=0}^{n-1} \binom{n}{i} \cdot i \cdot (1-a)^{n-i} \cdot (a)^i \tag{9.2}$$

where n is the number of datacenters and a is the availability of each datacenter. This model has been used in multiple previous reliability and availability works, e.g. [130][142]. The availability of each datacenter depends on the level of redundancy in its design. Industry commonly classifies datacenters into tiers [159], where each tier implies a different redundancy level and expected availability. At one extreme, Tier I datacenters have a single path for power and cooling distribution. At the other extreme, Tier IV datacenters have two active power and cooling distribution paths, with redundant components in each path. Existing Tier I datacenters have been found to achieve an availability of 99.67%, whereas Tier II datacenters achieve 99.74%, Tier III datacenters achieve 99.98%, and Tier IV datacenters achieve 99.995% [159].

Formulating the optimization problem

Using the available parameters the cost model and optimization problem can be formally defined. The problem setup consists of an HPC cloud service provider that seeks to selet locations for a set of datacenters out of a set of potential locations for them (D). The set of datacenters by design should be able to support a workload requiring a given power capacity (e.g., 10MW), with a given percentage (e.g., 80%) coming from local green sources at a given confidence. The optimization goal is to minimize the overall cost of the datacenter network, while respecting the power and availability constraints.

The inputs to the optimization are (1) the minimum total power capacity that must be supported by the datacenter network at any given point in time, (2) the minimum percentage of green energy, (3) the confidence that the minimum percentage of green energy is met within a given time period, (4) the amount of redundancy that will be built into each datacenter, (5) the CAPEX and OPEX costs for each location $d \in D$, and (6) the sizes of the power plants that can be used to supply the datacenters with brown energy (since this constrains the maximum size of a datacenter that can be placed at a given location). The outputs of the optimization are the optimal cost, and, at each location d, the required power capacity, the green-energy generation capacity, and the battery storage capacity.

Figure 9.1 provides the functions and constraints that define the problem. Further equation 9.3 defines the cost function to minimize ($TotalCost$), where $placed(d)$ is a boolean representing the placement of a datacenter at location d. Recall that CAP_ind is the CAPEX cost that is independent of the power capacity of the datacenter, CAP_dep is the CAPEX cost that depends on the size of the datacenter, and OP is the OPEX cost of operating the datacenter. All costs assumes 12 years of lifetime for datacenters and solar and wind plants and 4 years for IT equipment. In addition, our modeling assumes that the CAPEX costs already embody the depreciation costs and any interest payments they may incur.

$$TotalCost = \sum_{d \in D} placed[d] \cdot (CAP\_ind(d) + CAP\_dep(d) + OP(d)) \tag{9.3}$$

$TotalCost$ should be minimized under the constraints that follow the equation in the figure. The constraints include power consumption and generation capacity, confidence for generation of green energy, battery capacity [116], and brown power demand. In reality, the availability constraint is more complex than in the figure. In particular, in a network with at least 2 datacenters and S servers, we also ensure that the failure of 1 datacenter will leave $S/2$ servers available to handle the load.

Heuristic solver

This problem can be programmed as a MILP and solve it using any MILP solver. However, some of the constraints make the problem very hard to solve. In particular, confidence for

1. $\forall d \in D, \forall t \in T : capacity(d) \geq migratePow(d,t) + computePow(d,t)$
2. $\forall t \in T : \sum_{d \in D} computePow(d,t) = totalCapacity$
3. $\forall t \in T : P(\frac{\sum_{d \in D} greenPow(d,t)}{\sum_{d \in D} powDemand(d,t)} \geq minGreen) \geq confidence$
4. $\forall d \in D : placed[d] = 0 \Rightarrow capacity(d) = 0$
5. $\forall d \in D, \forall t \in T : powDemand(d,t) + extraGreenPow(d,t) = powAvail(d,t)$
6. $\forall d \in D, \forall t \in T : battLevel(d,t) = battLevel(d,t-1) + battEff \cdot battChargePow(d,t) \cdot t - battDisPow(d,t) \cdot t$
7. $\forall d \in D, \forall t \in T : 0 \leq battLevel(d,t) \leq battCap(d)$
8. $\forall d \in D, \forall t \in T : netLevel(d,t) = netLevel(d,t-1) + netChargePow(d,t) \cdot t - netDisPow(d,t) \cdot t$
9. $\forall d \in D, \forall t \in T : netChargePow(d,t) + battChargePow(d,t) + extraGreenPow(d,t) = solarPow(d,t) + windPow(d,t)$
10. $\forall d \in D, \forall t \in T : brownPow(d,t) \leq nearestPowPlantCap(d) \cdot F$
11. $n = \sum^{d \in D} placed(d) : \sum_{i=0}^{n-1} \binom{n}{i} \cdot i \cdot (1-a)^{n-i} \cdot (a)^i \geq minAvailability$

Figure 9.1: Optimization function and constraints

green-energy generation increases the number of variables and constraints exponentially. For this reason, heuristics may be used to solve the problem.

Here a heuristic is developed, which has two steps: (1) solve a simplified version of the problem using MILP; and (2) evolve this initial solution to obtain a solution for the actual problem. For simple constraints, the optimal solution can be found, otherwise a suboptimal is obtained.

The first step solves the problem for 100% confidence and obtains the optimal result. This is a MILP problem that can be solved using GUROBI [75] in a reasonably short amount of time. If this is a solution for the original problem, the solution is directly returned as final.

In the second step, a local search approach is used to find a solution for the initial problem. It starts from the solution obtained in the previous step, then generates random neighbors and evaluates if they are feasible and their cost. The way to get these neighbors is merging two existing datacenters, changing location of a placed datacenter, or removing a placed DC. It finishes when we get to a certain number of iterations without getting better results. The solving process can be parallelized, so it can try different neighbors at the same time, synchronizing periodically the parallel threads.

A case of study: Placing a datacenter network

Figure 9.2 shows the costs of building a network of datacenters that provides 50MW of computation and at least 50% of the energy comes from green sources. This placement assumes the use of net metering to store the surplus green energy in the grid and use it later. In particular, the obtained placement returns 3 datacenters, 2 in the USA and one in Kenya.

The largest datacenter, with a computation capacity of 19.8MW (∼70000 servers), is placed in New Hampshire. This location has a wind capacity factor of 56.8% (the highest capacity factor in all the evaluated locations). This location is fully powered by a wind farm of 51.4MW and does not use any brown energy (considering the net metered green energy). Note that the land for the wind farm represents the main cost as it requires more than $0.61km^2$. The second datacenter is placed in Cleveland where the wind capacity factor 20.9%. This datacenter has a wind farm of 15.2MW and uses and average of 21.9MW of brown power (76% of the total power).

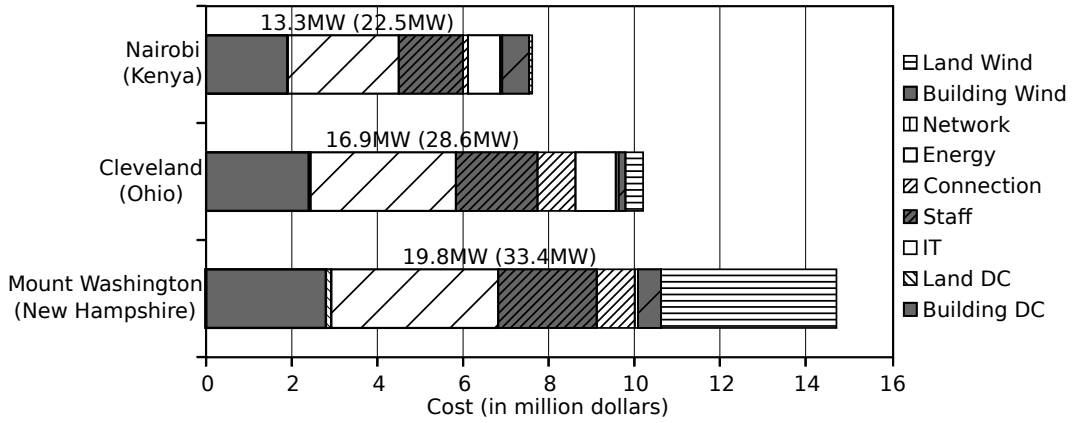


Figure 9.2: Cost of building a 50% green network of DCs with a computation capacity of 50MW

The smallest datacenter is placed in Kenya with a computing capacity of 13.3MW. Even though the wind capacity factor is pretty low (less than 7%), this datacenter is powered by a 60.1MW wind farm. The reason is the land is much cheaper in Kenya than in the other locations. In addition, 64% of the energy comes from brown energy as the electricity is relatively cheap.

9.3 Green datacenter placement tradeoffs

Green energy requirement

Figure 9.3 shows the total cost per month of a network of datacenters, as a function of the percentage of green energy required. Here netmetering is used to "store" surplus green energy into the grid, retrieving it when required. It can be seen that a 100% green network is almost two times more expensive ($\$43.5M/month$) than a network with no green energy requirements ($\$25.8M/month$), due to the green infrastructure requirements.

In the case of a solar only installation, the cost per month is almost three times higher ($\$70.5M/month$), due to the base cost of a solar installation, more than three times more expensive than a wind one ($\$5500/kW + 20m^2/kW$ vs $\$5500/kW + 12m^2/kW$). However, the cost of solar-only is less than two times higher than wind-only because the green farm represents less than the 40% of the total cost. Complementing a wind only location with solar when having net metering does not help much as net metering already adds predictability to wind.

