Power-aware Multi-DataCenter Management 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*—The cloud relies upon multi-datacenter (multi-DC) infrastructures distributed along the world, where people and enterprises pay for resources to offer their web-services to worldwide clients. Intelligent management is required to automate and manage these infrastructures, as the amount of resources and data to manage exceeds the capacities of human operators. Also, it must take into account the cost of running the resources (energy) and the quality of service towards web-services and clients. (De-)consolidation and priming proximity to clients become two main strategies to allocate resources and properly place these web-services in the multi-DC network. Here we present a mathematical model to describe the scheduling problem given web-services and hosts across a multi-DC system, enhancing the decision makers with models for the system behavior obtained using machine learning. After running the system on real DC infrastructures we see that the model drives web-services to the best locations given quality of service, energy consumption, and client proximity, also (de-)consolidating according to the resources required for each web-service given its load.**

I. Introduction

The current leading paradigm of distributed computing, known as Cloud Computing, has become crucial for the externalization of IT resources for business, organizations and people. The possibility of offering "everything as a service" (platform, infrastructure and services), allows companies to move their IT needs to external hosting, reducing costs as they pay only for resources they use. E.g. AmazonWS [2] offers virtualized resources to run web-services, with total transparent view of the infrastructure to their customers. Naturally, providers want in turn to optimize the use of the resources they have deployed with their own metrics. Because of the volume, heterogeneity, and complexity to be managed, this has become today a hard optimization problem. It is even harder for a typically provider who owns a multi-DC system, usually distributed through the world, and must balance response times, data and task location, and energy consumption.

Any management policy starts by identifying the factors to be optimized, in our case revenues and costs. Revenues come from servicing the clients of the hosted web-services with reasonable Quality of Service (QoS), and costs are mainly operational costs for the infrastructure (e.g. energy consumption). Energy-related costs (powering machines and disks, cooling, ...) have become nowadays a major cost factor for IT. These costs are reflected in electricity consumption, which grows linearly or more with the capacity of the datacenter, and also with environmental impact [16], social and government pressure, and public image. *Consolidation* is a common strategy used to save power: set the maximum number of services in the least viable amount of hosting machines, so the number of on-line machines and resources is minimized. Virtualization technology has made consolidation easier, as web-services are boxed in Virtual Machines (VMs) running in mutual isolation in the same Physical Machine (PM), and migrated when necessary.

Management at short time scales is today typically automated, as human operators cannot cope with the size of the resources to be managed and the speed at which decisions must be made. Autonomic computing methods typically work by using (or building) some model of the system, collecting information about the current state, and then combining model and observations to make online decisions. In this paper we follow a line of work in which machine learning and data mining methods are used to build the model. The idea is to automate and improve the process even more, by building models with a level of detail that the datacenter designer cannot reach, or by keeping the model updated as hardware, software, and demands change over time.

Unlike previous work, we consider management at the "cloud" or multi-DC scenario level, that is, joint optimization across a set of cooperating DataCenters (DCs), each one with its own resources and receiving requests to host virtualized web-services. DCs are interconnected to migrate VMs among them and transport client traffic between VMs and clients. They must keep the client-VM communication transparent to clients, so that clients do not have to know the internals of the VM hosting system. VM placement must provide each VM with the resources to satisfy the load allocated to it, provide proximity to the client if possible (to minimize network latency, hence response time), and minimize overall energy consumption, which may vary among geographically distant DCs and over time. Additionally, it must also be sensitive the latencies and bandwidth use incurred when VMs are migrated within or among DCs.

Here we model multi-DC system management, to schedule virtualized web-services within DCs and across DC networks, using energy consumption, resource allocation and QoS as decisive factors, as a mathematical optimization problem. We use machine learning / data mining techniques for predicting the effect of VM placement moves before actually performing them, without relying on expert knowledge or requiring human supervision in real time. In particular, such techniques let us map low-level, input metrics (such as allocated CPU, memory,

0190-3918/13 $26.00 © 2013 IEEE
DOI 10.1109/ICPP.2013.102

CPS Conference Publishing Services

or bandwidth of a tentative placement) to high-level, output metrics (such as response times or energy consumption). We previously studied these techniques for local DC management successfully [6], [8], [10], and here we scale to multi-DC systems taking into account new relevant factors like service-client proximity, migration overheads, energy costs at different locations, and modularity between inter-DC relations and information. The main contribution of this paper is thus the hierarchical extension of those techniques to the multi-DC scenario.

To test the approach we build, using machine learning (ML), predictive models of CPU, memory, bandwidth, response times and Service Level Agreement (SLA) fulfillment from a real system and workloads; we compare a previously studied method based on the Best-Fit algorithm for scheduling [8], with and without using the predictive models, to test each prediction and the benefits of using learned models; and finally we study the performance of our formulation in terms energy, latencies, and QoS factors on a real DC environment using the OpenNebula [19] virtualization platform.

This work is organized as follows: Section II presents previous work in this area. Section III explains the multi-DC business model. Section IV describes the mathematical model, the algorithms to be used and the methodology and study of learning and prediction of components and QoS. Section V shows the experiments performed to study the approach. Finally, Section VI summarizes conclusions and future work. A list of acronyms is given at the end.

