Adaptive Scheduling on Power-Aware Managed Data-Centers using Machine Learning

Josep Ll. Berral, Ricard Gavaldà, Jordi Torres
Universitat Politècnica de Catalunya and Barcelona Supercomputing Center
{berral,torres}@ac.upc.edu, gavalda@lsi.upc.edu

***Abstract*—Energy-related costs have become one of the major economic factors in IT data-centers, and companies and the research community are currently working on new efficient power-aware resource management strategies, also known as "Green IT". Here we propose 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. The principal contribution is the combination of consolidation and virtualization technologies, mathematical optimization methods, and machine learning techniques. The data-center infrastructure, tasks to execute, and desired profit are cast as a mathematical programming model, which can then be solved in a different ways to find good task schedulings. 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. The machine learning is used to estimate the initially unknown parameters of the mathematical model. In particular, we need to predict a priori resource usage (such as CPU consumption) by different tasks under current workloads, and estimate task service-level-agreement (such as response time) given workload features, host characteristics, and contention among tasks in the same host. Experiments show that machine learning algorithms can predict system behavior with acceptable accuracy, and that their combination with the exact or approximate schedulers manages to allocate tasks to hosts striking a balance between revenue for executed tasks, quality of service, and power consumption.**

***Index Terms*—Data centers, Energy, Heuristics, Machine Learning, Scheduling, SLA**

I. Introduction

Nowadays the concept of Cloud has become one dominating paradigm in the externalization of information and IT resources for people and enterprises. 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. As a consequence, the Cloud led to the creation of computing-resource provider companies, starting a *data-center race* offering computing and storage resources at low prices. The data-center 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, 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 power cost. The aim of this paper 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.

Here we propose a methodology of autonomic energy-aware scheduling that dynamically adapts to varying task types and workloads, and even to varying infrastructure. The main contribution is the combination of technologies such as virtualization and consolidation (to move tasks between hosts), mathematical programming (to create and solve models of tasks, hosts, and constraints), and machine learning and data mining (to build these models from examples of past behaviors). While all these techniques have been employed in the past for autonomic resource allocation in data-centers, we are not aware that they have been used together, and less in the context of energy-aware allocation.

The proposed methodology models a grid based data-center as a set of resources and as a set of jobs (web services), each with its energy requirements or load per time unit, involving a consumption cost and an execution reward, focusing the schedule on revenue maximization and power saving. The approach takes advantage of *virtualization*, which lets data-centers to allocate several but isolate jobs in the same physical machine isolating them and, most importantly, migrate running jobs between machines. Here we make extensive use of this latter possibility by shutting down down inactive physical machines and fully using those that remain active; this consolidation strategy is well-known for maximizing resource usage while reducing power consumpton. The challenge is how to do that while executing a maximal number of jobs without compromising QoS or performance. We solve this challege by formulating it as a mathematical optimization problem specified by a function to be optimized plus a number of constraints describing the correct and acceptable data center behavior.

The machine learning component is required because many of the parameters and functions in the optimization problem above are unknown a priori. One can view machine learning as the ability to create models from past experience, as opposed to explicit modelling by human experts. Roughly speaking, in our methodology, we use initial data to create a model for each element in the system (an application type, a workload,

a physical machine, a high-level service requirement). These momdels can be further updated from new test or real runs. The system can then use these models to make informed choices towards optimizing some externally defined criterion. For example, for a given task type (application, service), models can predict features such as minimum CPU usage and dependence on workload volume. For Quality of Service elements, such as response time, the model can predict their dependence on the resources allocated to the task and the low-level monitored quantities. All in all, the problem to solve at any given time is to decide which resources are assigned to each job, while maximizing a function result of the profit obtained by the job revenues, the power costs and the penalties due to QoS loss or SLA violation. Scheduling rounds are performed periodically to take into account changes in workloads and tasks entering or leaving the system. Model-building (training) phases can be carried out either on-line less frequently, or offline (as we do in this paper, for the time being).

In an a scheduling phase, the task is to solve (find an optimal solution) to the mathematical program made up from the parameters provided by the models of the data center elements. Solving such constraint optimization models exactly is often computationally expensive: it is in fact NP-complete in general for the so-called Integer Linear Programs (ILP) that we obtain. We report experiments with exact, off-the-shelf ILP solvers, but view as one of the main contributions of this work the observation that heuristics such as Best-Fit seem to find almost-optimal solutions in very reasonable time and memory. This is key if one wants to allow frequent scheduling rounds with low overhead on the whole system. We also observe that our framework is flexible enough to incorporate new constraints or policies as parts of the models to be solved. As an example, we show how to further schedule tasks so as to prefer schedules with few running-job migrations, in addition to concerns about revenue, SLA, and power consumption.

This work is organized as follows: Section II presents previous work in this area. Section III describes the approach, the prediction models, the ILP data-center model, and the solving methods. Section IV presents the evaluation of the method and discusses the results. Finally, Section V summarizes the conclusions and future work.

II. Related Work

The advantages of consolidation using virtualization were pointed out, e.g., Vogels et al. [26], while e.g. Nathuji et al.[20] widely explored its advantages from a power efficiency point of view. Petrucci et al. [21] proposed a dynamic configuration approach for power optimization in virtualized server clusters and outlines an algorithm to dynamically manage it. VM migration and VM placement optimization are studied by Liu et al. [19], in order to improve the VM placement and consolidate in a better way. Based on these works, Goiri et al. [12], [13] introduce the SLA-factor into the self-managing virtualized resource policies. 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.

Another key technique related with virtualization and consolidation is the ability to determine when to turn on and off physical machines. E.g. Goiri et al. [13], [10] 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. This turning on-off technique is also applied in by Kamitsos et al. [18], which sets unused hosts to a low consumption state in order to save energy.