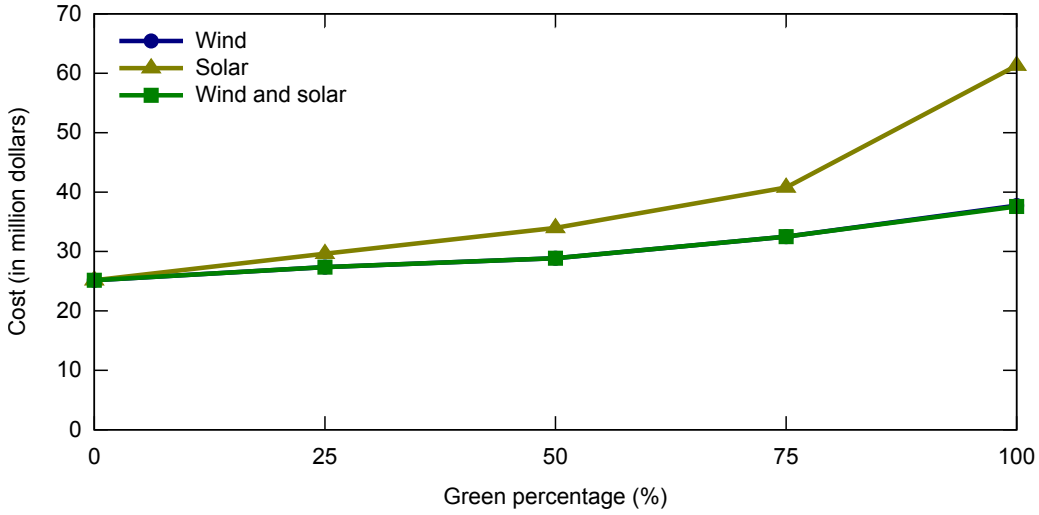
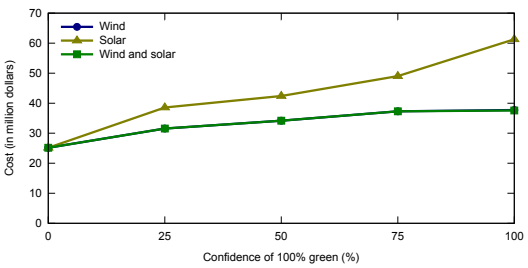


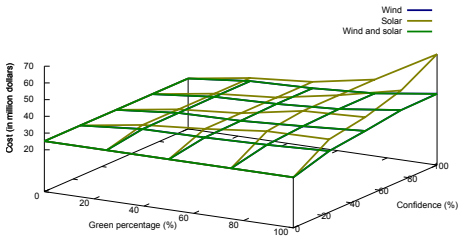
Figure 9.3: Cost of building a network of DCs depending on green % required using net metering

Confidence in green energy

Figure 9.4a shows the cost of confidence on having 100% green energy. Note that having 0% confidence is like having a completely brown network of datacenters and then the cost is much lower. Further, figure 9.4b shows the cost of confidence and the proportion of green. It can be seen that having high confidence and a high green energy percentage is the most expensive. The lower the confidence and the percentage of green energy, the closer to the cost of a completely brown network of datacenters.



(a) Cost of building a network of DCs depending on the confidence of being 100% green using net metering



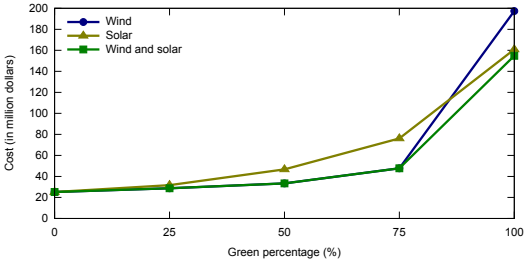
(b) Cost of building a network of datacenters depending on the confidence and the percentage of green using net metering

Figure 9.4: Cost of building a network of DCs depending on confidence using net metering

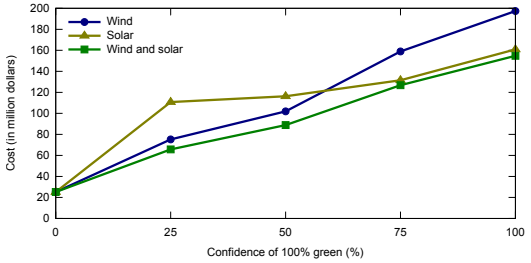
Energy storage and Batteries

As previously seen, the usage of net metering saves cost spikes on placement. Here the effect of having any energy storage are evaluated.

First evaluations are to see the effect of not having any energy storage. Figures 9.5a and 9.5b show that when green energy cannot be stored to deal with unpredictability, the cost spikes from around \$70M/month to more than \$200M/month. When not disposing of storage to handle unpredicted solar/wind blackouts for some periods, overbuilding happens for solar and wind farms and a lot of green energy is wasted during great part of the time. In particular, this effect strongly affects wind as it goes from being 40% (\$40M/month) cheaper than solar to be 14% (\$50/month) more expensive. As wind is more variable and this green energy cannot be stored to use it later, more datacenters are required.



(a) Cost of building a network of datacenters depending on the percentage of green required with no energy storage

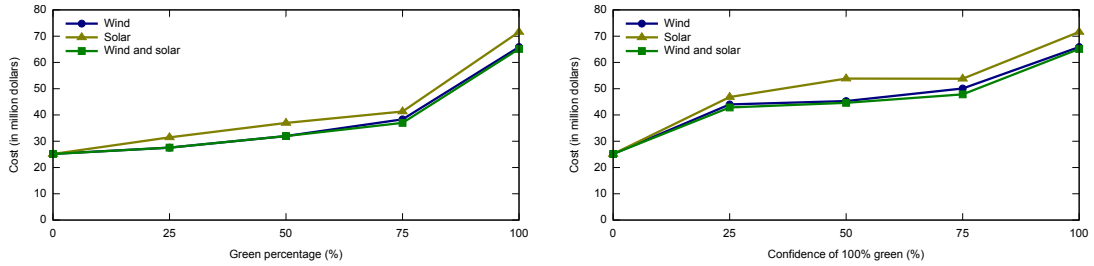


(b) Cost of building a network of datacenters depending on the confidence of being 100% green with no energy storage

Figure 9.5: Cost of building a network of datacenters depending on confidences with no energy storage

Figures 9.6a and 9.6b show the costs when using batteries to store energy instead of net metering. Costs for solar follow the same trends with a slightly higher cost than net metering, because unlike net metering, this battery capacity must be paid for, and this is limited. This

effect is clearly seen in a wind only setup where the total cost is much closer to the solar-only installation. This is because the deployment of 50% more batteries than for solar only to deal with the unpredictabile peaks and valleys of wind generation.



(a) Cost of building a network of datacenters depending on the percentage of green required using batteries

(b) Cost of building a network of datacenters depending on the confidence of being 100% green using batteries

Figure 9.6: Cost of building a network of datacenters depending on the confidence using batteries

Migration impact

Finally, an addition to the placement sizing is to be able to hold the load when migrating, as load must be maintained in origin and destination datacenter. In a conservative assumption where load must not be migrated the oversize is not required, while in a situation 100% is always being moved, some extra sizing is mandatory. When not having energy storage, load is moved trying to follow the renewables; but when having energy storage, this migration overhead is not noticeable as the number of migrations is seen to be very low.

Figure 9.7 shows the costs of a 100% green network of datacenters with no storage when migration needs are from 0% to 100% of the load. As seen, the overhead can increase the cost by almost $30M/month. This cost does not only come from the energy spent to migrate but also by the capital costs. To support higher migration overhead, more datacenters with higher capacities are needed.

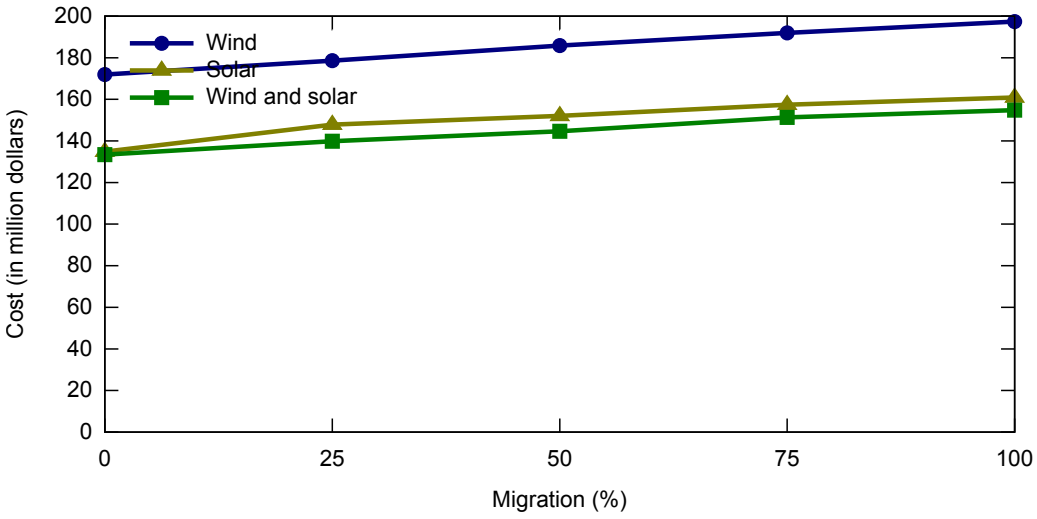


Figure 9.7: Cost of building a 100% green network of datacenters depending on the migration requirements with no energy storage

9.4 Scheduling VMs Among Green Energy Availability

Once found the placement for the green network of datacenters, they must be operated and their load to be distributed following the availability of green energy on each location. The most well known strategy for this goal is the "follow the renewables" (FTR). This approach migrates the load according to the forecast of green energy around the network while considering constraints like migration time, energy spent on migration or the amount of data to migrate.

Scheduling VMs Model and Mathematical Program

The placement model and problem is completed with an implementation of the FTR strategy, on top of a virtualized environment as it allows migrating the workload transparently. In practice it has been implemented for the OpenNebula datacenter virtualization platform. The scheduler periodically calculates the optimal placement for the VMs and migrates them accordingly, using again a mathematical program to model the problem. Table 9.2 shows the parameters and variables for the load distribution problem.

Domain of the Problem		
Locations L	Set of locations to place datacenters	
List of Parameters		
Capacity[L]	Datacenter capacity in L	in Watts
Green[L]	Current green power produced in L	in Watts
PriceBrown[L]	Current cost of brown in L	in $\$/kWh$
PUE[L]	Current PUE in L	in [1,1.69]
List of Variables		
Load[L], ≥ 0	Power load in L	in Watts
PreviousLoad [L]	Load executed in L at previous time slot	in Watts
Brown[L], ≥ 0	Additional brown power required in L	in Watts
NetMeter[L], ≥ 0	Additional brown power required in L	in Watts
Migration[L]	Power load migrated from L	in Watts
MigrOut[L], ≥ 0	Power load migrated out of location L	in Watts

Table 9.2: Summary of parameters and variables for the load placement problem

For every scheduling round, it calculates how much load will be executed in each location, also considering the cost to migrate load (VMs) from a datacenter to another. The cost to run a VM in two different locations for a period of time is in ($Migration$), and $MigrOut$ is the positive component of $Migration$, which makes the problem MILP with one binary variable per location. It is also done under the assumption the green energy available in the near future (i.e., next hour) can be predicted with high accuracy [114, 71]. In addition to these parameters and variables, the problem has the constraints shown in Figure 9.8, also objective cost function in Equation 9.4.