II. Related Work

Previous works on DC management use machine learning techniques to predict behaviors and select policies to be applied. Works like [23], [21], [20], [15], focused on energy-aware management, use Reinforcement Learning (RL) algorithms to decide resource allocation policies, including consolidation techniques. Other works like [24] use RL for predicting levels of quality of service on datacenter given a policy, or works like [4] use fuzzy logic to predict resource usage and make decisions. As far as we know, current approaches are oriented towards learning the consequences of using policies that depend on states of the multi-DC environment. Here we focus on learning resource models and environment models through machine learning, to supply decision makers with information that is unknown or uncertain.

Most of the current works on modeling the cloud discuss specific systems or model more general systems "by hand", which is expensive in expert time. In the framework MUSE [9] we find a mathematical model for autonomic automatically scheduling jobs to resources in datacenters, in order to optimize an economic objective function. In previous works, and in this one, we share their idea of a mathematical program to match resources, workloads and jobs. In [6], we considered the optimization of a single datacenter, considering only one basic resource: CPU usage; later in [8] we introduced machine learning techniques to model the simultaneous usage of several resources (CPU, memory, and I/O) and their relation to higher-level metrics such as response time and quality of service, to be used as oracles driving the decision maker. Here, we focus now in distributed DC systems where the global manager does not have full information of intra-DC level metrics that the local DC managers has; the approach is thus hierarchical or modular. Furthermore, we study the approach on an architecture where resource are more limited, thus there is more competition: In previous works we used HPC architectures, while in this one we use energy-aware architectures.

Load distribution among datacenter networks is also an important aspect to be dealt with. Virtualization technology allows an easy management and migration of jobs among hosting machines and DCs [18], and orchestrating this management on DCs and multi-DC networks is currently a challenging topic (see [12], [5]). In [25], migration strategies for virtualized jobs and hotspots are discussed within the Sandpipe framework. Other works focus explicitly on balancing load by following renewable energies, like [17] and [11], where optimization focuses on moving load to where renewable energy is available at each moment. Here we apply virtualization on our system to migrate the load across a worldwide distributed multi-DC network, balancing DC-user proximity vs. migration costs vs. energy consumption; a 'follow the sun/wind" policy could also be introduced easily into the energy cost computation.

III. Managing Multi-DCs

A. Business Model for Multi-DCs

Companies offering computational power or web hosting (e.g. Amazon [2]) base their business on offering customers resources from their multi-DC system for running web-services (often virtualized). Customers pay the provider according to a Service Level Agreement (SLA), generally focused on Quality of Service (QoS) objects toward the web-service clients. Usually this QoS depends on the amount of resources (CPU, Memory, IO...) granted to each VM (hence to the web-service), but these resources have a cost in running them (energy, maintenance, cooling, ...). The provider goal is to ensure the agreed QoS for the VMs, while minimizing the costs by reducing the resource usage. The business infrastructure is shown on Figure 1.

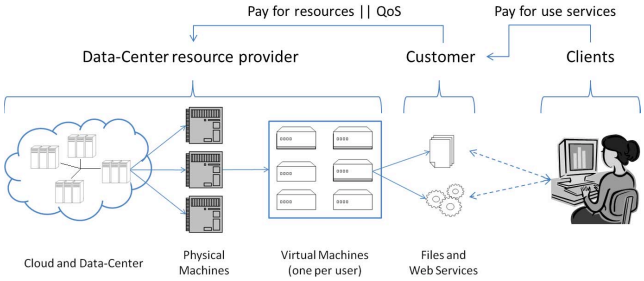

Figure 1. Multi-DC business infrastructure

A typical QoS measure on web sites is Response Time (RT). RT is affected by the time to process a client request and

dispatch it, also the proximity of the service to the client. The time to process and dispatch is affected by the resources dedicated to the VM and the load towards the web-service (number of requests competing for it) at that same time. A VM receiving insufficient resources will be slower to reply to user requests, but over-granting resources past a certain point will not necessarily speed-up the replies. Proximity to the client depends on the localization of the client requesting the web-service, the placement of the required VM holding the web-service, and the connection between the DC with the client. Here we consider that client requests going to non-local DCs can pass through our inter-DC network, while the client is connected through his/her local DC [1].

Finally, thanks to VM migration, web-services can be moved using a "follow the X" strategy. On green energy aware systems, VMs follow the sun, or the solar and wind production, minimizing the usage of "brown" energy. Other systems, like ours, use a "follow the load" policy, moving VMs close to their clients to provide good response times, unless local host overload or a locally high cost of energy forces otherwise. As mentioned, it would be easy to add to our policy a preference for locally available renewable energy over brown energy.

B. Collecting Information

In cloud-like architectures oriented to multi-DC infrastructures, middleware are in charge of managing VMs and PMs in an autonomic way, following policies and techniques based in the "Monitor, Analyze, Plan, Execute" schema [9]. We rely on such middlewares both for collecting high- and low-level data (monitoring), and for managing VMs and PM resources (executing). Figure 2 shows the typical multi-DC middleware infrastructure. This software controls each VM resource sharing by monitoring PM resources and adjusting VM placements and quotas, a decision maker reads all monitored information and makes decisions based on its given policies and functions to be optimized, also a gateway agent redirecting traffic and monitoring the RTs (and so QoS) for each web-service.