Previous works use machine learning in order to take benefit from the system information. Our previous work [4], based on Goiri et al. [13], proposed a framework that provides a consolidation methodology using turning on/off machines, power-aware consolidation algorithms, and machine learning techniques to deal with uncertain information while maximizing performance. Other approaches like Tan et al. [23] and G.Tesauro et al. [24] proposed managing power and resources allocation using reinforcement learning algorithms. The proposed techniques use learners like Q-learning, Sarsa and Markovian Decision Processes algorithms [22], focusing the decision making on the policies to be applied at each time. Other works applying machine learning to control resources and power, like G.Dhiman et al. [7], focus on policies for specific resources like hard disk and network states. As we reach to know, all the current approaches using ML for power and resource management are oriented towards the selection of policies and learning the relation *system state vs. specific policy*. In this work we are applying the ML techniques over the direct learning of resources/power vs. load behavior, to supply information to a more complex model, also made by us, representing the system.

Work done by J.S.Chase et al. present the MUSE framework [5], 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 mathematical model presented here follows the same outline and basic concepts they used, with the addition of our own policies and the machine learning contribution.

III. Framework Architecture

A. Management strategy

A cloud can be viewed as a set of jobs or tasks to be distributed along a set of resources, so main decisions to take are primarily what resources are given to what jobs, assuring that each job receives enough resources to be executed properly with respect to global goals and customer requirements.

In order to make this management efficient, the system must know or must be able to identify the different states of the system, how it behaves and and reacts to a given situation, and also identify and recognize the behavior of the tasks that are being run. For this reason this case of study focuses on two

aspects of the self-managing process: self-configuration and self-optimization. The self-configuration process applied here takes care of recognizing the behavior of the hosted jobs (web services, in this case of study) so the scheduler can, using the available data from the load of the job, to estimate the amount of resources the job is needing in order to run and give good responses. The self-optimization process applied here is in charge of scheduling the jobs to resources (jobs to hosts with available resources), minimizing the amount of resources used while estimating the jobs can run with a good quality of service responses. The self-adaption schema is seen in Figure 1.

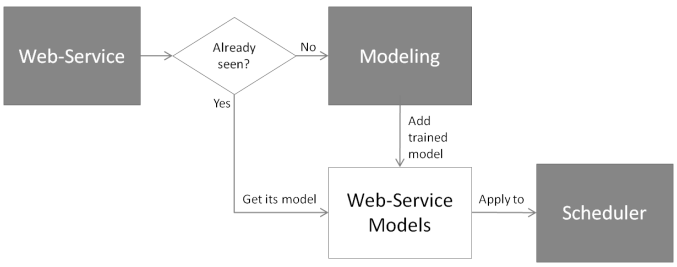

Figure 1. Web-services Modeling Schema

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 the their past behavior. These models can then be used to predict future behaviors. The basic schema of machine learning is found in Figure 2. In this work machine learning (ML) is used to find models for web services behavior and load versus resources relations, in order to predict useful information in runtime, as shown in the next subsections.

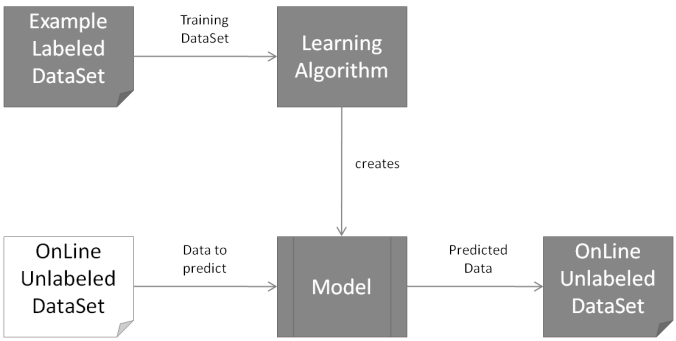

Figure 2. Machine Learning Basic Schema

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.

B. Consolidation Strategy

As shown in our previous work [4] for the case of CPU, the relation between resource usage and power grows non-proportionally and sub-lineally. This explains the potential for power saving by consolidation.

Example given, in a Intel Xeon 4-CPU machine, the power consumption (in Watts/hour) when in all CPU's are in idle state is 235, when only 1 CPU is active is 267.8, and when 2, 3, and 4 CPU's are active, the power consumption is respectively 285.5, 302.5, and 317.9. 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) processors and shutting down the second one.

C. Web Service Modeling

Each job 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 job (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 job entering the system, the amount of required resources as a function of the kind of load it is receiving.

When a new kind of job 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 transfered 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 web service could be specific software packages like Apache v.X, Tomcat v.Y, with attached modules like PHP or MySQL DB services. Each execution of the job provides pairs of $\langle$workload attributes, resources used$\rangle$ that can be used for (further) training the model describing this kind of job.

D. Prediction on Scheduling

Once in the scheduling phase, the goal is to assign as few resources as possible to running jobs keeping user satisfaction but reducing power usage. We assume in this work that the notion of user-satisfaction is captured by Service Level Agreements (SLA) for each job, that is an agreement between customer and provider about the quality of service (QoS) that the provider assures the job will offer externally.

For a web service, and for this case of study, a relevant SLA term is the maximum response time (RT) the service will provide per request. A commonly used form of SLA function for response time

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

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. The scheduling policy applied in this works follows the next points:

- Give each job the estimated resources according to their current load.
- Attempt to consolidate by maximizing the number of IDLE machines (no jobs in them) and shut them down.
- Attempt to consolidate more by reducing the given resources for the jobs while their expected response time does not exceed the minimum agreed.

E. ILP model of the Data-Center

A grid based data-center can be modeled as a set of resources, each one with a consumption cost, and a set of jobs to be executed with a set of resource requirements, profits and execution penalties. At each scheduling round what resources are assigned to each job must be decided, depending always in the requirements and conditions established by each SLA. The best solution will be that one that maximizes or minimizes a target function, usually describing the profit of the solution.