1. $\forall\ l \in L : Load[l] + Migration[l] = PrevLoad[l]$

2. $\forall\ l \in L : Load[l] + MigrOut[l] \leq Capacity[l]$

3. $\forall\ l \in L : (Load[l] + MigrOut[l]) \cdot PUE[l] = Green[l] + Brown[l] + NetMeter[l]$

Figure 9.8: Constraints for the load placement problem

$$Cost = \sum^{l \in L} Brown[l] \cdot PriceBrown[l] \tag{9.4}$$

These constraints are explained as follows: (1) for all locations, the load plus the load migrated (positive or negative) must be equal to the previous load; (2) for all locations, the load plus the

load being migrated cannot exceed the total capacity; and (3) for all locations, the total power required in a datacenter is equal to the produced green energy plus the brown energy (to be minimized) and the netmetering obtained from the netmetering credit (when enabled).

This MILP problem can be solved with, e.g., GUROBI [75] again. Once the scheduler finds the values of $Load[l]$ that minimizes the total cost, it decides which VMs to move. In particular, it orders the datacenters taking into account their $Migration[l]$. A positive $Migration[l]$ implies the datacenter is a "donor" while a negative values means a "receiver". It follows a first fit strategy to migrate VMs from a "donor" to the closest "receiver". To reduce useless migrations, this policy with the green energy prediction is extended to the next time slot, avoiding migrations when both locations have free energy

Experiments on load distribution

In the following experiments, the policy to distribute load according to the green energy availability is evaluated, using one host to emulate one datacenter, and moving VMs across them. All of this assuring that it is possible to migrate VMs working over 750MB of data in less that one hour, as seen in performed experiments migrating VMs between Barcelona (Spain) and Piscataway (New Jersey). So an hour is the time period considered in these load balancing experiments.

Having Net Metering : The first evaluation case of study is a situation with two datacenters, obtained from previous datacenter placement experiments, which are able to provide 50% confidence of being 100% using net metering to store green energy. Table 9.3 describes the characteristics of such datacenters. Note that only two datacenters are needed and they are only powered by wind as it is much cheaper.

Location	Capacity	Solar	Wind
Mount Washington, Vermont	42.2MW	-	28.9MW
Dodge City, Kansas	42.2MW	-	88.6MW

Table 9.3: Datacenter network details for 50% confidence of having 100% green energy using net metering

Figure 9.9 shows the energy profile of the two datacenters during one of the days of the Typical Meteorological Year. As seen, the load is split between the two datacenters. Around the 17th hour of the experiment, the datacenter in Kansas has surplus green energy that stores in the grid (net metering), to be returned again from the grid when it has no enough green energy. At the end of the year, the datacenter achieves the 50% confidence requirement.

Without Net Metering : As having net metering, load migration is not much required. For this reason, here the behavior of another example where we do not have any green energy storage is also shown. Table 9.4 describes the datacenters required to be 100% green with no storage. This placement requires four datacenters powered by a mix of solar and wind.

Location	Capacity	Solar	Wind
La Paz, Bolivia	42.4MW	69.3MW	12.3MW
Andersen, Guam	84.5MW	514.2MW	308.1MW
Harare, Zimbabwe	84.5MW	257.0MW	44.7MW
Acapulco, Mexico	74.3MW	321.2MW	-

Table 9.4: Datacenter network details for 100% green energy without energy storage

Figure 9.10 shows a clear follow the renewables pattern (i.e. follow the sun). It starts hosting the load in Guam, then it moves to Zimbabwe, then Bolivia and Mexico, and finally, it goes back to Guam. As seen in the aggregated figure (bottom-left), the green energy is clearly overbuild to

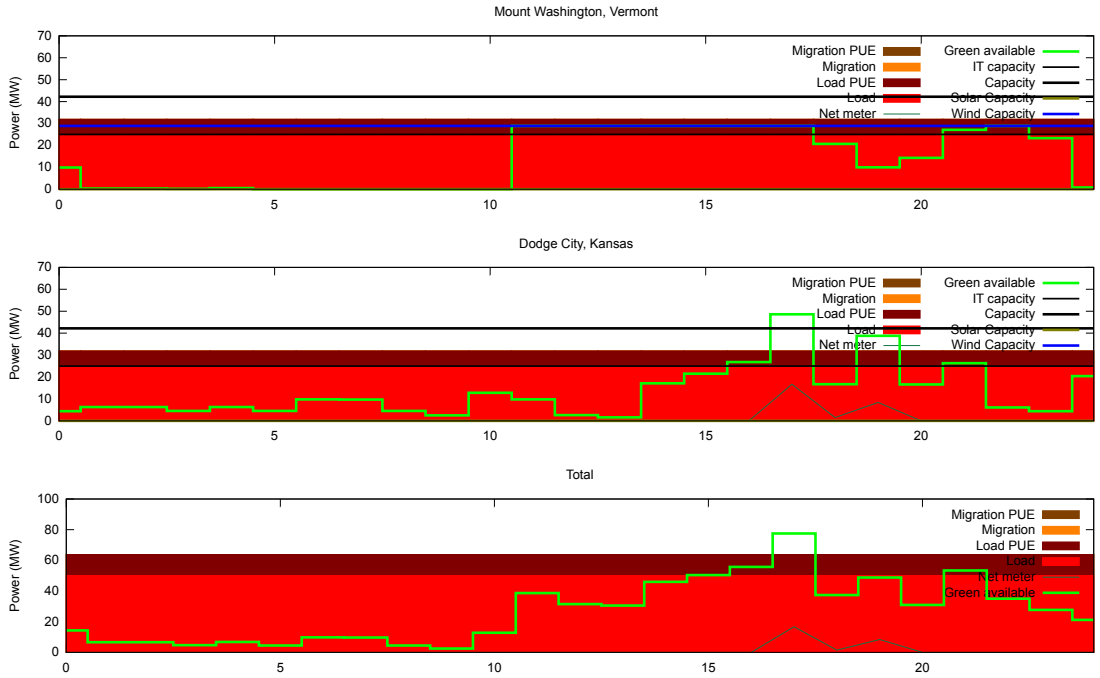


Figure 9.9: Load distribution to achieve 50% confidence of having 100% green energy using net metering

support days with low green energy availability. The figure also shows the overhead to migrate load between locations.

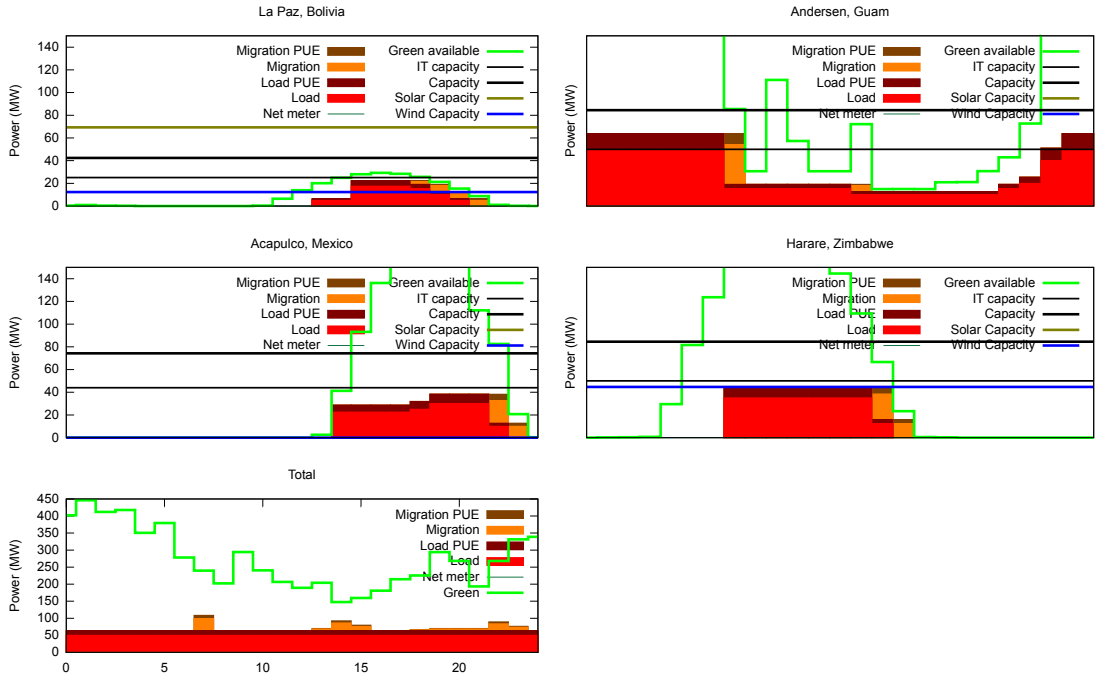


Figure 9.10: Load distribution to achieve 100% green energy without energy storage

9.5 Conclusions for Green Placement of DataCenters

In this new approach, the problem studied is the costs of building a network of datacenters fully or partially powered by green sources of energy. These green datacenters have solar or wind farms attached to harvest green energy and they are distributed along the world to exploit the variability of green sources. An analytical framework, using mathematical programming, MILP and local search heuristics, is proposed to place datacenters, minimizing the economic costs while maximizing the green power generation obtainment.

The mathematical formulation optimizes for the capacities and sizes for power generation infrastructure for each placement selected location. In particular, the main capital and operational costs for building an maintaining these datacenters in each geographic location are considered. To proceed with the testing and experimentation for this approach, the climatic and weather factors of more than 3000 locations is collected, to calculate the generation of solar and wind energy.

From this study we conclude that wind energy is cheaper for green datacenters. In particular, using energy storage (both net metering or batteries) it is the cheapest way to green datacenters. However, if high confidence in being completely green with no energy storage is required, the use of solar energy becomes cheaper.

Finally, based in the results and observations from the placement study, a scheduler using the "follow the renewables" policy is implemented, using as base middleware OpenNebula, to test the operation of such network of placed datacenters. This scheduler distributes the load across the datacenter "following the renewables", that is, depending on the energy availability for each location.

The work presented in this chapter is being submitted to a conference on the energy efficiency area on Fall 2013.

Chapter 10

Conclusions

10.1 Main Contributions

First of all we started with preliminar experiences with machine learning and web-services modeling, as shown in Chapter 3, to allow us to sketch the goals of this thesis and seek the way this work should visit in the so broad field of distributed systems, autonomic computing and data mining. Also these preliminary works provided experience on the field with novel approaches on self-management, like web-service user admission and balancing, around the concept of starting-up and shutting-down hosting machines on a economic saving policy, or denial of service protection and reaction policies. The novelty of the presented techniques is the use of machine learning models and predictions, to automate the modeling and adjusting process of the decision making mechanisms. The first works focus on applying machine learning on user modeling and behavior prediction, and user admission macropolicies. Given a set of web-service hosting machines that must be set up or shut down for cost saving, we decide which servers are used to host clients and which clients enter in the on-line hosts, all according to a benefit/cost function. Also other preliminary works focus on self-protection, introducing a novel method for distributed systems to defend themselves against overwhelming distributed denial of systems. A distributed data collection mechanism is used to recognize abnormal situations on traffic towards a web-service, and a classification system determines if it is a thread or not, spreading the required commands over the routers and firewalls. Further, supporting the study of machine learning for decision making is topic for another preliminary work presented here, which studies the application of machine learning over memory behavior attempting to manage self-healing schedules on web-services.