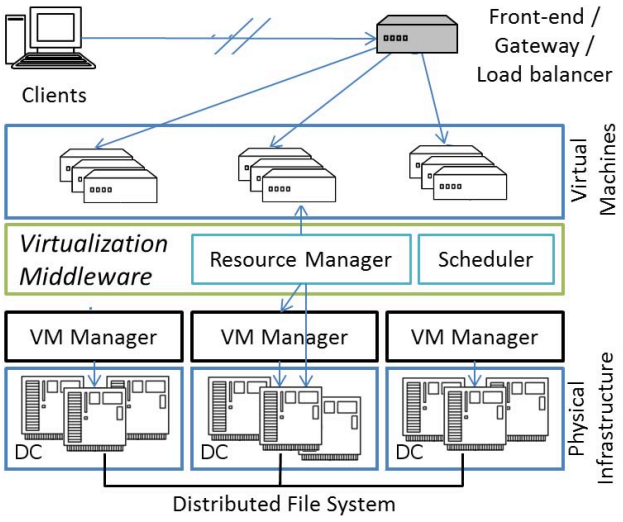

Figure 2. Virtualization middleware schema

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

Handling all this information becomes difficult the larger, more distributed, and loaded the system becomes. For this reason multi-DC systems management tend to decentralize, allowing each DC to administer their PMs and VMs, transferring VMs across DCs only when required. Here we propose allowing each DC to deal with its VMs and resources (as shown on [8]), bringing to the global scheduler information about the offered or tentative host where each VM may be placed for each DC. This modularity lets the global scheduling to drive the multi-DC by load sources, energy costs, and also predicting QoS using the provided host information for such DC; and after locating the VM into a DC, the local DC will decide if it reallocates properly the VM inside it or (de-)consolidates intra-DC.

C. Service Level Agreements

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

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

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

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.

Maximize:

$Profit = \sum^{i \in \text{VM}} f_{revenue}(SLA[i]) - \sum^{i \in \text{VM}} f_{penalty}(Migr[i], Migl[i], ISize[i]) - \sum^{h \in \text{PM}} f_{energycost}(Power[h])$

Output:

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

Parameters:

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

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

$pastSched$[PM, VM], previous schedule

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

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

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

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

Constraints:

1) $\forall i \in \text{VM} : \sum^{h \in \text{PM}} Schedule[h, i] = 1$

2) $\forall h \in \text{PM} : \sum^{i \in \text{VM}} GivenResources[i] \cdot Schedule[h, i] \leq Resources[h]$

3) $\forall h \in \text{PM} : Power[h] = f_{Power}(\sum^{i \in Schedule[h,\_]} GivenResources[i])$

4.1) $\forall i \in \text{VM} : Migr[i] = \lceil \sum^{h \in \text{PM}} (Schedule[h, i] \oplus pastSched[h, i]) \rceil^{1}$

4.2) $\forall i \in \text{VM} : Migl[i] = \sum^{h_1, h_2 \in \text{PM}^2} Schedule[h_1, i] \cdot pastSched[h_2, i] \cdot LatencyHH[h_1, h_2]$

5.1) $\forall i \in \text{VM} : ReqRes[i] = f_{RequiredResources}(VM_i, Load[i, \_])$

5.2) $\forall i \in \text{VM} : GivenRes[i] = f_{Occupation}(RequiredResources[i], Schedule[i, h])$

6.1) $\forall i \in \text{VM} : RT_{process}[i] = f_{RT}(Load[i, \_], RequiredResources[i], GivenResources[i])$

6.2) $\forall \langle i, l \rangle \in \langle \text{VM}, L \rangle : RT_{transport}[i, l] = \sum^{h \in \text{PM}} LatencyHL[h, l] \cdot Schedule[h, i]$

6.3) $\forall \langle i, l \rangle \in \langle \text{VM}, L \rangle : RT[i, l] = RT_{process}[i] + RT_{transport}[i, l]$

7) $\forall i \in \text{VM} : SLA[i] = f_{SLA}(RT[i, \_], RT_{0i}, \alpha_i)$

Figure 3. Mathematical Model

IV. Modeling The System

A. Mathematical Approach

The mathematical model shown in Figure 3 represents the constraints in our system and the metric to be maximized.

Desired output (solution): The schedule, containing which PM must hold each VM.

Objective Function: Maximize the sum of:

- income from customers for executed VMs. The $f_{revenue}()$ is agreed between provider and customer depending on the SLA.
- minus the penalties for SLA violation when migrating. The $f_{penalty}()$ is also agreed between provider and customer about how migrations must be penalized.
- minus the energy costs, as the sum of energy consumed by all on-line machines. The $f_{energycost}()$ is agreed between resource provider and energy provider.

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

Problem Parameters: Host (PM) Resources: CPU, Memory, and Bandwidth characteristics per PM; Job Load: Amount of load (number of requests, average bytes per request, average CPU process time per request, etc) for each different topological load source; the Previous Schedule; Latencies: latency between each load source and each PM (PMs in the same DC will have the same values), also latency between any two hosts; Image Size: size of VM images, to calculate the time required for migrating a VM; Baseline Response Time (RT_0) and Tolerance Margin (α): The two parameters in the SLA describing its fulfillment according to the resulting RT.