Basically the solution defined here is a mathematical model for scheduling a binary matrix $H \times J$, where H and J are the sets of (indexes of) hosts and jobs, and where each position $[h, j]$ indicates whether job j is or not in host h. A valid solution must satisfy the condition that a job must be run entirely in one and only one host, as (by assumption at this stage) it can not be split in different hosts. Each job needs a certain amount of resources to run properly at each moment, such as CPU quota, memory space and I/O access, predicted by the learned functions. Also each host has available a set of resources that can not be overloaded, with an associated function relating usage to power consumption. The generic model is as follows:

Maximize:

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

Output:

$Schedule[H, J]$, Integer Binary ; the Schedule

$GivenRes[J]$, Integer; resources for Job_i

Parameters:

$Resources(h)$, CPUs, Memory, etc. existing in host h

$RT_{i,0}$ and α, agreed RT and α for job i to fully satisfy its SLA

$Load_i$, requests per second, bytes per request, etc. for job i

$f_{Pwr}(h)$, power consumption function of h depending on its resource use

$f_{ReqRes}(i)$, max required resources for a job i given its load

$f_{MinRes}(i)$, min required resources for a job i given its load

Subject To:

(1) $\forall i \in J : \sum^{h \in H} Schedule[h, i] = 1$

(2) $\forall h \in H : \sum^{i \in J} GivenRes[i] \cdot Schedule[h, i] \leq Resources(h)$

(3) $Power = \sum^{h \in H} f_{Pwr}\left(\sum^{i \ runs \ in \ h} GivenRes[i] \right)$

(4) $\forall i \in J : f_{MinRes}(i) \leq GivenRes[i] \leq f_{ReqRes}(i)$

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

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

Its meaning in words is:

- The function to be maximized is the profit obtained from the revenue per each executed job according to their SLA fulfillment level $f_{revenue}(Job_i, SLA(Job_i))$, minus the power consumption $f_{powercost}(Power)$, all in euros.
- The output elements are the $H \times J$ schedule and the amount of resources given to each job.
- The parameters are the specifications per host, the load received per job, the function of power $f_{Pwr}(h)$, and the expected maximum and minimum resource usage of each job $f_{ReqRes}(i)$ and $f_{MinRes}(i)$. The resource usage functions come from a learned function, explained in next section.
- Constraints (1) and (2) ensure the coherence of the data-center system, avoiding jobs present in no host or two hosts, and overloading a host resource.
- Constraint (3) computes the the power consumption of a machine; at this stage we assume it depends solely on the number of processors used to execute its assigned jobs, following the example discussed in Section III-B.
- Constraint (4) bound the resource given to each job in its tentative host, bounds set by the predictions of our trained function relating *job load* with *required resources*. Constraint (5) is the function relating *load, resources given, resources needed* with response time. This function is the second to be learned, as explained in next section. Finally constraint (6) represents the SLA function for each job given their expected response time.

This model describes what settings of the output variables are acceptable solutions and their values. Also, the model is open to adding new constraints, prices, costs or penalties; in

case of improvement of the number or characteristics of the components of the data-center, the agreements between the provider and customers, or new features. An example is seen in the next section IV where we introduce an added cost for job migrations. Now, the problem is how to solve it, and how to find an optimal or near-optimal solution, given a specific scenario and specific functions.

Note: More about the model can be found in the technical report [2], where a first approach to the model, represented as an Integer Linear Program has been previously validated and tested using different policies, such as economic maximization and power-consumption or migration reduction.

F. Schedule Solving

In our scenario, the functions of revenue, power cost, power, SLA are linear functions, and we manage for the learned functions of resources and response time to be also linear. This makes the mathematic model to be solved by an Integer Linear Program (ILP) solver. Solving ILPs is well-known to be a NP-complete problem, and therefore exhaustive search algorithms are the only ones guaranteed to find the optimal solution. But heuristic approaches with often good performance are known, and also local-search methods can be very appropriate in a case as ours when we may have a fairly good solution from the previous scheduling round.

Here we have faced the possibility of using an exact solver and different heuristics and ad-hoc solvers. We have used GUROBI [14] ILP solver and two classical heuristics (First-fit and ordered Best-fit) in order to compare optimal values and cost in time. We have also used the ad-hoc λ-Round Robin strategy [4], a power-aware greedy algorithm which computes the approximate resources needed for the job execution, setting up just enough machines in this amount plus a λ percentage, and shutting down the rest of machines. The λ-RR schedules the jobs over the active, non-full, machines in a sequential order. The advantage of the λ-RR is that the amount of "wasted" resources is always at maximum the λ percent of the requirements, also the best-fit scheduling algorithm uses the profit function to be maximized for each potential placement.

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, hence opening the possibility in future work for dealing with non-linearities in constraints (4) (power consumption), (5) (predictive models) and (6) (SLA). Algorithms are shown in Algorithms 1, 2, 3. For more details in the classical and approximate algorithms, and their sub-optimal grants, consult the works of T. H. Cormen et al. [6] and V. Vazirani [25].

Algorithm 1 First Fit algorithm

```
for each job i:
    get_data(i);
    /* from learned function RT() */
    cpu_quota[i] <- get_estimate_max_cpu(i);
end for
for each job i:
    c_host <- 1;
    while (not fit(cpu_quota[i],c_host) or c_host<=maxHosts)
        c_host <- c_host + 1;
    done
    update_candidate_host(c_host,cpu_quota[i],i);
end for
```

Algorithm 2 λ-Round Robin algorithm

```
for each job i:
    get_data(i);
    /* from learned function RT() */
    cpu_quota[i] <- get_estimate_max_cpu(i);
end for
numHosts <- calculateNumHosts(cpu_quota[],lambda);
c_host <- 1;
for each job i:
    visited <- 0;
    while (not fit(cpu_quota[i],c_host) or visited<=numHosts)
        c_host <- (c_host + 1) % numHosts;
        visites <- visited + 1;
    done
    if (visited <= numHosts) :
        update_candidate_host(c_host,cpu_quota[i],i);
    else :
        cpu_quota[i] <- 0;
        update_candidate_host(null_host,cpu_quota[i],i);
    end if
end for
```

