Empowering Automatic Data-Center Management with Machine Learning

Josep Ll. Berral, Ricard Gavaldà, Jordi Torres
Universitat Politècnica de Catalunya and Barcelona Supercomputing Center
Jordi Girona 31, 08034 Barcelona, Spain
berral@ac.upc.edu, gavalda@lsi.upc.edu, torres@ac.upc.edu

Abstract

The Cloud as computing paradigm has become nowadays crucial for most Internet business models. 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 (hardware, applications, workloads, customer needs...). Here we show how a combination of scheduling algorithms and data mining techniques helps improving the performance and profitability of a data-center running virtualized web-services. We model the data-center's main resources (CPU, memory, I/O), quality of service (viewed as response time), and workloads (incoming streams of requests) from past executions. We show how these models to help scheduling algorithms make better decisions about job and resource allocation, aiming for a balance between throughput, quality of service, and power consumption. We test our approach first with real data and web-services on a data-center simulator, and further validate it in a real execution on a reduced scale cluster running the OpenNebula virtualization platform.

1 Introduction

Cloud Computing, the new paradigm of distributed and clustered computing, has become a crucial model for the Internet architecture towards the externalization of information and IT resources for people and organizations. It brings the possibility of offering "everything as a service" (platform, infrastructure, and services), allowing companies to move their IT, previously in private owned data-centers, to external hosting. But the control and optimization of these infrastructures is not easy, as many elements and actors interact and influence its profitability, performance, and power and resource consumption.

We distinguish three main actors: The *data-center 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 she has a certain freedom to allocate his customer's tasks to physical and virtual resources over time. 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 (QoS) or Service Level Agreements (SLA) that they deem sufficient for their desired level of client satisfaction. An example customer would be the owner of an e-shop wishing to offer the shop as a web-service. *Clients* are remotely connecting to the customer's web-services, e.g., browsing the e-shop and possibly purchasing from it. In this work we mostly take the manager's view and explore the following trade-off: we must provide sufficient resources to the customer's web-service to meet the agreed QoS and SLA, but no more, as any allocated resource beyond that incurs in extra costs, mainly power consumption.

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 uptime, response time and availability). Here we model the cloud scenario as a set of data-center resources and a set

of web services, each resource with a maximum quota of usage and energy requirements, and each service with resource requirements (load per time unit), performance requirements, and an execution reward. The management and decision making consists on placing each service on a hosting machine that assures the availability of the required resources, and that the other services being hosted in the host do not compete excessively for these available resources. A good strategy used often is *consolidation*: attempting to set the maximum number of services in the least viable amount of hosting machines, so the number of running machines and resources is minimized. Consolidation has been made much easier with the emergence of *virtualization,* a mechanism to box jobs in virtual machines that can run in mutual isolation in the same physical machine and moved across machines when necessary.

The decision-making process for optimal allocation of tasks and resources should obviously improve if we can exploit the information from the system status to make predictions about future evolution and about the consequences of our decisions. As many of the parameters and functions involved in this optimization problem are unknown *a priori* and vary over time, explicit modeling is very difficult, and data mining and machine learning methods are a more viable option. We will use these techniques to create models from past experience, and in particular models for each element in the system (an application type, a workload, a physical machine, a high-level service requirement). We can then plug these models into traditional (or not so traditional) scheduling techniques for optimizing system performance according to our goals. Additionally, model building should not take place once and for all, we should keep them updated, and revise them using real execution data.

In this work we present a methodology for using machine learning techniques (ML) to model the main resources of a web-service based data-center from low-level information, and learn high-level information predictors to drive decision-making algorithms for virtualized job schedulers, without much expert knowledge or real-time supervision. For a given web service application, we build models to predict features such as minimum CPU usage, memory occupation and bandwidth status, and dependence on workload volume. For quality of service elements, such as response time, a 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 is to decide the effects of resource allocations on hosts using consolidation towards services QoS and on power consumption. To test the approach we prepare ML models of CPU, memory, and bandwidth from a real system and real workloads; we compare a set of ad-hoc and off-the-shelf algorithms for scheduling, with and without using predictive ML functions in a realistic simulator. Then we also validate the algorithms and models on a real data-center using the virtualization platform OpenNebula [13].

This work is organized as follows: Section 2 presents previous work in this area. Section 3 describes the architecture of the scenario. Section 4 shows the models and algorithms used. Section 5 explains the machine learning models. Section 6 describes the models and experiments for quality of service prediction. Finally, Section 7 summarizes conclusions and future work.