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

job, from the RT obtained, the basic RT and tolerance margins agreed with the customer. This function can be used over each request or over the average RT (weighting the different load sources).

Power consumption in multi-core computers depends non-linearly on the number of active cores and CPU usage. E.g. in a Intel Atom 4-Core machine (the ones used in our experiments), power consumption when one core is active is 29.1 watts. It grows to only 30.4, 31.3, and 31.8 watts when 2, 3, and 4 cores are active, respectively. This implies that two such machines using one core each consume much more energy than a single machine executing the same work on two (or even four) cores if we shut down the second machine. This explains the potential for power saving by consolidation. Further, usually in DCs, for each 2 watts consumed by the machine, an extra watt is required for cooling, another reason to reduce energy consumption.

To calculate a migration penalty, we take a perhaps pessimistic approach and assume that while migrating a VM (freezing the VM, transporting the image, and restoring it) the VM fails to respond entirely, so its SLA fulfillment is 0. Finally, to determine required and given resources, we can get information from the monitors, or use the ML predictors to be explained in the next subsection. Also to determine the RT and SLA, in a reactive system we can try to obtain it statistically from the previous executions, while we are doing it proactively using our learned models.

B. Adaptive Models

When making decisions we often find that the required information 1) is not available, 2) is highly uncertain, 3) cannot be read because of privacy issues, or 4) obtaining it interferes too much with the system. Examples of this occur when reading from both PMs and VMs, and information coming from VMs is extremely delicate to handle and interpret. Observed resource usage can be altered by the observation window, the span of time between samples, or the stress of its hosting PM. Overheads of virtualization also add noise to the resource observation, independently of the load received by each VM. Opening the VM to read information from its internal system log could be against customer privacy agreements. Furthermore monitors can also add overhead to the PM, altering VM's performance; e.g. during experiments we occasionally observed monitors peaking up to 50% of an Atom CPU thread.

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 reactive to changes. In today's systems, this occurs continuously due to e.g. automatic software updates.

Here we build predictive models for all elements that could be considered relevant for deciding VM placement. From load characteristic of each web-service and its clients (Requests per Time Unit, average Bytes per Request, average Computing Time per Request in no-stress context), we learn and predict the resources that the VM will use to serve its requests (CPU, memory, I/O network traffic, and energy). As reported in previous work [8], the memory used by a PM memory can be safely assumed to be the sum of the memory allocated to its VM's, and PM network I/O is the sum of the I/O of its VM's. But total CPU used by a PM typically exceeds the sum of CPU power used by its VM's, due to management overhead; we thus learn the function describing the amount of PM CPU used as a function of the number of VM's and their metrics. We add to these predicted values information on the current load arriving to each VM and information from the gateway element (sizes of the queues of pending requests for these VMs, which represent additional immediate load). This information suffices to predict, by an another previously trained predictive model, response time and/or SLA fulfillment level. We then have all the elements (processed jobs, SLA fulfillment, and energy costs) to compute the profit generated by this particular PM.

Also in previous work [8] we determined that resource usage and response time, in this setting, can be modeled reasonably well by piecewise linear function, which explains that a regression trees (decision trees with linear regressions at the leaves) work well as predictive models. We used in particular the M5P regression tree method in the WEKA package. One exception is the prediction of SLA for each VM, where we used the k-Nearest Neighbor technique, which works by comparing the current situation with those seen before and choosing the most similar one(s). See [14] for more details on these algorithms.

Table I shows, for each predicted element, the ML method used for prediction, the correlation between the real and predicted values when validating the model, the mean absolute error and error standard deviation, the number of instances for training and validating the model, and the range (min,max) of the data. Our hypothesis is that predicting these data with such low error will help the decision maker to manage better the datacenter than not having it. A choice we had was whether to predict the RT and compute the SLA fulfillment value or try to predict the SLA fulfillment value directly. We noted here that better results are obtained if SLA is predicted directly, possibly because it has a bounded range so it is less sensitive to outliers.

C. Scheduling Algorithms

Let us discuss the methods used for solving the mathematical program above. In general, the functions mentioned in the program need not be linear, but it is not difficult to find reasonable piecewise linear approximations. One could then use a Mixed Integer Linear Program (MILP) solver, but as seen in our previous work [6], even decent out-of-the-box solvers (e.g. GUROBI [13]) required several minutes to schedule 10 jobs among 40 candidate hosts. The problem aggravates when as we want to use more complex functions (e.g., k-NN for SLA) and become Now, by having more complex functions (SLA function becomes a K-NN method, visiting several times all $examples \times variables$ for each tentative

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

Table I
LEARNING DETAILS FOR EACH PREDICTED ELEMENT AND SELECTED METHOD. WE USED A 66%/34% TRAINING/TESTING DATASET SPLIT IN ALL CASES.

solution), exhaustive MILP methods become infeasible if we want to get schedules at least once per hour. This made us to look for heuristic algorithms that produce fast and good approximations; as in [6], we used the classic Ordered Best-Fit method [22] for bin packing. In our terminology, it tries to place each VM in order in the PM where it fits best — see Algorithm 1. In the algorithm, the profit function is the responsible of computing the SLA, energy, migration and latency factors, computing the profit for each tentative placement.