Algorithm 3 Descending Best-Fit algorithm

```
for each job i:
    get_data(i);
    /* from learned functions CPU() and RT()*/
    min_q <- get_estimate_min_cpu(i);
    max_q <- get_estimate_max_cpu(i);
end for
reorder(jobs,decreasing);
for each job i:
    best_profit_aux <- 0;
    best_quota_aux <- 0;
    candidate_host <- 0;
    for each host h:
        <profit_aux,q_aux> <- golden_search(i,h,max_q,min_q);
        if (profit_aux > best_profit_aux) :
            best_profit_aux <- profit_aux;
            best_quota_aux <- q_aux;
            candidate_host <- h;
        end if
    end for
    update_candidate_host(candidate_host,best_quota,i);
end for
```

IV. EXPERIMENTS

A. Environment and Workload

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 data-center in order to check the scalability of the method. The information about behaviors and response times, corresponding to the learning executions, has been obtained from the execution of the test workloads (explained in detail later) on a Intel Xeon 4 Core 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]. Then a full scalability test has been performed using the workloads against simulated data-center, recreated in a R [16] version of the cloud simulator EEFSIM made by Nou et al. [11], [17] and being programmed with the properties of the real model machines.

The simulated data-center contains a set of 20 machines, representing copies of the model machine, each containing 4 processors – 20 $\times$ 4CPU's. We thus do not fully use the potential of our model to deal with heterogeneous data-centers with several machine kinds. 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.

The workload used corresponds to the Li-BCN Workload 2010 [3], 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 an hybrid architecture) with different input loads and with different processing units. 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 model for these test web-sites, as proposed in [17], is focused on the response time high-level metric and, relating this metric with both incoming users load and CPU/MEM/BWD usage of the server. The following stress experiments represent a run of these workload pieces during 4 days (monday to thursday).

In order to set the economic values of each involved element, the EEFSIM and its related works 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). Also we fixed the power cost to 0.09 euro/KWh, the current average in Spain [1].

As a parameter defining the Quality of Service, the response rime at the data-center output is used. As a case of study a penalization of $SLA(job) \cdot Revenue(job)$ (price of job per hour) is applied in order to add a profit factor. The jobs on workload have as RT_0 the values of $0.004s$ or 0.008 (each job 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, job pricing, power pricing, and the SLAα parameter have been set to other values in some of the experiments.

B. Learning Process and Validation

The usual Data Mining methodology to create and validate models is to run a validation test to check how the model adapts to data, and a final test using brand new data not seen at any time during the model building. The data sets used for learning are two example executions of 2 hours length from different days, stressing in the proper way the real web site, and monitoring the relevant data to work with. This data, referred as *input variables*, are the following: the number of requests, bytes per request, CPU and MEM consumed by the job inside the VM, CPU and MEM demanded by the VM, CPU and MEM consumed in the whole system, bandwidth of the network connection, average time per request consumed by the web service job, and response time obtained at the network gateway.

The first model is trained to predict the usage of two most obvious resources, CPU and memory, and is called as an oracle when building the ILP to provide the f_{MinRes} function in constraint (4) of the model. However, experiments have revealed that while CPU can be related directly with load in a given time unit, predicting the necessary memory must take into account not only the current observable state of the system, but also its past history due to the memory bloating, which in this case web servers experiment due to data cache, client and session pools, etc. For the experiments reported here, we address CPU prediction and leave the more complex study of memory model to future works.

To obtain this relation load $\sim$ CPU, the job model must learn without stress of external factors, so the web service job is run inside a VM without any other job running on the virtual or physical machine. The *output variable*, the value to be predicted, is the CPU consumed by the web service inside the VM, as observing the data obtained from the data collection the VM demands CPU to the physical machine according to its internal demand with the addition of a small overhead around the 10% of CPU. In the workbench scenario built to collect data from the job, 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 (discarding bandwidth this time).

The chosen learning method, demonstrated in previous works and experiments to work for piecewise linear functions, is a regression tree algorithm *M5P* from the popular machine learning package WEKA [15]. The M5P algorithm builds a decision tree with a linear regression model at each leaf and has a good compromise between accuracy and computation cost. The mean squared error of the model so built 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. Figure 3 shows the real vs prediction values from the model selection test and the validation test with new data.

The second model to obtain is the relation among Response Time (RT) $\sim$ load, resources needed and resources obtained, and constitutes the f_{RT} function in constraint (5), and derives the f_{ReqRes} in constraint (4) as finding the minimum resources to accomplish a given RT. Here the objective is to learn how the web service jobs RT behaves in front of stress, so we can predict RTs for different possible job schedules. After a feature selection process, the most relevant attributes selected to predict the RT for this situation are the four load attributes (requests per second, bytes per request, available bandwidth in Mbps and time per request on web service), plus the total used CPU from physical machine, the given CPU to the VM and the required CPU from the web service job. Note that the memory attributes do not appear this time, as memory becomes a non-critical resource for the experiments done with