2 Related Work

There are relevant previous works on autonomic computing and data-center management using machine learning techniques to predict behaviors or determine usefulness of policies. Approaches like the ones presented by Vengerov et al. [19], Tesauro et al. [16], Tan et al. [15] and Kamitsos et al. [11], use Reinforcement Learning algorithms to manage resource allocation and power consumption; specifically techniques like Q-learning, SARSA and MDP explained in Sutton et al. [14], to select policies to be applied at each time. Other works focus on resource status prediction, like Dhiman et al. [6] applying ML to select policies for specific resources like hard disk and network states. Andrezejak et al. [2] use fuzzy logics to predict resource usage, and Vienne et al. [20] use RL for predicting levels of quality of service on data-center resources management. Also, ML is also used to detect failures on resources and manage fall-back policies. Alonso et al. [1] apply ML and regression methods to learn to detect memory bugs from hosting machines, and predict the time to crash of it; and works like Hamerly et al. [9] predict failures for disk drives using Bayesian predictors. 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 are focusing on learning resource models and environment models, using them to supply accurate on-line information to decision makers.

Most of the current works on modeling the cloud present ad-hoc systems or specific expert knowledge. Chase et al. presented MUSE [5], a framework for modeling and autonomically control cloud hosting centers and its policies. The approach is based on an economical managing 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. Also, other works like Goiri et al. [7] present a similar way to model the cloud, where each policy to be applied on jobs and resources is represented by a set of conditions with rewards and penalties, based on the job objectives and the resource capabilities. The modeling in their approach focuses on Service Level Agreement (SLA) between the resource provider and the customer. Based on the idea of modeling resources and requirements from these works, Berral et al [3] modeled a data-center using a mathematical program, used to optimize virtual machine scheduling on a data-center from learned resource behaviors and jobs requirements. Our work here goes substantially beyond that, since then we considered only one resource (CPU usage) while here we model all the main resources on the data-center. 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. For example, 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. Additionally, we test our approach both by simulation and on a small but totally real environment, instead of only on a simulated one.

All the current approaches using ML for resource management, as far as we know, are oriented towards learning the characteristics of specific components or towards evaluating the consequences of using policies on given states of the cloud or its elements for HPC jobs. 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.

3 Managing Data-Centers

3.1 Service Data-center Architecture

Typical commercial data-centers 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 (SLA), negotiated by customers to accommodate their clients requests at sufficient satisfaction level. The data-center contains physical computing machines (PM), and each one holds virtual machines (VM) 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. Figure 1 shows the business infrastructure.

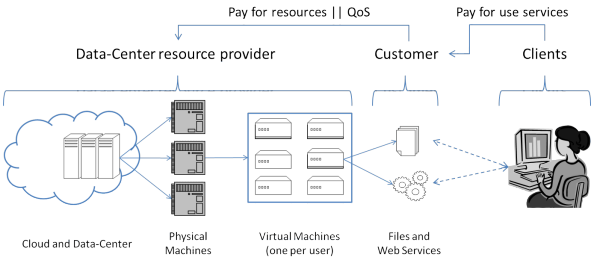

Figure 1: Data-center business infrastructure

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 goal is to maximize the profit of each VM fulfilling the QoS 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 (RT) for web

queries, the web site uptime 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 uptime 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 data-center (CPU, memory and input/output network).

3.2 Information Monitoring

Middleware software such as OpenNebula [13], Eucaliptus [12] or EmotiveCloud [17] is typically used in cloud-like architectures in order to manage PMs, VMs, network elements and traffic. We will rely on the existence of some such middleware both for sensing (collecting high- and low-level data) and for acting (managing tasks, workloads, and VM and PM resources). Figure 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 a Director Machine, reads all monitored lectures and makes decisions on a MAPE control loop [5]; also an agent on the Gateway receiving orders to redirect traffic and also monitor itself as part of a possible traffic bottleneck, also monitor RT (and so QoS) for each VM.