Virtualization has opened new oportunities for resource management in datacenters. The virtualization technology encapsulates Web-based applications or HPC jobs in virtual machines and see them as a single entity which can be managed in an easier and efficient way. It is used as a tool for consolidating underused nodes and save power, but it also allows new solutions to well-known challenges such as resource heterogeneity management. One of the works proposed in this thesis, as seen in Chapter 4, exposes a way to model and manage a virtualized data center which mainly focuses on the allocation of VMs in data center nodes according to multiple facets while optimizing the provider's profit. We consider energy efficiency, virtualization overheads, and SLA violation penalties, while providing the ability to outsource resources to external providers.

Lately, energy-related costs have become an important economical and social factor towards IT infrastructures and datacenters and companies. Research community is also challenged to find better and more efficient power-aware management strategies, as there is a big gap to be covered in this area yet. In Chapter 5 we focused on energy-efficiency datacenters, introducing an intelligent consolidation methodology using different techniques, including turning on/off machines, power-aware consolidation algorithms, also applying machine learning techniques to deal with uncertain information while maximizing performance. Based on our previous experiences, we applied machine learning techniques in these scenarios for modeling system behaviors in order to predict power consumption levels, CPU loads and SLA timings, improving scheduling deci-

sions. These techniques are evaluated covering the whole control cycle of a real scenario, with the EEF-Simulator, oriented to energy efficiency, with representative heterogeneous workloads, and measuring the quality of the results according to SLA defined goals. The results obtained indicate that our approach is close to the optimal placement and behaves better that static or ad-hoc models when the level of uncertainty increases, thanks to the machine learning component.

Knowing the principal factors on datacenter hosting businesses, like economic profit, customer satisfaction and operational costs (like power consumption) for its correct performance, we show in Chapter 6 how to represent the problem of scheduling a set of web-services to hosts in a datacenter, relying on the advantages of the virtualization and consolidation techniques, as a mathematical program represented in an integer linear model. We study the behavior of the model and the trade-offs between those factors, and then we apply it in a framework for autonomic scheduling of tasks and web-services on cloud environments, optimizing the profit taking into account revenue for task execution minus penalties for service-level agreement violations, minus power consumption cost. We combine here the previous seen technologies of consolidation and virtualization, mathematical optimization methods, and finally applying the machine learning methods to complement unknown or unavailable data. We prove the concept using an exact solver for mixed integer linear programs, but since the problem is NP-hard and so untreatable, we show that approximate algorithm solvers provide valid alternatives for finding good enough schedules, close to the optimals. We apply here machine learning to estimate unknown parameters, avoiding to model by hand web-service behaviors and requirements, as a first approach for joining the learnability and predictability with an ordered mathematical problem statement.

Managing a cloud and optimizing its performance on a moment-by-moment basis is not easy given as the amount and diversity of elements involved. This makes the Cloud and its architecture a new scenario for data mining and machine learning to discover information, and use it to improve datacenter management with modeling and prediction. In this thesis we show how to model the basic cloud resources using machine learning, predicting resource requirements from context information, like amount of load and clients, and also predicting the quality of service from resource planning, in order to feed schedulers and decision makers. As shown in Chapter 7, these learning techniques, combined to classical heuristics and approximate algorithms, help to improve the performance and profitability of a data-center running virtualized web-services. We model the datacenter main resources (CPU, Memory and Network/IO), the quality of service (represented as response time or SLA fulfillment) and workloads (incoming streams of requests) from observing past executions of this same system. Then these models help scheduling algorithms to make better decisions about job and resource allocation, aiming for a balance between throughput, quality of service, and power consumption. From here on, the approaches are tested not only with real data and real web-services but also on a real execution on a reduced scale environment, reproducing each set of requests given the workload upon the web-service.

The Cloud does not only rely on single datacenter systems, but also services are distributed across the world to provide localized service, reducing latencies among services and clients, or taking advantage of energy costs and obtainance. So in Chapter 8 we expand the presented model towards a multi-datacenter scenario, where infrastructures are distributed along the planet, and people and enterprises pay for resources to offer their web-services to worldwide clients. Intelligent management is required to automate and manage these cloud multi-datacenter infrastructures, as the amount of resources and data to manage exceeds the capacities of human operators, taking advantage of local energy prices, but assuring quality of service and also involving the proximity to clients and customers. We expand the mathematical model to describe the scheduling problem across a multi-datacenter system, while applying our learned models. After running the system on real DC infrastructures we see that the model drives web-services to the best locations given quality of service, energy consumption, and client proximity.

Finally, we present in Chapter 9 an approach for building datacenter networks, searching for places where these datacenters can be entirely powered by "green energy", being as much "brown energy" independent as possible. We quantify the costs for high-performance computing cloud service providers, characterizing areas around the world as potential locations for green datacenters, and illustrating the location selection tradeoffs by quantifying the minimum cost of achieving different amounts of renewable power at different levels of confidence. After that, we design and implement a cloud middleware capable of migrating virtual machines across the

green datacenters to follow the availability of renewable energy. Among other interesting results, we demonstrate that green cloud services can achieve significant reductions in carbon dioxide emissions at a relatively low cost by intelligently placing their green datacenters around the world.

10.2 Topics for Further Research

Energy Efficiency Techniques

This thesis, focusing on energy efficiency, has explored techniques like consolidation, using the advantage provided by virtualization. Thanks to virtualization we could take advantage of being able to dimension the job and web-service containers, and even more important, to move them across physical machines, placing jobs and services in the closest locations. There are also other techniques, like dynamic voltage/frequency scaling (DVFS), that offer other kind of actions to save energy alternatively to turn on/off resources, and they can be introduced in the presented models and strategies, as extra operations that may be performed, complementing consolidation. E.g. instead of using only operations like "migrate a VM from location A to location B" and "switch power on host C", operations like "change the frequency of host A" or "change the operational mode of host A" can be added to the problem. The mathematical problem would add a new degree of freedom to the scheduling solver, as it could play with a $Hosts \times VMs$ placement scheduler and also a $Hosts \times WorkingMode$ planning.

Non-Economical "Green" Strategies

Here we based our approaches in an economic model, based on the revenue per SLA fulfillment against power costs. But there are other approaches and green computing proposals more oriented toward optimizing energetic utility on resources or keep in mind the importance of non-economical factors, like social and environment elements. Combining economic models using SLA enforcement driving functions with these new factors, pseudo-economic models can result. Research would take into account how to merge directly these two kind of elements, as at this time the only way to compare green energy and brown is in price, and when green energy becomes too much expensive (because of lack of availability or high construction costs) it is directly discarded. The results would find a set of trade-offs between the profitability of the datacenter business and "green" performance ratio of datacenters, allowing to involve more factors related to green computing, like the source of energy powering the datacenters, not represented by economic models.

Learning HPC Job Behaviors

When learning relations on load versus jobs, in this thesis we focused mainly on web-services. Other kind of jobs like HPC jobs, despite having a load versus deadline relation apparently easy to find, just by test and observation, can be also subject of study. The SLA objects and measures to be taken would be different to web-services, as the most usual QoS measure for HPC jobs is finishing it on time. Despite that, modeling these jobs, in order to find the proper combination of jobs $\times$ computing machines, could improve the utilization of datacenter resources while preserving the QoS of these kind of jobs. For these jobs, an inconvenient arises when modeling them, as the kind of HPC jobs are more sparse than web-services, that all share a limited set of web platforms with similar behavior. Techniques like clustering would help to classify these jobs and take advantage of their characteristics, also techniques like reinforcement learning would help to schedule them properly by trial and observation.

Learning Load Behaviors

At this time, all approaches here presented are based on the fact that we can predict or we know at short time-scale the workload. When managing web-services, we have so far assumed that the load in the near future will be very similar to the load currently being received. But for most webservices load experiences important, even drastic, fluctuations over time. A new research

line would be to learn to predict the temporal evolution of the load and clients/users behaviors over time and over each kind of web-service. For this kind of research methods oriented towards learning time series could be used. Treating information about current load and characteristics of the current load as a time serie, as this information will only be seen once and discarded after its usage, we can learn to predict the future load and its characteristics. In many web-services we can see that load presents periodical patterns (hourly, daily, weekly, monthly, yearly ...), so if we are able to discover them we will be able to configure and optimize the system better by constantly adapting it with better accuracy.

Learning About Resources and Services

At the beginning of the thesis, the principal resource involved in job management was CPU. During this research, Memory and Network IO has been added. But other resources are still to be introduced in the model, i.e. disks and storage systems. In a web-service environment, other services like file systems, application services and database services can be understood as "resources". At this time each VM had its own FS, app-server and DB-server, and we treated it as part of the CPU/MEM/IO process, as it is difficult to discriminate it inside a VM. But on a system where these services are distributed among dedicated systems, we can start monitoring them and treating them as shared resources. Many web-services can try to access an app-server, a database or a file in a shared disk filesystem.

These non-physical-component resources must be added to the mathematical model, new resource predictors should be learned from load and context. Even an availability function should be learned from aggregated load and resource, the same way SLA predictors do, to determine the QoS this resource is offering when it becomes overloaded by different web-service requests.

Improvements on Model Selection

During all this thesis we held as hypotesis that machine learned models could achieve similar results as handcrafted models, without much expert intervention. However, here we required "expert intervention" at each learning process, when selecting and validating models. We still have advantage as once the expert has found the adequate machine learning algorithm, it can be applied for the same kind of systems and adapt to each one specifically without so much effort. Also we have the advantage of human learning over the machine learning models. A good research topic, from the machine learning side, would be to find a method able to reduce the expert intervention even when creating those learned models. In fact, building more autonomous, self-adjusting learning systems is one of the most active research lines in machine learning these days.

On-Line Model Learning

At this time all the learning process is done off-line, and this means that all training and validation data must be obtained before the model is created, also when retraining the model, data must be collected and labeled before having the model updated. Another improvement to be done, in order to automate even more the learning process, is to realize this learning on-line. An on-line methodology would allow, first of all, to learn models from the first instant we receive data from the system. The period of supervision would be reduced, as the labeling of the input data should be done automatically to provide some sense to the approach.