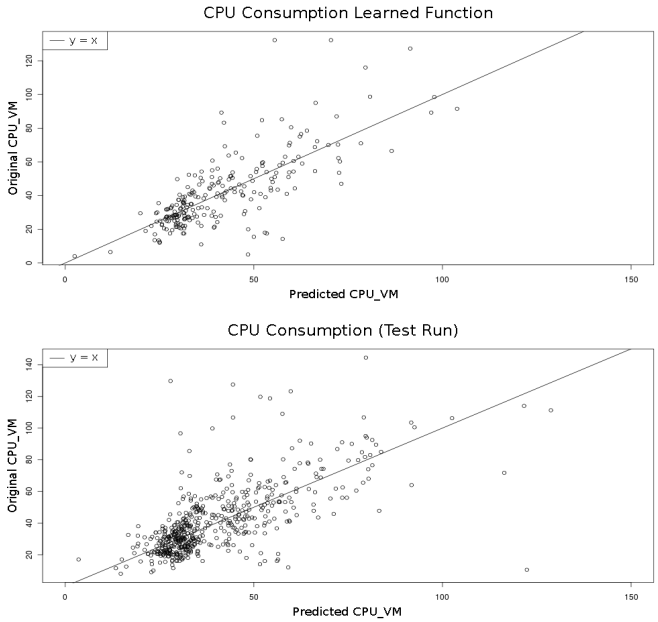

Figure 3. CPU Real vs Predicted in Validation Test

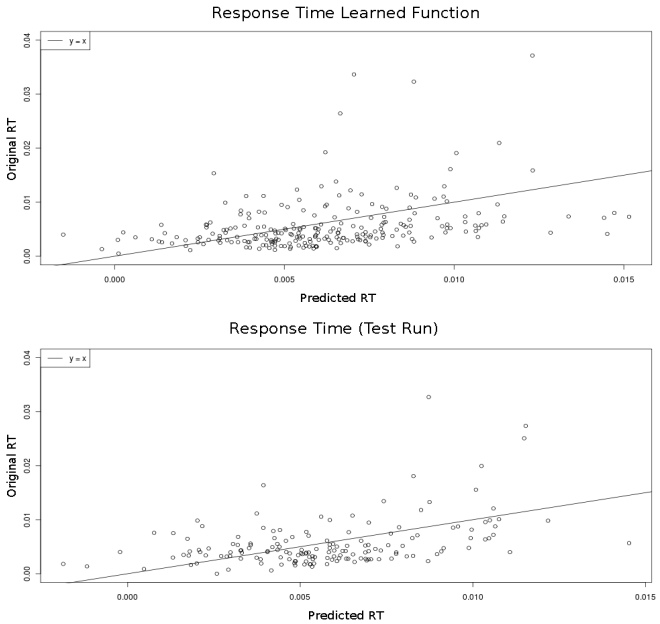

Figure 4. RT Real vs Predicted in Validation Test

the Xeon machine, while in previous experiments using a non-HPC machine (Celeron M, single core @ 2Ghz, 512Mb RAM) the RAM memory became a critical resource and it was present during the prediction process. Next and future work will focus the behavior of memory and its modeling and usage prediction. As a learning method, we simply used linear regression because it gave reasonably good results and is conveniently linear to be plugged in in constraint (5). The selected algorithm this time is the linear regression method. With the prediction model, the mean squared error obtained from the predictions is around $2.374979 \cdot 10^{-}5s$ (stdev 0.004), each one with around 3000 instances (another two different subsets of the workload). Figure 4 show the real vs prediction values from the validation test and the final test with new data.

C. ILP and Algorithms testing

Once it can predict the amount of CPU and RT of jobs conditioned to their environment, the scheduler can now test different scheduling configurations before applying one. Next, the ILP solver and the mentioned heuristic methods are run on the workload explained above for comparison.

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 the current input, we set a time limit of 4 minutes for the ILP solver. Further, for these experiments the λ for the Round Robin is set to 30 because the study in [4] seemed to indicate it is optimal for settings similar to ours. Also, the best-fit algorithm uses as input the jobs ordered by descending minimum CPU required (learned f_{MinRes} function). Tables I and II show the result for the run of all algorithms with different parameters, and its statistic information from running each method 10 times.

		Avg QoS	Power	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
ILP 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

Parameters: Power Cost = 0.09 euro/Kwh,
Job Revenue = 0.17 euro, MaxRT = $2 \cdot RT_0$

Table II
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. Additionally, as discussed before, it can potentially accommodate non-linear constraints better than ILP. It is thus a strong candidate to replace the exhaustive search method in future work. Figure 5 shows dynamically the comparative of power and SLA fulfillment level over time for the different used schedulers.

In order to test how changing power price or SLA function thresholds affects the solution, some long runs have been

Parameters	Method	Power (Kw)	Migs	Profit	AvgQoS	MaxQoS	MinQoS	UsedCPU	UsedHosts
	FirstFit	290.2	1421	177.290	0.519	0.524	0.514	3523	921
Power Cost: 0.09 e/Kwh	λ-RR	358.9	520	241.048	0.716	1	0.001	3917	1237
Job Revenue: 0.17 e	Desc. BestFit	203.4	1665	321.002	0.866	1	0.321	2230	660
MaxRT: $2 \cdot RT_0$	ILP Solver	169.8	1963	326.935	0.878	1	0.348	1568	571
	FirstFit	293.2	1491	69.871	0.515	0.520	0.510	3567	930
Power Cost: 0.45 e/Kwh	λ-RR	230.7	519	132.610	0.604	1	0.001	3985	1161
Job Revenue: 0.17 e	Desc. BestFit	139.8	1634	255.980	0.818	1	0.166	1504	456
MaxRT: $2 \cdot RT_0$	ILP Solver	151.3	1998	270.543	0.862	1	0.304	1483	503
	FirstFit	289.8	1513	310.569	0.859	0.861	0.857	3529	919
Power Cost: 0.09 e/Kwh	λ-RR	232.7	527	311.619	0.849	1	0.261	4077	1177
Job Revenue: 0.17 e	Desc. BestFit	155.7	1404	350.892	0.932	1	0.565	1777	508
MaxRT: $10 \cdot RT_0$	ILP Solver	161.1	2007	354.891	0.852	1	0.276	1697	528

Table I
SCHEDULING COMPARATIVE BETWEEN TECHNIQUES

performed changing the price of power in front of a constant revenue per job. Table III and Figure 6 show shows the results of the different experiments, indicating the power consumed, the profit to be maximized, the levels of QoS obtained, the average computational time spent (limited to 4 minutes), the gap between the solution found and the lower bound obtained by the solver within allowed time, and the sum used CPUs and hosts.