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

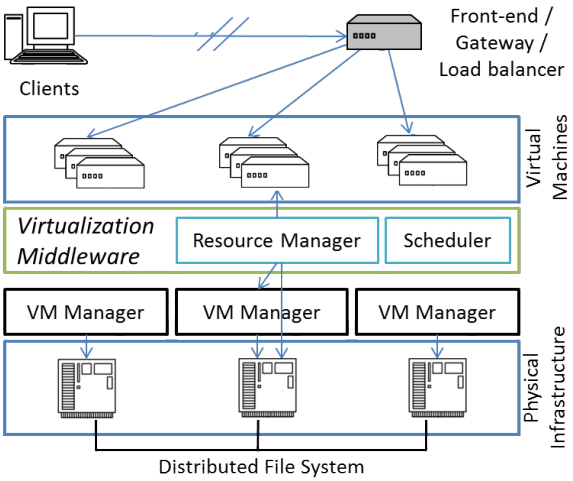

Figure 2: Virtualization middleware schema

times; average requested bytes; and resource usage and bandwidth.

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.

3.3 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 (ML) methods. ML is a subfield of Data Mining in charge of creating (mostly predictive) system models automatically from real examples of observed past behaviors. We base our work in a ML hypothesis:

Hypothesis 1 *For each situation, there may be 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 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. So in this work we use these ML modeling and prediction techniques in order to obtain accurate models of the data-center we are managing, and predict information and behaviors to manage it properly.

4 Methodology

4.1 Framework Schema

The first contribution of this work 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.

Figure 3 shows the information flow and elements composing our decision making schema. 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.

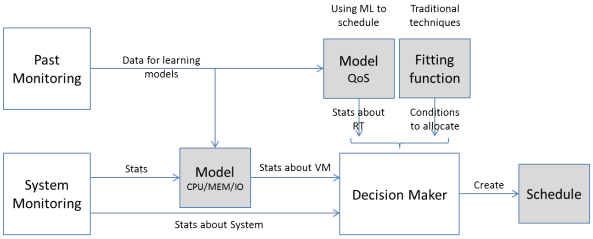

Figure 3: Information flow schema using models

4.2 Architecture Details

Measuring the QoS. Quantifying the quality of service as perceived by the end client is a highly complex issue, with psychological factors as important as technological ones. It is clear, though, that Response Time (RT), the amount of time our data-center requires to reply a request, is a central ones. Indeed RT is typically a main character in Service Level Agreements, which are in some sense the contracts where provider and customer commit to a common notion of QoS. Here we make the simplifying assumption that SLA fulfillment, is strictly a function of RT; further factors such as uptime 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 data-center domain and network, not at the client side since he may use unpredictable thinking time and he may have a slow machine or connection at his end.

A common "Response Time to QoS function" in SLAs is to set a threshold α and a desired response time RT_0, and set SLA fulfillment level to

$$SLA(RT) = \begin{cases} 1 & \text{if } RT \leq RT_0, \\ 1 - \frac{RT}{\alpha \cdot RT_0} & \text{if } RT_0 \leq RT \leq \alpha \cdot RT_0, \\ 0 & \text{if } RT > \alpha \cdot RT_0 \end{cases}$$

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.

Power Consumption. Several costs can be considered from running a data-center (construction, maintenance, power consumption, ...), but the most important directly related to the resource usage is power consumption. Multiprocessor computers have the advantage that their energetic consumption depends primarily on the CPU usage, and once on-line, increasing load does not make consumption to grow linearly. E.g. in a Intel Xeon 4-CPU machine, the power consumption (in Watts/hour) when all CPUs are in idle state is 235, and 267.8, 285.5, 302.5, and 317.9 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) processors and shutting down the second one. This explains the potential for power saving by consolidation.

4.3 Scheduling Algorithms

Following the schema from MUSE [5] and Berral et al., [3], the data-center 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. The details are omitted here, for space reasons. They are similar to [3], except that, as mentioned, we consider 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. Here we use approximate, heuristic, and faster algorithms: the generic for bin packing problems, Ordered First-Fit and Best-Fit algorithms [18]. We also use two the BackFilling and λ-Round Robin algorithms [7], specialized for load-balancing via consolidation. Algorithms are depicted in Algorithms [1,2,3,4].

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.

5 Learning Models

The following subsections explain each modeling and Figure 1 shows the numeric results for each one.

Algorithm 1 λ-Round Robin algorithm

```
for each vm i:
    get_data(i);
    res_quota[i] <- get_required_resources(i);
for each host j:
    res_avail[j] <- get_total_resources(j);
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;
    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);
```

Algorithm 2 Backfilling algorithm

```
for each vm i:
    get_data(i);
    res_quota[i] <- get_required_resources(i);
for each host j:
    get_data(j);
    res_avail[j] <- get_available_resources(j);
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;
            k--;
```

Algorithm 3 Descending First-Fit algorithm

```
for each vm i:
    get_data(i);
    res_quota[i] <- get_required_resources(i);
for each host j:
    res_avail[j] <- get_total_resources(j);
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;
```

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 popular WEKA [8] machine learning package for all our learning tasks. We tested different regression methods (LinReg, SVM, M5P...) for each model, with tuned parameters, selecting the simplest model among the ones that bring the best or almost-best results.