Also the on-line methods have the advantage that, when they get deprecated, there is no need to "stop" the system and collect data again to retrain the model. There are ways to detect when the model became deprecated, flush the old learned model and create a new one, or update the old one automatically. All in all without too much supervision. Data stream mining methods can be applied here, when constant input information can fit into our model or bring new behaviors to be learned. E.g. when the web-service is changing and updating its contents periodically, the users and clients may change their behavior. So the workload would be changing also periodically, and learned models should be able to detect the changes in trends, discard old information and old models, and create new ones.

As seen in the hardware/software infrastructure and the information flow (Figures 2.2 and 4.2), here a component whould be checking the monitors for changes on load trends or changes on resource behaviors, and try to adapt the prediction models by updating them, or in worst case re-training them. The system would start with an empty model or a generic model, and it would be adjusted with observed and checked information from monitors. Notice that not all ML algorithms can be updated (but all can be re-trained), so depending on how much we require the models to be up-to-date we should choose wisely which algorithms we use for regression and classification.

Further, for models that do not require supervision, reinforcement learning (RL) could be used then. RL methods keep updating the models checking always each decision made with its consequent observations of the system. So approaches different from predicting resources from load, like e.g. QoS effects from a given schedule, could be implemented using RL and trained online.

Better Heuristics and Solvers

Due to the intractability of solving the mathematical model in an exhaustive way, we had to fall back to heuristic and approximate algorithms. Treating the problem as a Generall Allocation Problem (a scheduling problem), we could apply classical algorithms like first-fitting and best-fitting. But when adding more requirements and dimensions to the problem (like adding DVFS or different system configurations), the problem will turn into a more complex one, and then other algorithms should be researched.

During the modeling of datacenters and study of the MILP exhaustive solving, we attempted to apply alternatives like Lagrange relaxation techniques. These techniques consist in turn the problem in a more simple one by moving hard constraints to the maximization function with a λ factor to be enforced when the constraint is violated, and to be relaxed otherwise. This method is iterative and returns a solution after solutions converge. It does not reduce complexity but in some situations finds acceptable solutions faster than MILP linear relaxation + branch and bound methods. We abandoned for the moment this approach because the problem converted to Lagrange format in the standard way became intractable in memory. However, further research could either make it feasible or at least usable to find fast solutions to subparts of the scheduling problem.

Managing Multi-DC Networks

In this thesis we made a first approach towards the management of a network of datacenters, all driven by local energy costs, proximity to web-service clients, and preserving the quality of service. But there is a lot of work to be done in this topic. First of all, more experiments around the topic can be done, studying more details like a bigger fluctuation of energy prices and their impact on the scheduling policy; like sudden changes in the world workload provoking huge VM migrations among DCs, and then study the migration plan; or like mixing different workloads in the network, treating the network as a new resource, and the possibility of it becoming a bottleneck.

After this, the production of green energy seen in the last chapter of the thesis could be fully integrated in the energy model for multi-DCs management. Turn the multi-DC network into a "green" multi-DC system, where web-services are reallocated according to client proximity, resource availability, and green energy availability.

10.3 List of Publications

The contributions of this thesis appear in the following publications:

Journals and Book Chapters

[31] **Josep Ll. Berral**, Iñigo Goiri, Ramon Nou, Ferran Julià, Josep O. Fitó, Jordi Guitart, Ricard Gavaldà, Jordi Torres, "Toward Energy-Aware Scheduling Using Machine Learning"

Energy-Efficient Distributed Computing Systems, Chapter 8. Editors: Albert Y. Zomaya, Young Choon Lee, July 2012 .ISBN 9780470908754

[66] Iñigo Goiri, **Josep Ll. Berral**, J. Oriol Fitó, Ferran Julià, Ramon Nou, Jordi Guitart, Ricard Gavaldà, Jordi Torres, "Energy-efficient and multifaceted resource management for profit-driven virtualized data centers" Future Generation Computer Systems.5 - 28,pp. 718-731.05/2012. ISSN 0167-739X

Publications in International Conferences and Workshops

[X] **Josep Ll. Berral**, Íñigo Goiri, Thu Nguyen, Ricard Gavaldà, Jordi Torres, Ricardo Bianchini. "Building Low-Cost Green DataCenters". To be submitted to a conference next Fall 2013.

[29] **Josep Ll. Berral**, Ricard Gavaldà, Jordi Torres. "Power-aware Multi-DataCenter Management using Machine Learning". The 2nd International Workshop on Power-aware Algorithms, Systems, and Architectures (PASA-2013). Lyon, France. October 1, 2013.

[27] **Josep Ll. Berral**, Ricard Gavaldà, Jordi Torres. "Empowering Automatic Data-Center Management with Machine Learning". The 28th ACM Symposium on Applied Computing (SAC 2013) Data-Mining track, Coimbra, Portugal, March 18-22 2013.

[25] **Josep Ll. Berral**, Ricard Gavaldà, Jordi Torres. "Adaptive Scheduling on Power-Aware Managed Data-Centers using Machine Learning". The IEEE International Conference on GRID Computing (GRID 2011), Lyon, France, September 22-23 2011.

[30] **Josep Ll. Berral**, Iñigo Goiri, Ramon Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, Jordi Torres. "Towards energy-aware scheduling in data centers using machine learning". 1st ACM/SIGCOM Intl. Conf. on Energy-Efficient Computing and Networking (eEnergy 2010), Passau, Germany, April 13-15 2010.

Preliminary and Collaboration Publications in Intl. Confs and Workshops

[3] Javier Alonso, **Josep Ll. Berral**, Ricard Gavaldà and Jordi Torres. "Adaptive on-line software aging prediction based on Machine Learning". The 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010), Chicago (IL), USA, June 28 - July 1, 2010.

[4] Javier Alonso, **Josep Ll. Berral**, Ricard Gavaldà, Jordi Torres. "J2EE Instrumentation for software aging root cause application component determination with AspectJ". 15th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS 2010), Atlanta (GA), USA, April 19-23 2010.

[67] Iñigo Goiri, J. Oriol Fitó, Ferran Julià, Ramón Nou, **Josep Ll. Berral**, Jordi Guitart, and Jordi Torres. "Multifaceted Resource Management for Dealing with Heterogeneous Workloads in Virtualized Data Centers ". The 11th ACM/IEEE International Conference on Grid Computing (GRID 2010), Brussels, Belgium, October 25-29, 2010.

[70] Iñigo Goiri, Ferran Julià , Ramón Nou, **Josep Ll. Berral**, Jordi Guitart and Jordi Torres. "Energy-aware Scheduling in Virtualized Datacenters". IEEE International Conference on Cluster Computing (CLUSTER 2010), Heraclion (Creta), Greece, September 20 - 24, 2010.

[133] Nicolas Poggi, Toni Moreno, **Josep Ll. Berral**, Ricard Gavaldà, Jordi Torres, "Self-Adaptive Utility-Based Web Session Management." Computer Networks.53 - 10,pp. 1712 - 1721.07/2009 .ISSN 1389-1286

[32] **Josep Ll. Berral**, Nicolas Poggi, Javier Alonso, Ricard Gavaldà, Jordi Torres, Manish Parashar. "Adaptive Distributed Mechanism Against Flooding Network Attacks Based on Ma-

chine Learning." 1st ACM Workshop on Artificial Intelligence on Security (AISec 2008), Alexandria (VA), USA, October 27th 2008.

[156] Jordi Torres, David Carrera, Vicenç Beltran, Nicolas Poggi, **Josep Ll. Berral**, Ricard Gavaldà, Eduard Ayguadé, Toni Moreno, Jordi Guitar, "Tailoring resources: Energy efficient consolidation strategy goes beyond virtualization." The International Conference on Autonomic Computing (ICAC 2008), Chicago (IL), USA, 2 to 6 of July, 2008, IEEE International Conference on Autonomic Computing.

[135] Nicolas Poggi, **Josep Ll. Berral**, Toni Moreno, Ricard Gavaldà, Jordi Torres, "Automatic Detection and Banning of Content Stealing Bots for E-commerce." Workshop on Machine Learning in Adversarial Environments for Computer Security, British Columbia, Canada, December the 8th, 2007, The Neural Information Processing Systems (NIPS 2007) Foundation (Proceedings of the NIPS Machine Learning in Adversarial Environments for Computer Security 2007).

[109] Toni Moreno, Nicolas Poggi, **Josep Ll. Berral**, Ricard Gavaldà and Jordi Torres, "Policy-based autonomous bidding for overload management in eCommerce websites." Group Decision and Negotiation Meeting - GDN Section of INFORMS, Montreal, Canada, 14-1 of May, 2007, Group Decision and Negotiation, INFORMS (Proceedings of the Group Decision and Negotiation 2007, pages 162-166).

[131] Nicolas Poggi, Toni Moreno, **Josep Ll. Berral**, Ricard Gavaldà and Jordi Torres, "Web Customer Modeling for Automated Session Prioritization on High Traffic Sites." 11th International Conference on User Modeling (UM 2007), Corfu, Grece, 25-29 of June, 2007. User Modeling Inc. (Lecture Notes in Computer Science, Volumen 4511 páginas 450-454, Springer).

Technical Reports

[28] **Josep Ll. Berral**, Ricard Gavaldà, Jordi Torres. "Modeling cloud resources using Machine Learning". Research Report number: UPC-LSI-13-7-R, March 2013.

[26] **Josep Ll. Berral**, Ricard Gavaldà, Jordi Torres. "'Living In Barcelona' Li-BCN Workload 2010" Research Report number: UPC-LSI-11-1-T, January 2011.

[24] **Josep Ll. Berral**, Ricard Gavaldà, Jordi Torres. "An Integer Linear Programming Representation for DataCenter Power-Aware Management" Research Report number: UPC-LSI-10-21-R, November 2010.

[87] Ferran Julià, Jordi Roldan, Ramon Nou, Oriol Fitó, Alex Vaqué, Iñigo Goiri, **Josep Ll. Berral**. "EEFSim: Energy Efficency Simulator" Research Report number: UPC-DAC-RR-CAP-2010-15, June 2010.

Bibliography

[1] Google app engine. http://code.google.com/appengine.

[2] Douglas Alger. Choosing an optimal location for your data center. In *InformIT*, 2006.

[3] Javier Alonso, Jordi Torres, Josep Ll. Berral, and Ricard Gavaldà. Adaptive on-line software aging prediction based on machine learning. In *IEEE/IFIP Intl. Conf. on Dependable Systems and Networks (DSN 2010)*, 2010.

[4] Javier Alonso, Jordi Torres, Josep Ll. Berral, and Ricard Gavaldà. J2ee instrumentation for software aging root cause application component determination with aspectj. In *IPDPS Workshops*, pages 1–8. IEEE, 2010.