Algorithm 1 Descending Best-Fit algorithm

```
for each vm i:
    get_data(i);
    res_req[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:
        profit <- profit(v,h,res_req[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);
```

If starting from scratch, the running time is proportional to the product of number of VM's times number of PM's. We considered a number of points to reduce it. One is that we do not include in the scheduling process VMs and PMs that are already performing well in a consolidated way, which is probably the large majority if the result of the previous scheduling round was good. Also, the method only considers for scheduling across DC's those virtual machines that could improve its QoS if moved across DCs (namely, because all PM's in their current DC already have a very high load). Thus, each DC only provides to the global scheduler a set of available physical machines and a set of VM's that may benefit if scheduled somewhere else. We thus have a two-layer approach: a number of intra-DC scheduling problems, solved starting from a possibly quite good previous schedule, and one global inter-DC problem, with a narrow interface to the intra-DC problems. In our experiments, this approach largely reduces solving cost, compared to the theoretical worst-case. Additional optimizations include considering only once identical empty host machines and not considering almost full hosts that cannot accommodate additional VM's.

V. EXPERIMENTS

A. Environment Description

We have performed the experiments on low-energy consumption machines (Intel Atom 4 Core), where resource management is critical in order to accept as much load as possible without degrading QoS. PMs run the Oracle VirtualBox virtualization platform, and each VM runs a web-service software stack (Apache, PHP, MySQL). The workload used corresponds to the Li-BCN Workload [7], a workbench and collection of traces from different real hosted web-sites offering from file hosting to image-gallery services; the workload was properly scaled to create heavy load for each experiment. Here we use client-service transactional benchmarks, but other kind of web-services based on message-passing could also be of interest.

Our scenario, as a case of use, is composed of four DCs in different continents (e.g. Brisbane, Australia; Bangaluru, India; Barcelona, Spain; Boston, Massachusetts), connected by high-speed network (network energy costs are not considered in this work; we keep this as future work). For each DC there is an amount of clients accessing the the services according to their local workload. Note that we performed the experiments in our local DC, but introducing network latencies and delays between machines and clients corresponding to the four different simulated geographical locations. This should suffice as a proof of concept for the model, learning components, and VM behaviors.

To price each element involved in the system, we established that providers behave as a cloud provider similar to Amazon EC2, where customers rent VMs in order to run their web-services (0.17 euro per VMh). For energy costs, we obtained the energy cost (euros per kWh) for the different places where we have a DC placed, so the cost of running a PM will depend on the DC where it is placed. Also, the migration costs depend on the latency and bandwidth between DC connections. We took as example the intercontinental network of the Verizon network company [3] to obtain latencies between locations and assumed a fixed bandwidth of 10 Gbps.

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

B. Intra-DC Comparatives

The first set of experiments are to check the benefits of driving an intra-DC scheduling for VMs using the learned models. As seen in previous works [8], Best-Fit performs better among greedy classical ad-hoc and heuristics, and here we check it against the environment. We compare 1) the Best-Fit algorithm checking if a VM can fit in the PM given the resources it has used in the last 10 minutes, and optimizing just power and latency to clients; 2) the Best-Fit algorithm with resource overbooking (BF-OB), i.e. booking for a VM double the resources it requires, to account for unexpected load peaks; and 3) the ML-enhanced Best-Fit, which uses the predicted CPU, memory, and I/O required for each VM to decides if it fits in a PM. The goal is to see how much the learned models help Best-Fit to (de-)consolidate in a way that maintains high throughput and SLA without wasting energy.

We set up 4 PMs with OpenNebula and VirtualBox, holding a total of 5 VMs, a PM acting as gateway and DC manager, and 4 machines generating LiBCN10 scaled load towards the 5 VMs. Figure 4 shows the results of running the workload for 24 hours, with a scheduling round every 10 minutes.

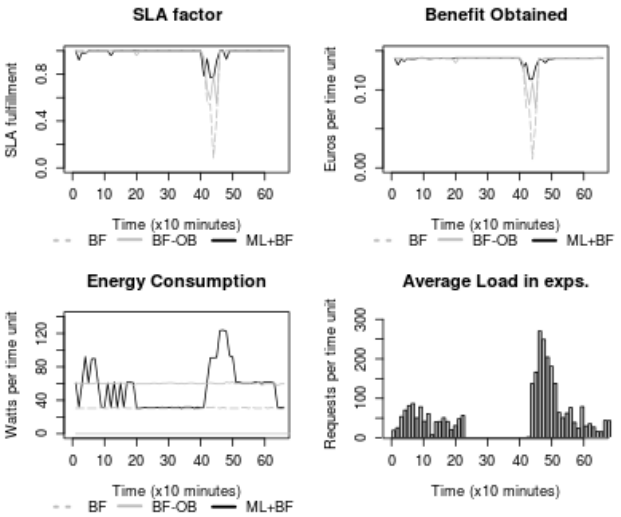

Figure 4. Results and Factors for Intra-DC Scheduling

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

Not reported here, for space reasons, is the fact that these ML-augmented versions can automatically adapt to changes in task execution prices, SLA penalties, and power price as shown on [6]. 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.