Algorithm 4 Descending Best-Fit algorithm

```
for each vm i:
    get_data(i);
    res_quota[i] <- get_required_resources(i);
for each host j:
    res_avail[j] <- get_total_resources(j);
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;
    assign_vm_to_host(c_host,v);
    update_resources(res_avail[c_host],v);
```

5.1 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 $E[timepr]$, and the average number of bytes exchanged per request, $E[bytespr]$. The function to be learned, one for each type of PM in the data center, 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.

In order to learn this function, we used a M5P algorithm [8], a decision tree holding linear regressions on its leaves. The choice makes sense, as CPU consumption may be in significantly different load regimes, but reasonably linear in each.

5.2 Memory Prediction

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 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. 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 $E[requests]$, $E[bytespr]$, $E[timepr]$; we have investigated a simple model

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

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.

5.3 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 data-centers, implying that the usage of disk requires network.

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. After experimenting with several different models (LinReg, decision trees, SVMs), we again selected the tree-of-models M5P algorithm because of its results.

5.4 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.

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.

6 Experiments

6.1 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.

A set of Intel Xeon 4 Core machines (3Ghz, 16Gb RAM) have been used as test-bed, running jobs of kind [Apache, PHP, MySQL] in a virtualized environment [Ubuntu Linux 11.10 SE, VirtualBox v3.1.8]. Also, to test the ML-augmented algo-

	ML Method	Training	Validation	MRE	MAE	StDev	Data range
Predict CPU	M5P ($M = 50$)	3968 inst	7528 inst	0.164	2.530%	4.511	$[2.37, 100.0]$% CPU
Predict MEM	Linear Reg.	107 inst	243 inst	0.0127	4.396 MB	8.340	$[124.2, 488.4]$ MB
Predict IN	M5P ($M = 30$)	1623 inst	2423 inst	0.193	926 Pkts	1726	$[56, 31190]$ #Pkts
Predict OUT	M5P ($M = 30$)	1623 inst	2423 inst	0.184	893 Pkts	1807	$[25, 41410]$ #Pkts
Predict RT	M5P ($M = 4$)	38040 inst	15216 inst	0.00878	9.9 ms	0.0354	$[0, 2.78]$s, $\widehat{RT}$ 17ms

Table 1: Learning details for each predicted element. All training processes are done using random split of instances (66/34)

rithms, we performed a full test using the workload against an analytic simulation of our data-center, a R version of the cloud simulator EEFSIM made by Julià et al. [10]. The simulator allowed us to test different configurations and models before validating the model on a real data-center, also recreate big scenarios easily.

The workload used corresponds to the Li-BCN Workload 2010 [4], a collection of traces from different real hosted web-sites offering services, from file hosting to forum services (deployed on Apache v2.1 with an hybrid architecture). These web-applications correspond to a set of customers running their services in a virtualized environment.

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 data-center 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 data-center 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 $RT \geq 2RT_0$).

6.2 ML-augmented scheduling algorithms

In the following experiments we compare the λ-RR algorithm, 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 $mempm_h + memvm_{vm} \leq MaxMEM_h$.

These comparative experiments are simulator-based, running 20 VMs containing web-services in a data-center 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 2 and Figure 4 show the results.

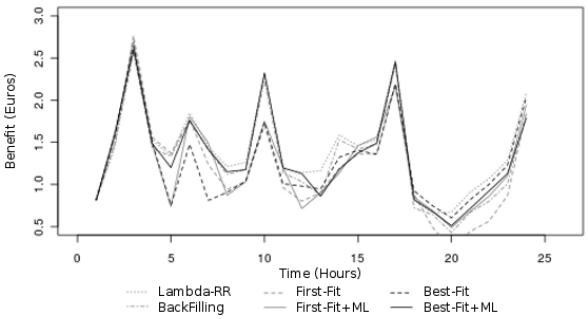

Figure 4: Evolution of the revenue per algorithm

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

	Benefit (euro)	Avg.Consumption (watt)	Avg.QoS	Total Migrations	Avg.Used PMs/hour
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 2: Comparative of algorithms from the relevant business model values

configurations for this kind of data-center, calculated in [7]. 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.