[5] Javier Alonso, Jordi Torres, and Ricard Gavaldà. Predicting web server crashes: A case study in comparing prediction algorithms. In *Proceedings of the 2009 Fifth International Conference on Autonomic and Autonomous Systems*, ICAS 2009, pages 264–269, Washington, DC, USA, 2009. IEEE Computer Society.

[6] Javier Alonso, Jordi Torres, Luis Moura Silva, Rean Griffith, and Gail Kaiser. Towards self-adaptable monitoring framework for self-healing. Technical Report TR-0150, Institute on Architectural issues: scalability, dependability, adaptability, CoreGRID - Network of Excellence, July 2008.

[7] Amazon DirectConnect (Jan.2013). http://www.amazon.com/DirectConnect/.

[8] Amazon WebServices (Jan.2013). http://aws.amazon.com/.

[9] AMD. Pacifica x86 virtualization. http://enterprise.amd.com/us-en/AMD-Business/Business-Solutions/Consolidation/Virtualization.aspx.

[10] Artur Andrzejak, Sven Graupner, and Stefan Plantikow. Predicting resource demand in dynamic utility computing environments. In *Intl. Conf. on Autonomic and Autonomous Systems (ICAS '06)*, 2006.

[11] Artur Andrzejak and Luis Silva. Using machine learning for non-intrusive modeling and prediction of software aging. In *IN: IEEE/IFIP NETWORK OPERATIONS & MANAGEMENT SYMPOSIUM (NOMS 2008)*, pages 7–11, 2008.

[12] Paolo Anedda, Simone Leo, Simone Manca, Massimo Gaggero, and Gianluigi Zanetti. Suspending, migrating and resuming HPC virtual clusters. *Future Generation Computer Systems*, 26(8):1063–1072, 2010.

[13] Apache Foundation, 2013. `http://www.apache.org`.

[14] Apache tomcat server, 2013. `http://tomcat.apache.org`.

[15] Karen Appleby, Sameh Fakhouri, Liana Fong, Germán Goldszmidt, Michael Kalantar, Srirama Krishnakumar, Donald Pazel, John Pershing, and Benny Rochwerger. Oceano-SLA based management of a computing utility. In *7th IFIP/IEEE International Symposium on Integrated Network Management*, volume 5. Citeseer, 2001.

[16] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[17] Ask.com. `http://www.ask.com`.

[18] Atrapalo web travel agency, 2013. `http://www.atrapalo.com/`.

[19] Ramamurthy Badrinath, R. Krishnakumar, and R.K.P. Rajan. Virtualization aware Job Schedulers for Checkpoint-restart. In *Proceedings of the 13th International Conference on Parallel and Distributed Systems (ICPADS 2007), Hsinchu, Taiwan, December 5-7*, volume 2, pages 1–7, 2007.

[20] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using magpie for request extraction and workload modelling. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 18–18, Berkeley, CA, USA, 2004. USENIX Association.

[21] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[22] Luiz Barroso and Urs Hölzle. The Case for Energy-Proportional Computing. *Computer*, 40(12):33–37, 2007.

[23] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[24] Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. An Integer Linear Programming Representation for DataCenter Power-Aware Management, 2010. http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=1096.

[25] Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. Adaptive Scheduling on Power-Aware Managed Data-Centers using Machine Learning. In *12th IEEE International Conference on Grid Computing (GRID 2011)*, 2011.

[26] Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. Li-BCN Workload 2010, 2011. http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=1099.

[27] Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. Empowering Automatic Data-Center Management with Machine Learning. In *28th ACM Symposium on Applied Computing (SAC'13)*, 2013.

[28] Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. Modeling cloud resources using Machine Learning, 2013. http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=10XX.

[29] Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. Power-aware Multi-DataCenter Management using Machine Learning. In *2nd International Workshop on Power-aware Algorithms, Services and Architectures (PASA'13)*, 2013.

[30] Josep Ll. Berral, Íñigo Goiri, Ramon Nou, Ferran Julià, Jordi Guitart, Ricard Gavalda, and Jordi Torres. Towards energy-aware scheduling in data centers using machine learning. In *1st International Conference on Energy-Efficient Computing and Networking (eEnergy'10)*, pages 215–224, 2010.

[31] Josep Ll. Berral, Íñigo Goiri, Ramon Nou, Ferran Julià, Josep O. Fitó, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. *Toward Energy-Aware Scheduling Using Machine Learning.* Wiley Series on Parallel and Distributed Computing. Wiley, 2012.

[32] Josep Ll. Berral, Nicolas Poggi, Javier Alonso, Ricard Gavaldà, Jordi Torres, and Manish Parashar. Adaptive distributed mechanism against flooding network attacks based on machine learning. In *AISec '08: Proceedings of the 1st ACM workshop on Workshop on AISec*, pages 43–50, New York, NY, USA, 2008. ACM.

[33] Ricardo Bianchini and Ram Rajaniony. Power and Energy Management for Server Systems. *IEEE Computer, Special issue on Internet data centers*, 37(11):68–76, 2004.

[34] Damien Borgetto, Henri Casanova, Georges Da Costa, and Jean-Marc Pierson. Energy-aware service allocation. *Future Gener. Comput. Syst.*, 28(5):769–779, May 2012.

[35] Rajkumar Buyya, David Abramson, and Jonathan Giddy. An economy grid architecture for service-oriented grid computing, 2001.

[36] Rajkumar Buyya, David Abramson, and Srikumar Venugopal. The grid economy. *Proceedings of the IEEE*, 93(3):698–714, 2005.

[37] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Department of Computer Science and Software Engineering (CSSE), The University of Melbourne, Australia. He*, pages 10–1016, 2008.

[38] George Candea, Aaron B. Brown, Armando Fox, and David Patterson. Recovery-oriented computing: Building multitier dependability. *Computer*, 37(11):60–67, 2004.

[39] George Candea, George C, Emre Kiciman, Steve Zhang, Armando Fox, Pedram Keyani, and O Fox. Jagr: An autonomous self-recovering application server, 2003.

[40] David Carrera, Malgorzata Steinder, Ian Whalley, Jordi Torres, and Eduard Ayguadé. Enabling resource sharing between transactional and batch workloads using dynamic application placement. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 203–222, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[41] Karen J. Cassidy, Kenny C. Gross, and Amir Malekpour. Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers. In *2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings*, pages 478–482. IEEE Computer Society, 2002.

[42] Sumir Chandra, Shweta Sinha, Manish Parashar, Yeliang Zhang, Jingmei Yank, and Salim Hariri. Adaptive runtime management of samr applications, 2002.

[43] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, and Amin M. Vahdat. Managing energy and server resources in hosting centers. In *18th ACM Symposium on Operating System Principles (SOSP)*, 2001.

[44] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing Energy and Server Resources in Hosting Centers. *ACM SIGOPS Operating Systems Review*, 35(5):103–116, 2001.

[45] Yiyu Chen, Amitayu Das, Wubi Qin, Anand Sivasubramaniam, Qian Wang, and Natarajan Gautam. Managing Server Energy and Operational Costs in Hosting Centers. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):303–314, 2005.

[46] David M. Chess, Charles Palmer, and Steve R. White. Security in an autonomic computing environment. *IBM Syst. J.*, 42(1):107–118, 2003.

[47] Byung-Gon Chun, Gianlucca Iannaccone, Giuseppe Iannaccone, Randy Katz, Gunho Lee, and Luca Niccolini. An energy case for hybrid datacenters. *ACM SIGOPS Operating Systems Review*, 44(1):76–80, 2010.

[48] Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons, and Jeffrey S. Chase. Correlating instrumentation data to system states: a building block for automated diagnosis and control. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[49] Toni Cortes, Carsten Franke, Yvon Jégou, Thilo Kielmann, Domenico Laforenza, Brian Matthews, Christine Morin, Luis Pablo Prieto, and Alexander Reinefeld. XtreemOS: a Vision for a Grid Operating System, 2008.

[50] Toni Cortes and Ramon Nou. AEM prototype, D3.3.6, XtreemOS deliverable, 2008.

[51] Georges Da Costa, Marcos Dias de Assunção, Jean-Patrick Gelas, Yiannis Georgiou, Laurent Lefèvre, Anne-Cécile Orgerie, Jean-Marc Pierson, Olivier Richard, and Amal Sayah. Multi-facet approach to reduce energy consumption in clouds and grids: the green-net framework. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 95–104, New York, NY, USA, 2010. ACM.

[52] George B. Dantzig and Mukund N. Thapa. *Linear programming 1: introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[53] Rajarshi Das, Gerald Tesauro, Jeffrey O. Kephart, David W. Levine, Charles Lefurgy, and Hoi Chan. Autonomic multi-agent management of power and performance in data centers, 2008.

[54] Data Center Knowledge. Apple Plans 20MW of Solar Power for iDataCenter, 2012. "http://www.datacenterknowledge.com/archives/2012/02/20/apple-plans-20mw-of-solar-power-for-idatacenter/".

[55] Data Center Knowledge. Data Centers Scale Up Their Solar Power, 2012. "http://www.datacenterknowledge.com/archives/2012/05/14/data-centers-scale-up-their-solarpower/".

[56] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: the montage example. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[57] Gaurav Dhiman. Dynamic power management using machine learning. In *IEEE/ACM Intl. Conf. on Computer-Aided Design 2006*, 2006.

[58] Tadashi Dohi, Katerina Goseva-popstojanova, and Kishor S. Trivedi. Analysis of software cost models with rejuvenation. In *Proc. of the IEEE Intl. Symp. on High Assurance Systems Engineering, HASE-2000, November 2000. Statistical Non-Parametric Algorithms to Estimate the Optimal Software Rejuvenation*, pages 25–34, 2000.

[59] Elmootazbellah N. Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. *Lecture notes in computer science*, pages 179–196, 2003.

[60] Europe's energy portal. http://www.energy.eu.

[61] David Filani, Jackson He, Sam Gao, M. Rajappa, A. Kumar, R. Shah, and R. Nagappan. Dynamic Data Center Power Management: Trends, Issues and Solutions. *Intel Technology Journal*, 2008.

[62] J.Oriol Fitó, Íñigo Goiri, and Jordi Guitart. SLA-driven Elastic Cloud Hosting Provider. In *Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP'10)*, pages 111–118, 2010.

[63] Ian Foster. What is the grid? - a three point checklist. *GRIDtoday*, 1(6), July 2002.

[64] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid - enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15:2001, 2001.

[65] GNU. Gnu linear programming kit, 2010. http://www.gnu.org/software/glpk/ (accessed 22 September 2010).