C. Inter-DC Comparatives

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

As a case of study, here we set one PM to represent a DC. As the intra-DC scheduler will arrange local PMs to a correct SLA fulfillment level, this PM will represent an on-line machine available to host a VM just entering the system or arriving from another DC. Each DC has an access point for clients (an ISP) machine collecting all the requests originating in the area where the DC is and sent to any VM in our system. Requests arriving to a DC but aimed to a VM on a another DC will be routed through our network, experiencing the latency between the the local DC and the remote DC. We apply our workload upon each VM from each ISP, but scaling each of the four workloads differently and simulating the effect of different time zones and load time patterns. Table II shows the prices and latencies used.

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

Table II
PRICES AND LATENCIES USED IN THE EXPERIMENTS. LATENCIES ARE IN MS [10GBPS LINE])

Follow the Load and Consolidation

First of all we perform a "sanity check", looking at the movements of VM without adding SLA or Energy factors yet (the simple "follow the load" policy). That is, the driving function is SLA taking into account only the request latency. Given this, Best-Fit places each VM as close to its major load source as possible. Figure 5 shows the movement of a single VM being driven only by this kind of SLA, without any resource competition or energy awareness. The VM follows the main source load to reduce the average latency to its globally distributed clients.

After checking that "follow the load" occurs, we introduced the energy consumption factor. When the function to be optimized includes energy costs, the scheduler will consolidate more noticeably while also taking into account

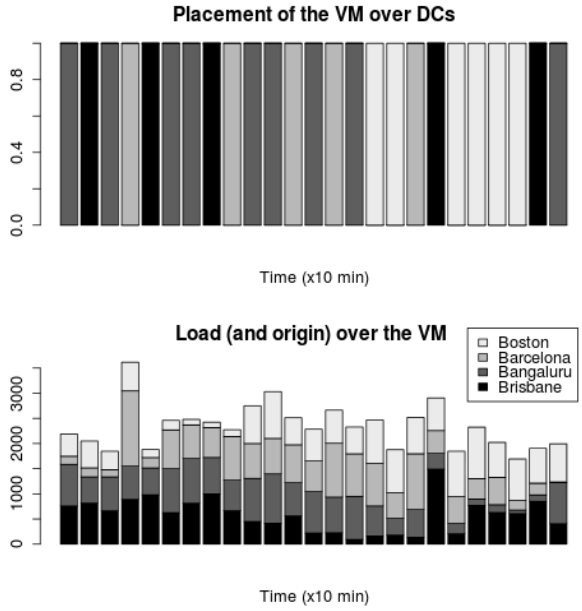

Figure 5. VM placement following the Load for Inter-DC Scheduling

client proximity and migration costs. In short, it will tend to consolidate either in the place closest to the load, or in the place where energy is cheapest, depending on the parameter values. Results are not reported.

Benefit of De-locating Load

Next, again for sanity check, we consider a somewhat artificial scenario with a single DC (in an averaged location for energy costs and latencies), where all VMs are held fixed receiving all the load, and compare it to another scenario where this DC can de-locate VMs (migrate VMs to other DCs temporarily) when it is overloaded. Despite having worse latencies and migration overheads when de-locating, SLA fulfillment increases from an average SLA of 0.8115/1 to an SLA of 0.8871/1 per VM doing this. This would translate, in the current experiment, to an average net benefit increase of 0.348 euro/VM in a day.

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

Full Inter-DC Scheduling

Once checking latency and energy factors, and observing the de-location benefit from a DC point of view, we perform the complete scheduling of the multi-DC system. Results are given in Figure 6. We note the following facts:

1) When load is heavy, the scheduler distributes VMs across DCs, deconsolidating across DCs as the intra-DC scheduler does within each DC. With the range of parameters and prices tested, SLA fulfillment and the associated revenue is still the most important factor driving deconsolidation. This can be seen in particular in highest load moments, or when SLA is below 1.
2) When SLA is not compromised, energy consumption pushes for consolidation into the DC with cheapest energy (see the low load moments).
3) When a potential VM move does not bring any improvement in SLA or energy use, the VM either stays in its DC or is consolidated to the nearest DC in latency.

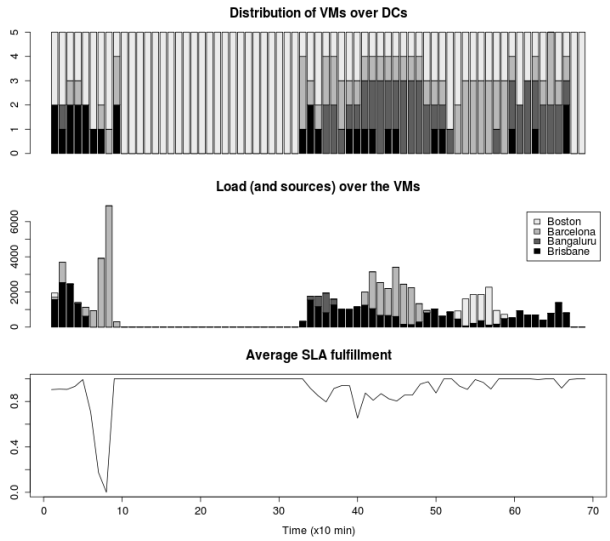

Figure 6. Results and Factors for Inter-DC Scheduling

Note that the workload generator produced a flash-crowd effect in the workload in minutes 70-90, for about 15 minutes, which clearly exceeds the capacity of the system. We kept this part of the workload in the test for realism.