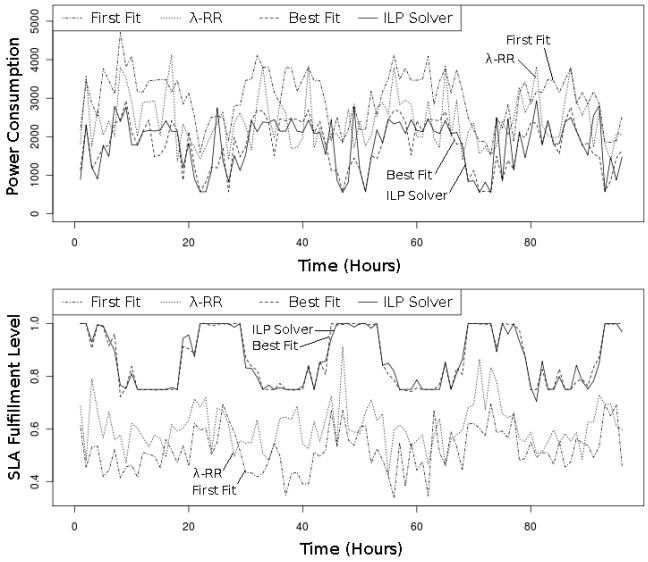

Figure 5. Power and SLA Comparative on the Schedulers

As expected, increasing the cost of power makes the model use less of it and compress jobs in fewer active CPUs. Also the profit drops down as power cost rises. The average RT is also degraded as the number of CPUs decreases. Remember that all jobs must be run, so when power surpasses the revenue, profit eventually becomes negative, which means there would be no business for the data-center company. Note that when power cost is reduced to zero, some jobs still have punctually low SLA fulfillments due to high loads in the given workload, surpassing the capabilities of a full host. In this case, the provider cannot be held responsible for the service degradation, and the customer should have asked for either more powerful machines or more of them and use content distribution mechanisms.

D. Reducing migrations

Once seen that the model is able to schedule following a set of policies depending on the values for each involved factor (service level agreement fulfillment or power cost) using the exhaustive or the approximate algorithms, a problem that could arise in a real system is the amount of migrations done. Some migration processes can be expensive in time or produce small downtimes in the migrated service, and so they should be avoided if possible. However, the model presented here does not care about the number of migrations, as the solution found may by chance turn the previous schedule upside down - even if a conservative solution exists with exactly the same profit!

As a proof of concept, suppose that migrating a VM can last up to 5 minutes, and in this time the service may stop responding. The SLA fulfillment would then be 0 for this interval. A way to maximize the profit considering that each migration can imply zero revenue during X minutes is to add a penalty for each migration up to the revenue of the given job in the migration time. E.g. when a job with an income of 0.17 euros/hour is migrated (max 5 minutes), the service provider grants the customer a discount of 0.014 euros for it, covering a priori the chance of service degradation during migration. With this idea, we modify the model as shown next, adding an extra term to the profit function in our model, so the profit per scheduling round will result as shows next model. Note that the constraint is linear as one of the arguments of the "xor" is a parameter, not a variable.

Maximize:

$Profit' = Profit - f_{penalty}(Migrations(Job_i))$

Parameters:

$oldsched[Hosts, Jobs]$, as the last schedule for current jobs

Subject To:

$$Migrations = \frac{1}{2} \sum_{\langle h,j \rangle \in oldsched} (schedule[h,j] \oplus oldsched[h,j])$$

Euro/Kwh	Power (Kw)	Migs	Profit (euro)	AvgQoS	MaxQoS	MinQoS	TimeSpent	Gap LB	UsedCPU	UsedHosts
0.00	362.1	2025	345.425	0.883	1	0.376	117.162	0.093	4556	1139
0.01	185.1	1959	341.138	0.875	1	0.351	202.274	0.331	1595	630
0.09	174.4	1962	327.500	0.866	1	0.314	233.820	0.386	1570	589
0.14	172.4	1983	318.973	0.865	1	0.308	231.629	0.399	1590	581
0.45	151.3	2007	270.544	0.862	1	0.304	238.393	1.176	1483	503
0.90	138.6	1944	206.419	0.845	1	0.213	240.001	1.437	1384	459
1.80	128.2	2010	93.286	0.823	1	0.168	238.926	1.565	1315	422
3.60	119.5	1973	-125.868	0.776	1	0.140	240.001	1.965	1279	391
7.20	110.9	2000	-530.341	0.683	1	0.015	240.001	2.269	1261	357
14.40	110.3	2008	-1328.806	0.659	1	0.026	237.494	2.519	1255	355
28.80	110.7	1973	-2929.763	0.658	1	0.015	231.558	3.163	1253	357