[66] Íñigo Goiri, Josep Ll. Berral, J. Oriol Fitó, Ferran Julià, Ramon Nou, Jordi Guitart, Ricard Gavaldá, and Jordi Torres. Energy-efficient and multifaceted resource management for profit-driven virtualized data centers. *Future Gener. Comput. Syst.*, 28(5):718–731, May 2012.

[67] Íñigo Goiri, J.Oriol Fito, Ferran Julià, Ramon Nou, Josep Ll. Berral, Jordi Guitart, and Jordi Torres. Multifaceted resource management for dealing with heterogeneous workloads in virtualized data centers. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 25–32, 2010.

[68] Íñigo Goiri, Jordi Guitart, and Jordi Torres. Elastic Management of Tasks in Virtualized Environments. In *Proccedings of the XX Jornadas de Paralelismo 2009*, pages 671–676, 2009.

[69] Íñigo Goiri, Ferran Julià, Jorge Ejarque, Marc De Palol, Rosa M. Badia, Jordi Guitart, and Jordi Torres. Introducing Virtual Execution Environments for Application Lifecycle Management and SLA-Driven Resource Distribution within Service Providers. In *IEEE International Symposium on Network Computing and Applications (NCA'09)*, pages 211–218, 2009.

[70] Íñigo Goiri, Ferran Julià, Ramon Nou, Josep Ll. Berral, Jordi Guitart, and Jordi Torres. Energy-aware Scheduling in Virtualized Datacenters. In *Proceedings of the 12th IEEE International Conference on Cluster Computing (Cluster 2010), Heraklion, Crete, Greece, September 20-24*, 2010.

[71] Íñigo Goiri, Kien Le, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. Greenhadoop: leveraging green energy in data-processing frameworks. In *7th ACM European Conf. on Computer Systems (EuroSys)*, 2012.

[72] Green Grid Consortium, 2009. `http://www.thegreengrid.org`.

[73] Laura Grit, David Irwin, Aydan Yumerefendi, and Jeffrey Chase. Virtual machine hosting for networked clusters: Building the foundations for 'autonomic' orchestration. In *Conf. on Virtualization Technology in Distributed Computing (VTDC)*, 2006.

[74] Michael Grottke, Rivalino Matias, and Kishor Trivedi. The fundamentals of software aging. In *In Proc of 1st Int. Workshop on Software Aging and Rejuvenation (WoSAR), in conjunction with 19th IEEE Int. Symp. on Software Reliability Engineering, Seattle*, November 2008.

[75] GUROBI. Gurobi optimization, 2013. http://www.gurobi.com/.

[76] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[77] Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. Evaluation of Job-scheduling Strategies for Grid Computing. In *Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000), Bangalore, India, December 17*, pages 191–202. Springer, 2000.

[78] Jürgen Hofer and Thomas Fahringer. Grid application fault diagnosis using wrapper services and machine learning. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 233–244, Berlin, Heidelberg, 2007. Springer-Verlag.

[79] Gunther A. Hoffmann, Kishor S. Trivedi, and Miroslaw Malek. A best practice guide to resources forecasting for the apache webserver. In *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, PRDC '06, pages 183–193, Washington, DC, USA, 2006. IEEE Computer Society.

[80] Kurt Hornik. The R FAQ, 2010. ISBN 3-900051-08-9.

[81] Tibor Horvath, Tarek Abdelzaher, Kevin Skadron, and Xue Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Transactions on Computers*, 56(4):444–458, 2007.

[82] Hans-Jorg Hoxer, Kerstin Buchacker, and Volkmar Sieh. Implementing a user mode linux with minimal changes from original kernel. In *In Proceedings of the 2002 International Linux System Technology Conference*, pages 72–82, 2002.

[83] IBM. Ibm advanced power virtualization. http://www-03.ibm.com/systems/p/apv/f.

[84] IBM. Ibm blue cloud. http://www.ibm.com/cloud-computing/us/en/.

[85] IBM. Solver cplex, 2003. http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/ (accessed 17 September 2010).

[86] Intel. Intel virtualization technologies. http://www.intel.com/technology/virtualization/.

[87] Ferran Julià, Jordi Roldàn, Ramon Nou, J.Oriol Fitó, Alex Vaquè, Goiri. Í nigo, and Josep Ll. Berral. EEFSim: Energy Efficency Simulator, 2010.

[88] Ioannis Kamitsos, Lachlan Andrew, Hongseok Kim, and Mung Chiang. Optimal sleep patterns for serving delay-tolerant jobs. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking (eEnergy 2010), University of Passau, Germany, April 13-15*, pages 31–40, 2010.

[89] Nirav H. Kapadia, José A.B. Fortes, and Carla E. Brodley. Predictive application-performance modeling in a computational grid environment, 1999.

[90] Jeffrey O. Kephart. Research challenges of autonomic computing. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 15–22, New York, NY, USA, 2005. ACM.

[91] Jeffrey O. Kephart. A vision of autonomic computing, 2005.

[92] Bithika Khargharia, Salim Hariri, and Mazin Yousif. Autonomic Power and Performance Management for Computing Systems. *Cluster Computing*, 11(2):167–181, 2008.

[93] Kevin Lai, Lars Rasmusson, Eytan Adar, Li Zhang, and Bernardo A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent Grid Syst.*, 1(3):169–182, August 2005.

[94] Kien Le, Ricardo Bianchini, Margaret Martonosi, and Thu D. Nguyen. Cost-and Energy-Aware Load Distribution Across Data Centers. In *Proceedings of the Workshop on Power Aware Computing and Systems (HotPower 2009), Big Sky, MT, USA, October 10*, 2009.

[95] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. A data mining framework for building intrusion detection models. In *In IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[96] Young C. Lee and Albert Y. Zomaya. Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid-Volume*, pages 92–99. IEEE Computer Society, 2009.

[97] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy Management for Commercial Servers. *Computer*, 36(12):39–48, 2003.

[98] Lei Li, Kalyanaraman Vaidyanathan, and Kishor S. Trivedi. An approach for estimation of software aging in a web server. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering*, ISESE '02, pages 91–, Washington, DC, USA, 2002. IEEE Computer Society.

[99] Minghong Lin, Zhenhua Liu, Adam Wierman, and Lachlan L. H. Andrew. Online algorithms for geographical load balancing. In *Proceedings of the International Green Computing Conference*, San Jose, CA, 5-8 Jun 2012.

[100] Minghong Lin, Zhenhua Liu, Adam Wierman, and Lachlan L.H. Andrew. Online algorithms for geographical load balancing. In *Intl. Green Computing Conference (IGCC)*, 2012.

[101] Linux vserver. http://linux-vserver.org/Paper.

[102] Liang Liu, Hao Wang, Xue Liu, Xing Jin, Wen B. He, Qing B. Wang, and Ying Chen. GreenCloud: a New Architecture for Green Data Center. In *6th International Conference on Autonomic Computing and Communications*, 2009.

[103] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. Renewable and cooling aware workload management for sustainable data centers. *SIGMETRICS Perform. Eval. Rev.*, 40(1):175–186, June 2012.

[104] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven H. Low, and Lachlan L. H. Andrew. Geographical load balancing with renewables. *ACM SIGMETRICS Performance Evaluation Review (PER)*, March 2012.

[105] Eliot Marshall. Fatal Error: How Patriot Overlooked a Scud. *Science*, page 1347, March 1992.

[106] Peter Mell and Tim Grance. The nist definition of cloud computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011.

[107] Sun Microsystems. Sun ultrasparc t1 hypervisor. http://opensparc-t1.sunsource.net/specs/Hypervisor-api-current-draft.pdf.

[108] Mayank Mishra, Anwesha Das, Purushottam Kulkarni, and Anirudha Sahoo. Dynamic resource management using virtual machine migrations. *IEEE Communications Magazine*, 50(9):34–40, 2012.

[109] Toni Moreno, Nicolas Poggi, Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. Policy-based autonomous bidding for overload management in ecommerce websites, 2007.

[110] Toni Moreno, Nicolas Poggi, Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. Policy-based autonomous bidding for overload management in ecommerce websites. In *Proceedings of the Group Decision and Negotiation 2007*, pages 162–166. Springer-Verlag, 2007.

[111] Mysql database server, 2013. `http://www.mysql.com`.

[112] Ripal Nathuji, Karsten Schwan, A. Somani, and Y. Joshi. Vpm tokens: virtual machine-aware power budgeting in datacenters. *Cluster Computing*, 12(2):189–203, 2009.

[113] Dirk Neumann, Jochen StoBer, Arun Anandasivam, and Nikolay Borissov. Sorma - building an open grid market for grid resource allocation. In Jorn Altmann and Daniel Veit, editors, *GECON*, volume 4685 of *Lecture Notes in Computer Science*, pages 194–200. Springer, 2007.

[114] Íñigo Goiri et al. Greenslot: Scheduling energy consumption in green datacenters. In *Supercomputing*, November 2011.

[115] Íñigo Goiri et al. Intelligent placement of datacenters for internet services. In *ICDCS11*, 2011.

[116] Íñigo Goiri et al. Parasol and greenswitch: Managing datacenters powered by renewable energy. In *ASPLOS*, 2013.

[117] Nimbus. Nimbus science cloud. http://workspace.globus.org/clouds/nimbus.html.

[118] Nirvanix web services. http://developer.nirvanix.com/.

[119] Ramon Nou. Energy Efficiency: A Case Study. Technical Report UPC-DAC-RR-CAP-2009-14, Technical University of Catalonia (UPC) - Computer Architecture Department, 2009.

[120] Ramon Nou, Ferran Julia, Jordi Guitart, and Jordi Torres. Dynamic resource provisioning for self-adaptive heterogeneous workloads in smp hosting platforms pdf. In *ICE-B 2007, the International Conference on E-business (2nd)*, Jul 2007.

[121] Ramon Nou, Samuel Kounev, Ferran Julià, and Jordi Torres. Autonomic QoS control in enterprise Grid environments using online simulation. *J. Syst. Softw.*, 82(3):486–502, 2009.

[122] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGRID 2009)*, Washington DC, USA, 2009.

[123] Brian Oley. Where is the best place to build a data center? In *Data Center Journal*, 2010.

[124] Omnet, 2009. `http://www.omnet.org`.

[125] Vinicius Petrucci, Orlando Loques, and Daniel Mossé. A Dynamic Configuration Model for Power-efficient Virtualized Server Clusters. In *11th Brazillian Workshop on Real-Time and Embedded Systems (WTR)*, 2009.

[126] Vinicius Petrucci, Orlando Loques, and Daniel Mossé. A framework for dynamic adaptation of power-aware server clusters. In *Proceedings of the ACM symposium on Applied Computing (SAC 2009), Honolulu, Hawaii, USA*, pages 1034–1039, 2009.