Again, deconsolidation effects are seen when load (number of requests) increases or requests become more expensive to answer. In these cases, the system improves SLA by deconsolidation, countering the migration penalization and also enforcing the reduction of service-client latencies.

Benefit of Inter-DC Scheduling

We finally address experimentally the main question of the paper: is inter-DC optimization is better than intra-DC optimization? that is, does the ability to move VMs among DCs provide better solutions than keeping each VM within its DC only? Here we compare two scenarios: 1) The static global multi-DC network, where the VMs for each DC stay fixed without moving across DCs, where clients around the world can access every version but each web-service stays always in the same DC near its potential clients or customer selected DC; in other words, DCs do not cooperate by exchanging VM's, but just by redirecting the load they receive to its intended VM, local or located somewhere else. And 2) the dynamic multi-DC scenario we propose, where VMs may migrate among DCs to

improve global benefit. The benefit of the dynamic approach is basically the capability of moving the VMs towards the place where the energy is cheaper and/or available, or else to a DC with lower load for increased QoS.

At this stage, we chose for realism to use actual electricity prices for the four locations we have considered, which are relatively similar. As energy costs rise and markets become more heterogeneous and competitive, one should anticipate larger variations of energy prices across the world, and the benefit of inter-DC optimization priming energy consumption should be more obvious. This is particularly so as renewable sources such as solar energy become more widespread, because of their hour-to-hour variability and its very low cost once the production infrastructure is in place.

Figure 7 shows the comparison among the static context and the dynamic, when wanting to consolidate VMs among DCs.

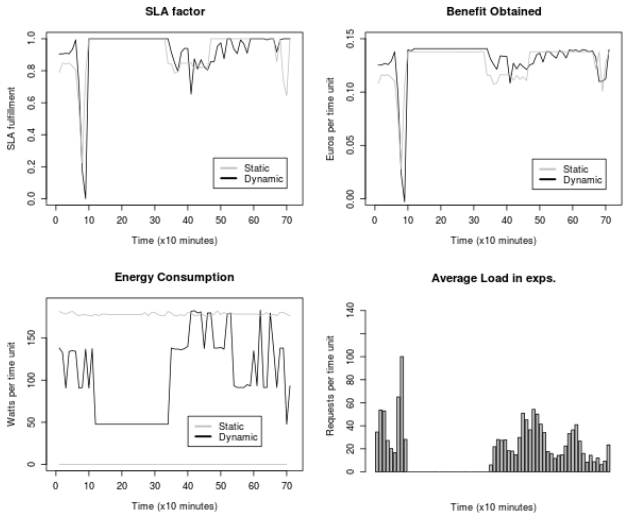

Figure 7. Comparative Static vs Dynamic Inter-DC for 5 VMs

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

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

Table III
COMPARATIVE OF RESULTS FOR THE MULTI-DC PER 5 VMS

Previous studies [10] showed that consolidation can achieve a power consumption reduction of more than 30% without counting the energy saving on cooling overheads (which may cause around a 1.5 increase in power consumption). So while maintaining SLA stable, we are able to improve energy consumption in a 42% by consolidate/deconsolidate in an inter-DC way, and further improve benefit by a 2% a day, for VMs that can not be consolidated in their local DCs.

Trade-Offs for QoS and Energy Costs

Finally, trade-offs between QoS and energy costs depend in the amount of load the VMs are receiving. Figure 8 shows the relation of the three variables from the observations of the given scenario; "load" is represented by amount of requests per time unit, as the most significant attribute of the load. Given the amount of load, as we want to improve the SLA fulfillment we are forced to consume more energy. For each level of load he can infer a characteristic function SLA vs Energy. This plot would allow a manager to visualize how much energy needs to be used to achieve a desired level of QoS or, conversely, what level of QoS can be achieved under some energy budget.

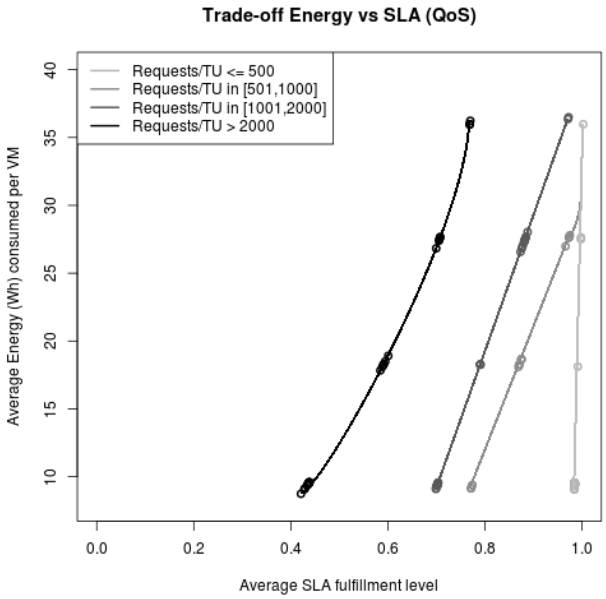

Figure 8. Relation of the SLA vs Energy vs Load

VI. CONCLUSIONS

Optimizing the schedule and management of multi-DC systems requires balancing several factors, like economic revenues, Quality of Service and operational costs such as energy. This problem can be modeled as a mathematical problem, solved approximately using e.g. greedy algorithms, and can also be enhanced using machine learning models to resolve uncertain or unavailable information, which lets the system make decisions adaptively without much explicit expert modeling.