Parameters: Power Cost = variable, Job Revenue = 0.17e, MaxRT = $2 \cdot RT_0$

Table III
SCHEDULING ILP SOLVER WITH DIFFERENT ELECTRIC COSTS

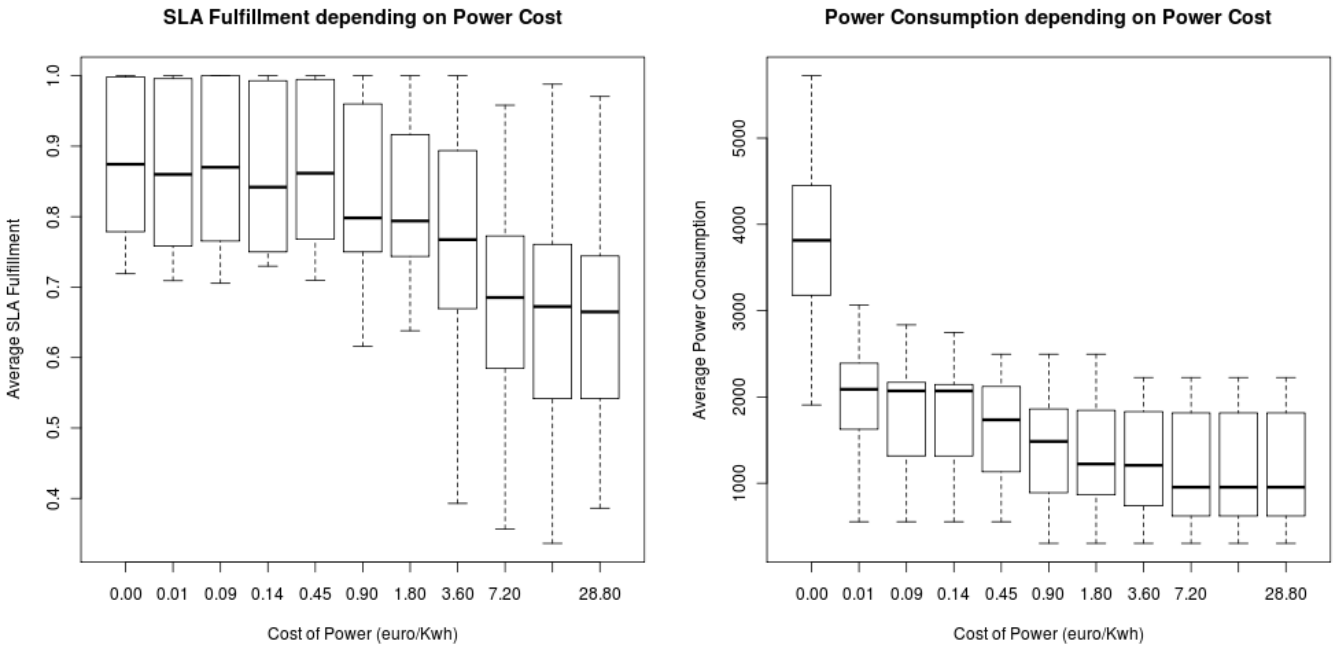

Figure 6. Power vs SLA fulfillment

Method	Power (w)	Migs	Profit (euro)	AvgQoS	MaxQoS	MinQos	UsedCPU	UsedHosts
Descending BestFit	203175.7	991	304.8858	0.860513	1	0.316643	2250	658
ILP Solver	194974.0	232	321.3193	0.877271	1	0.346642	1702	662

Parameters: Power Cost = 0.09 euro/Kwh, Job Revenue = 0.17 euro, MaxRT = $2 \cdot RT_0$, Migration Penalty = 0.014 euro

Table IV
SCHEDULING COMPARATIVE BETWEEN TECHNIQUES APPLYING MIGRATION PENALTIES

Table IV shows the results for the migration-aware versions of the exhaustive solver and BestFit. Comparing with Table II, it can be seen that migrations are drastically reduced, with only a small profit loss. For example, the ILP solver goes from 327 down to 321 euro profit but from 1962 to 232 migrations, in a 4 day period. Note that the greedy algorithm take less profit from this addition. Table V shows that applying the new migration policy, the sensitivity to job price and power costs still applies.

E. Discussion

As shown in the experiments, solving an Integer Linear Program can be an intractable problem, and depending on the kind of model to be solved using exhaustive search and heuristics do not work as good as with easy problems. Some tricks and methods can be applied in order to make the model lighter, and having a combinational problem [Hosts $\times$ Jobs $\times$ CPUquota per Job] we can attempt to reduce the range of each variable. E.g., if we dispose of an heuristic of function

Euro/Kwh	Power (w)	Migs	Profit (euro)	AvgQoS	MaxQoS	MinQoS	TimeSpent	Gap LB	UsedCPU	UsedHosts
0.00	365585.0	195	353.559632	0.883090	1	0.368497	94.885208	0.047113	4600	1150
0.01	242807.6	225	340.002485	0.879603	1	0.351277	115.189271	0.055512	1798	846
0.09	194974.0	232	321.319368	0.877271	1	0.346642	191.980312	0.213529	1702	662
0.14	170743.2	330	312.523225	0.878515	1	0.355842	216.122916	0.273056	1619	571
0.45	150265.3	445	264.770629	0.870027	1	0.310506	230.415729	0.637188	1492	498
0.90	137242.4	550	199.360765	0.857122	1	0.229214	234.020833	0.966561	1380	455
1.80	128062.9	600	81.188014	0.840108	1	0.190497	239.263333	1.574775	1327	422
3.60	120122.9	776	-136.816646	0.815860	1	0.155251	240.000312	1.964834	1285	392
7.20	115384.5	889	-559.543473	0.776640	1	0.101097	235.176979	2.647854	1274	374
14.40	110087.6	837	-1344.66673	0.737365	1	0.065805	232.421979	2.951466	1248	357
28.80	110451.0	913	-2933.12591	0.738801	1	0.085581	229.433333	3.560830	1251	356

Parameters: Power Cost = variable, Job Revenue = 0.17 euro, MaxRT = $2 \cdot RT_0$, Migration Penalty = 0.014 euro

Table V
SCHEDULING ILP SOLVER WITH DIFFERENT ELECTRIC COSTS AND MIGRATION PENALTY

telling the maximum number of hosts to be required given amount of jobs with a determined demand, we can reduce the size of the Host pool for that specific schedule problem. The same can be applied for the CPUquota per Job, as the minimum CPU required will be the given by our learned function f_{MinRes} (CPU demanded in a no-stress situation) and the maximum CPU can be obtained through the learned f_{RT} function to know how much CPU the job will demand to accomplish its RT in a full machine. Obviously, this size reduction will not improve the complexity of the problem, but will allow us to test it in a period of time considered "tractable", obtaining complete solutions and searching for lighter approximate algorithms.

In the problem we are focusing, the main factor that makes the model a "hard problem" is that the goal function depends on the resources given to the job but also the occupation of the host. This occupation is the sum of resources used by all the jobs in the host, and their interference and competition for them makes that giving resources to a job increases its SLA but degrades other jobs SLA. This occupation hides some variables to be specified in our next works, like virtualization overhead, usage of Input/Output devices, architectural overheads, etc. Knowing them will permit the model to separate the simple occupation overhead from specific variables easier to be handled.

In this work we tested some alternatives to the exhaustive solver. One of them is the approximated algorithm Best-Fit, compared against the exact solver in our experimentations. The Best-Fit finds solutions very near the optimal, with the only inconvenient of performing more migrations than the exact solver for each schedule round. This is due the fact that for each scheduling round, just that one job changes its requirements enough, the ordering of jobs can change enough to make a solution very different to the previous scheduling round one. Note that the descending order for job requirements is better than the ascending one minimizing the power consumption (the descending order is usually better for optimizing the original best-fit for knapsack problems [8]). Looking at the results we should think about how much important is this small profit difference between the Best-Fit results and complete exhaustive results, in front of the time taken to solve the model, and also the power saved (not in economic values but in power saving goals).