[127] Jean-Marc Pierson. Allocating resources greenly: reducing energy consumption or reducing ecological impact? In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 127–130, New York, NY, USA, 2010. ACM.

[128] Jean-Marc Pierson. Green Task Allocation: Taking into account the ecological impact of task allocation in clusters and clouds. *Journal of Green Engineering*, 1(2):129–144, janvier 2011.

[129] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP 2001), Barcelona, Spain, September 9*, volume 180, pages 182–195, 2001.

[130] Eduardo Pinheiro, Ricardo Bianchini, and Cezary Dubnicki. Exploiting redundancy to conserve energy in storage systems. In *SIGMETRICS/Performance*, pages 15–26, 2006.

[131] Nicolás Poggi, Toni Moreno, Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. Web customer modeling for automated session prioritization on high traffic sites. In *Proceedings of the 11th International Conference on User Modeling. Corfu*, pages 25–29, 2007.

[132] Nicolas Poggi, Toni Moreno, Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. Web customer modeling for automated session prioritization on high traffic sites. In *UM '07: Proceedings of the 11th international conference on User Modeling*, pages 450–454, Berlin, Heidelberg, 2007. Springer-Verlag.

[133] Nicolas Poggi, Toni Moreno, Josep Ll. Berral, Ricard Gavaldí, and Jordi Torres. Self-adaptive utility-based web session management. *Comput. Netw.*, 53(10):1712–1721, July 2009.

[134] Nicolas Poggi, Toni Moreno, Josep Ll. Berral, Ricard Gavaldí, and Jordi Torres. Self-adaptive utility-based web session management. *Comput. Netw.*, 53(10):1712–1721, 2009.

[135] Nicolás Poggi, Josep Ll. Berral, Toni Moreno, Ricard Gavaldà, and Jordi Torres. Automatic detection and banning of content stealing bots for e-commerce, 2007.

[136] The FreeBSD Project. The freebsd documentation project, 2007.

[137] Ioan Raicu, Yong Zhao, Catalin Dumitrescu, Ian Foster, and Mike Wilde. Dynamic resource provisioning in grid environments, 2007.

[138] Supranamaya Ranjan, J. Rolia, H. Fu, and E. Knightly. Qos-driven server migration for internet data centers. In *10th International Workshop on Quality of Service (IWQoS 2002)*, pages 3–12. Citeseer, 2002.

[139] RDLab - Department of Software UPC, 2011. http://rdlab.lsi.upc.edu/.

[140] Suzanne Rivoire, Mehul A. Shah, Parthasarathy Ranganathan, and Christos Kozyrakis. JouleSort: a balanced energy-efficiency benchmark. In *2007 ACM SIGMOD international conference on Management of data*, page 376, 2007.

[141] Kai Shen, Hong Tang, Tao Yang, and Lingkun Chu. Integrated resource management for cluster-based internet services. *SIGOPS OS Rev.*, 36(SI):225–238, 2002.

[142] Daniel P. Siewiorek and Robert S. Swarz. *Reliable computer systems - design and evaluation (3. ed.)*. A K Peters, 1998.

[143] Luis Moura Silva, Javier Alonso, Paulo Silva, Jordi Torres, and Artur Andrzejak. Using virtualization to improve software rejuvenation. *Network Computing and Applications, IEEE International Symposium on*, 0:33–44, 2007.

[144] Lindsay I. Smith. A tutorial on principal components analysis, 2002.

[145] Borja Sotomayor, Kate Keahey, and Ian Foster. Overhead matters: A model for virtual resource management. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006), Tampa, Florida, USA, November 11*, page 5, 2006.

[146] Borja Sotomayor, Kate Keahey, and Ian Foster. Combining Batch Execution and Leasing using Virtual Machines. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC 2008), Boston, MA, USA, June 23–27*, pages 87–96, 2008.

[147] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14–22, September 2009.

[148] SPECweb2009 E-commerce workload, 2009. http://www.spec.org/web2009/.

[149] Matt Stansberr. Data center locations ranked by operating cost. In *SearchDatacenter.com*, 2006.

[150] Christopher Stewart and Kai Shen. Some joules are more precious than others: Managing renewable energy in the datacenter. In *HotPower*, 2009.

[151] Ann T. Tai, Herbert Hecht, Savio N. Chau, and Leon Alkalaj. On-board preventive maintenance: Analysis of effectiveness and optimal duty period. In *Proceedings of the 3rd Workshop on Object-Oriented Real-Time Dependable Systems - (WORDS '97)*, WORDS '97, pages 40–, Washington, DC, USA, 1997. IEEE Computer Society.

[152] Ying Tan, Wei Liu, and Qinru Qiu. Adaptive power management using reinforcement learning. In *International Conference on Computer-Aided Design (ICCAD '09)*, New York, NY, USA, 2009. ACM.

[153] Gerald Tesauro, Rajarshi Das, Hoi Chan, Jeffrey O. Kephart, David Levine, Freeman Rawson, and Charles Lefurgy. Managing power consumption and performance of computing systems using reinforcement learning. *Advances in Neural Information Processing Systems*, 20, 2007.

[154] Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Intl. Conf. on Autonomic Computing (ICAC 2006)*, 2006.

[155] The Grid Workloads Archive, 2009. `http://gwa.ewi.tudelft.nl`.

[156] Jordi Torres, David Carrera, Vicenç Beltran, Nicolás Poggi, Kevin Hogan, Josep Ll. Berral, Ricard Gavaldà, Eduard Ayguadé, Toni Moreno, and Jordi Guitart. Tailoring resources: The energy efficient consolidation strategy goes beyond virtualization. In *Proceedings of the 2008 International Conference on Autonomic Computing*, ICAC '08, pages 197–198, Washington, DC, USA, 2008. IEEE Computer Society.

[157] Tpc-w java version, 2013. `http://pharm.ece.wisc.edu/tpcw.shtml`.

[158] Kishor S. Trivedi, Kalyanaraman Vaidyanathan, and Katerina Goseva-popstojanova. Modeling and analysis of software aging and rejuvenation. In *In Proceedings of the IEEE Annual Simulation Symposium*, pages 270–279, 2000.

[159] W. Pitt Turner, John H. Seader, Vince Renaud, and Kenneth G. Brill. Tier classifications define site infrastructure performance, 2008.

[160] Kalyanaraman Vaidyanathan and Kishor S. Trivedi. A measurement-based model for estimation of resource exhaustion in operational software systems. In *Proceedings of the 10th International Symposium on Software Reliability Engineering*, ISSRE '99, pages 84–, Washington, DC, USA, 1999. IEEE Computer Society.

[161] Kalyanaraman Vaidyanathan and Kishor S. Trivedi. A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, 2:2005, 2005.

[162] Alex Vaqué, Íñigo Goiri, Jordi Guitart, and Jordi Torres. *EMOTIVE Cloud: The BSC's IaaS Open Source Solution for Cloud Computing*. IGI Global, 2012-01-31 2012.

[163] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[164] David Vengerov and Nikolai Iakovlev. A reinforcement learning framework for dynamic resource allocation: First results. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 339–340, Washington, DC, USA, 2005. IEEE Computer Society.

[165] Verizon (Jan.2013). http://www.verizonenterprise.com/about/network/latency.

[166] Akshat Verma, Puneet Ahuja, and Anindya Neogi. Power-aware dynamic placement of hpc applications. In *ICS '08: International Conference on Supercomputing*, pages 175–184, New York, NY, USA, 2008. ACM.

[167] Akshat Verma, Gargi Dasgupta, Tapan Kumar, Nayak Pradipta, and De Ravi Kothari. Server workload analysis for power minimization using consolidation, 2009.

[168] Pascale Vicat-Blanc Primet, Jean-Patrick Gelas, Olivier Mornard, Dinil Mon Divakaran, Pierre Bozonnet, Mathieu Jan, Vincent Roca, and Lionel Giraud. State of the art of os and network virtualization solutions for grids. Technical report, INRIA, September 2007. "Delivrable #1 : HIPCAL ANR-06-CIS-005".

[169] Vmware. http://www.vmware.com/.

[170] Werner Vogels. Beyond server consolidation. *Queue*, 6(1):20–26, 2008.

[171] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and Scott W. Stornetta. Spawn: A distributed computational economy. *Software Engineering*, 18(2):103–117, 1992.

[172] Yi-Min Wang and Ming Ma. Strider search ranger: Towards an autonomic anti-spam search engine. In *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, page 32, Washington, DC, USA, 2007. IEEE Computer Society.

[173] Jonathan Wildstrom, Peter Stone, and Emmett Witchel. Autonomous return on investment analysis of additional processing resources. In *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, page 15, Washington, DC, USA, 2007. IEEE Computer Society.

[174] Jonathan Wildstrom, Peter Stone, Emmett Witchel, and Mike Dahlin. Machine learning for on-line hardware reconfiguration. In *In Proceedings of the 20th International Joint Conference On Artificial Intelligence*, pages 1113–1118, 2007.

[175] Jonathan Wildstrom, Emmett Witchel, and Raymond J. Mooney. Towards self-configuring hardware for distributed computer systems. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 241–249, Washington, DC, USA, 2005. IEEE Computer Society.

[176] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and gray-box strategies for virtual machine migration. In *4th USENIX Conf. on Networked systems design & implementation (NSDI)*, 2007.

[177] XtreemOS European Project, 2006-2010. `http://www.xtreemos.eu`.

[178] Guangsen Zhang and Manish Parashar. Cooperative mechanism against ddos attacks. In *In: IEEE International Conference on Information and Computer Science (ICICS 2004), Dhahran, Saudi Arabia*, 2004.

[179] Guangsen Zhang and Manish Parashar. Cooperative defense against ddos attacks. Las Vegas, NV, USA, 06/2005 2005. CSREA Press.

[180] Qi Zhang. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *In proceedings of the International Conference on Autonomic Computing, ICAC 2007*, page 27, 2007.

[181] Qi Zhang, Ludmila Cherkasova, Ningfang Mi, and Evgenia Smirni. A regression-based analytic model for capacity planning of multi-tier applications. *Cluster Computing*, 11(3):197–211, 2008.

[182] Yanwei Zhang, Yefu Wang, and Xiaorui Wang. Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *Middleware*, volume 7049 of *Lecture Notes in Computer Science*, pages 143–164. Springer, 2011.

[183] H. Zhu and Manish Parashar. Self-adapting, self-optimizing runtime management of grid applications using pragma. In *PRAGMA, Proc. of NSF NGS Program Workshop, IEEE/ACM 17th IPDPS*, 2003.