Taking advantage of virtualization technology, we presented a model to solve a multi-DC scheduling problem which balances and optimizes the economic factors above. Experiments showed that the ML models can provide the required information to consolidate/deconsolidate across DCs according to the amount and geographic origin of the load for each VM, the latencies among clients and DC's and among DC's, and the different energy prices at different locations.

A few issues for future study are 1) how we decide which VMs are excluded from inter-DC scheduling or which PMs are offered as host candidates for scheduling; this affecting directly to scalability of the method; and provide information about how many PMs/VMs we can manage per scheduling round; 2) The inclusion of more operational costs like networking costs and bandwidth management 3) The green energy into the scheme not only to reduce energy costs but also environmental impact of computation. 4) The use of online learning methods, able to retrain continuously on recent data, to make the system react quickly to changes in either application behavior, hardware or middleware changes, or workload characteristics.

ACKNOWLEDGMENT

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 (SGR2009-1428).

ACRONYMS

DC	DataCenter
ISP	Internet Service Provider
ML	Machine Learning
Multi-DC	Multi-DataCenter
PM	Physical Machine
QoS	Quality of Service
RT	Response Time
SLA	Service Level Agreement
VM	Virtual Machine

REFERENCES

[1] Amazon DirectConnect (Jan.2013). http://www.amazon.com/DirectConnect/.
[2] Amazon WebServices (Jan.2013). http://aws.amazon.com/.
[3] Verizon (Jan.2013). http://www.verizonenterprise.com/about/network/latency.
[4] A. Andrzejak, S. Graupner, and S. Plantikow. Predicting resource demand in dynamic utility computing environments. In *Intl. Conf. on Autonomic and Autonomous Systems (ICAS)*, page 6, july 2006.
[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.
[6] J. Berral, R. Gavaldà, and J. Torres. Adaptive Scheduling on Power-Aware Managed Data-Centers using Machine Learning. In *12th IEEE International Conference on Grid Computing (GRID 2011)*, 2011.
[7] J. Berral, R. Gavaldà, and J. Torres. Li-BCN Workload 2010, 2011. http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=1099.
[8] J. Berral, R. Gavaldà, and J. Torres. Empowering Automatic Data-Center Management with Machine Learning. In *28th ACM Symposium on Applied Computing (SAC)*, 2013.
[9] 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.
[10] Í. Goiri, F. Julià, R. Nou, J. Berral, J. Guitart, and J. Torres. Energy-aware Scheduling in Virtualized Datacenters. In *Proceedings of the 12th IEEE International Conference on Cluster Computing (Cluster 2010), Heraklion, Crete, Greece, September 20-24*, 2010.
[11] I. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenhadoop: leveraging green energy in data-processing frameworks. In *7th ACM European Conf. on Computer Systems (EuroSys)*, 2012.
[12] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase. Virtual machine hosting for networked clusters: Building the foundations for 'autonomic' orchestration. In *Conf. on Virtualization Technology in Distributed Computing (VTDC)*, 2006.
[13] GUROBI. Gurobi optimization, 2013. http://www.gurobi.com/.
[14] 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.
[15] I. Kamitsos, L. Andrew, H. Kim, and M. Chiang. Optimal Sleep Patterns for Serving Delay-Tolerant Jobs. In *1st Intl. Conf. on Energy-Efficient Computing and Networking (eEnergy)*, 2010.
[16] J. Koomey. Estimating total power consumption by servers in the US and the world. *Final report. February*, 15, 2007.
[17] M. Lin, Z. Liu, A. Wierman, and L. L. Andrew. Online algorithms for geographical load balancing. In *Intl. Green Computing Conference (IGCC)*, 2012.
[18] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo. Dynamic resource management using virtual machine migrations. *IEEE Communications Magazine*, 50(9):34–40, 2012.
[19] 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.
[20] Y. Tan, W. Liu, and Q. Qiu. Adaptive power management using reinforcement learning. In *Intl. Conf. on Computer-Aided Design (ICCAD)*, 2009.
[21] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *3rd Intl. Conf. on Autonomic Computing (ICAC)*, 2006.
[22] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.
[23] D. Vengerov and N. Iakovlev. A reinforcement learning framework for dynamic resource allocation: First results. In *2nd Intl. Conf. on Autonomic Computing (ICAC)*, june 2005.
[24] P. Vienne and J.-L. Sourrouille. A middleware for autonomic QoS management based on learning. In *5th Intl. Workshop on Software Engineering and Middleware (SEM)*, 2005.
[25] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *4th USENIX Conf. on Networked systems design & implementation (NSDI)*, 2007.