Another alternative tested was the Lagrange relaxation method [9], converting "hard constraints" into Lagrange factors. Although it could appear lighter than the exhaustive solver, the main problem found was that for each element (host, job or CPUquota range) several constraints must be added, and the number of lagrange factors to be solved increases turning the problem intractable with very few hosts and jobs. Finally another alternative, not tested here but plausible, would be to consider a "divide and conquer" method, splitting the set of hosts and jobs into small sets, solving each subset with an ILP. The given solutions could be optimal for each set in a tractable amount of time, and research should be focused on finding the best way to create these sub-sets in order to optimize the final sum of profits after scheduling (another NP-hard problem).

V. CONCLUSIONS AND FUTURE WORK

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 job 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 jobs and system behaviors towards resources and SLA metrics in an automatic manner through machine learning, and apply them to full-scale data-center models, letting schedulers and decision makers to have more adjusted estimation functions a priori, in order to make better their decisions for each system and kind of job.

Also, by focusing the data-center model towards factors to be optimized (like the economic ones, the service quality to clients, and the energy-related ones), a mathematical model can be built and be solved by different methods. General allocation problems like the presented job×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 with our model we can find optimal solutions according to the introduced policies, but paying a high cost in computation time. For this reason, other algorithms (heuristics and approximate) can be applied over the same model, obtaining, in the case of the ordered Best Fit, solutions enough close to the optimal in extremely low time.

Plans for future work include examining models and scenarios where the CPU and Memory are not the only critical resources, so the adaptive machine learning methods have to deal with a more complex space of attributes, identifying different situations and contexts. The effects of the memory behavior of web server platforms will be studied to model and predict that factors that affect the memory occupation. Also, 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 in order to solve the ILP model without applying solvers with high cost in time.

ACKNOWLEDGMENT

We would like to thank to Íñigo Goiri, Ferran Julià, Ramon Nou, J.Oriol Fitó and Jordi Guitart from UPC-BSC for lending us their testbed workbench in order to evaluate our methods. This work has been supported by the Ministry of Science and Technology of Spain under contract TIN2008-06582-C03-01.

REFERENCES

[1] Europe's energy portal. http://www.energy.eu.

[2] J. Berral, R. Gavaldà, and J. Torres. An Integer Linear Programming Representation for DataCenter Power-Aware Management, 2010. http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=1096.

[3] J. Berral, R. Gavaldà, and J. Torres. Li-BCN Workload 2010, 2011. http://www.lsi.upc.edu/ jlberral/documents/libcn-2011.pdf.

[4] J. Berral, Í. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavalda, and J. 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.

[5] J. S. Chase, D. C. Anderson, P. N. Thakar, and A. M. Vahdat. Managing energy and server resources in hosting centers. In *In Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP*, pages 103–116, 2001.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms, second edition, 2001.

[7] G. Dhiman. Dynamic power management using machine learning. In *In Proceedings of the 2006 IEEE / ACM International Conference on Computer-Aided Design*, 2006.

[8] G. Dósa. The tight bound of first fit decreasing bin-packing algorithm is ffd(i)=(11/9)opt(i)+6/9. In *In ESCAPE*, 2007.

[9] M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. 27(1):1–18, Jan. 1981.

[10] J. O. Fitó, Í. Goiri, and J. Guitart. SLA-driven Elastic Cloud Hosting Provider. In *Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP'10)*, pages 111–118, 2010.

[11] Í. Goiri, J. Guitart, and J. Torres. Characterizing Cloud Federation for Enhancing Providers Profit. In *3rd International conference on Cloud Computing (CLOUD 2010)*, pages 123–130, 2010.

[12] Í. Goiri, F. Julià, J. Ejarque, M. De Palol, R. M. Badia, J. Guitart, and J. 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.

[13] Í. Goiri, F. Julià, R. Nou, J. Berral, J. Guitart, and J. Torres. Energy-aware Scheduling in Virtualized Datacenters. In *12th IEEE International Conference on Cluster Computing (Cluster 2010)*, 2010.

[14] GUROBI. Gurobi optimization, 2011. http://www.gurobi.com/.

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[16] K. Hornik. The R FAQ, 2010. ISBN 3-900051-08-9.

[17] F. Julià, J. Roldàn, R. Nou, O. Fitó, Vaquè, G. Í., and J. Berral. EEFSim: Energy Efficency Simulator, 2010.

[18] I. Kamitsos, L. Andrew, H. Kim, and M. Chiang. Optimal Sleep Patterns for Serving Delay-Tolerant Jobs. In *1st International Conference on Energy-Efficient Computing and Networking (eEnergy'10)*, 2010.

[19] L. Liu, H. Wang, X. Liu, X. Jin, W. He, Q. Wang, and Y. Chen. GreenCloud: a New Architecture for Green Data Center. In *6th International Conference on Autonomic Computing and Communications*, 2009.

[20] R. Nathuji, K. Schwan, A. Somani, and Y. Joshi. Vpm tokens: virtual machine-aware power budgeting in datacenters. *Cluster Computing*, 12(2):189–203, 2009.

[21] V. Petrucci, O. Loques, and D. Mossé. A Dynamic Configuration Model for Power-efficient Virtualized Server Clusters. In *11th Brazillian Workshop on Real-Time and Embedded Systems (WTR)*, 2009.

[22] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[23] Y. Tan, W. Liu, and Q. Qiu. Adaptive power management using reinforcement learning. In *International Conference on Computer-Aided Design (ICCAD '09)*, pages 461–467, New York, NY, USA, 2009. ACM.

[24] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *In Proc. of ICAC-06*, pages 65–73, 2006.

[25] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, Germany, 2001.

[26] W. Vogels. Beyond server consolidation. *Queue*, 6(1):20–26, 2008.