6.3 Validation on Real Machines

After the initial experimental check on the simulated data-center, we moved 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 data-center nodes, 1 as gateway and 1 attacking machine reproducing client requests, in a different data-center 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 5 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.

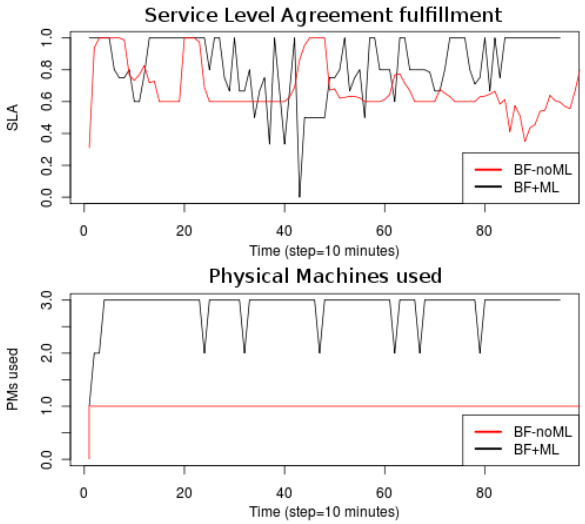

Figure 5: BF-noML against BF+ML SLA (based on response time) and machines used

We observe that the ML-augmented versions of Best-fit and First-fit improve the final figure (rev-

enue) 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, for space reasons, 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 [3]. 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 Conclusions

In this work we presented a methodology for modeling cloud computing resources of a web-service based data-center 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. The models and schedulers have been trained in a real data-center using as input real web-service traces, tested in a simulation environment to compare behaviors in large scale data-centers, 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 deconsolidating by predicting the required computing resources and the resulting RT for a given schedule.

Next steps will focus on scalability and on hierarchically modeling the cloud system as a set of data-centers where services can not only move between machines but among locations around the world. Also we will focus on the network side, including the service time DC-client as another SLA object, bringing the services near their demand. Finally, in the future this technique and research will move towards on-line machine learning, as models will be created on the fly, when new environments (machines, networks, ...) and new kind of web-services enter into the system to be managed.

Acknowledgments

Thanks to *RDlab-LSI* for their support. This work has been supported by the Spanish Ministry of Science 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 (2009-SGR-1428).

References

[1] J. Alonso, J. Torres, J. L. Berral, and R. Gavaldà. Adaptive on-line software aging prediction based on machine learning. In *IEEE/IFIP Intl. Conf. on Dependable Systems and Networks (DSN 2010)*, 2010.

[2] A. Andrzejak, S. Graupner, and S. Plantikow. Predicting resource demand in dynamic utility computing environments. In *Intl. Conf. on Autonomic and Autonomous Systems (ICAS '06)*, 2006.

[3] J. Berral, R. Gavaldà, and J. Torres. Adaptive Scheduling on Power-Aware Managed Data-Centers using Machine Learning. In *Intl. Conf. on Grid Computing (GRID 2011)*, 2011.

[4] J. Berral, R. Gavaldà, and J. Torres. Li-BCN Workload 2010, 2011. http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=1099.

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

[6] G. Dhiman. Dynamic power management using machine learning. In *IEEE/ACM Intl. Conf. on Computer-Aided Design 2006*, 2006.

[7] 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.

[8] 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.

[9] G. Hamerly and C. Elkan. Bayesian approaches to failure prediction for disk drives. In *Intl. Conf. on Machine Learning (ICML'01)*, San Francisco, CA, USA, 2001.

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

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

[12] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. 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.

[13] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14–22, Sept. 2009.

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

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

[16] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Intl. Conf. on Autonomic Computing (ICAC 2006)*, 2006.

[17] A. Vaqué, Í. Goiri, J. Guitart, and J. Torres. Emotive cloud: The bsc's iaas open source solution for cloud computing, 2012-01-31 2012.

[18] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[19] D. Vengerov and N. Iakovlev. A reinforcement learning framework for dynamic resource allocation: First results. In *Intl. Conf. on Autonomic Computing (ICAC 2005)*, 2005.

[20] P. Vienne and J.-L. Sourrouille. A middleware for autonomic qos management based on learning. In *Intl. Wksp. on Software Engineering and Middleware*, 2005.