2010 IEEE 3rd International Conference on Cloud Computing

Autonomic Management of Cloud Service Centers with Availability Guarantees

Bernardetta Addis, Danilo Ardagna and Barbara Panicucci
Politecnico di Milano
Dipartimento di Elettronica Informazione
Email: {addis,ardagna,panicucci}@elet.polimi.it

Li Zhang
IBM T.J. Watson Research Center
Email: zhangli@us.ibm.com

***Abstract*—Modern cloud infrastructures live in an open world, characterized by continuous changes in the environment and in the requirements they have to meet. Continuous changes occur autonomously and unpredictably, and they are out of control of the cloud provider. Therefore, advanced solutions have to be developed able to dynamically adapt the cloud infrastructure, while providing continuous service and performance guarantees.**

A number of autonomic computing solutions have been developed such that resources are dynamically allocated among running applications on the basis of short-term demand estimates. However, only performance and energy trade-off have been considered so far with a lower emphasis on the infrastructure dependability/availability which has been demonstrated to be the weakest link in the chain for early cloud providers.

The aim of this paper is to fill this literature gap devising resource allocation policies for cloud virtualized environments able to identify performance and energy trade-offs, providing a priori availability guarantees for cloud end-users.

Keywords: Infrastructure Cloud, Maintenance and Management of Cloud Computing.

I. INTRODUCTION

Cloud computing is an emerging paradigm that aims at streamlining the on-demand provisioning of software, hardware, and data as services, providing end-users with flexible and scalable services accessible through the Internet [8].

Modern cloud infrastructures live in an open world, characterized by continuous changes in the environment and in the requirements they have to meet. Continuous changes occur autonomously and unpredictably, and they are out of control of the cloud provider. Therefore, in order to provide infrastructure or software as a service, advanced solutions have to be developed able to dynamically adapt the cloud infrastructure, while providing continuous service and performance guarantees.

Recent studies [8], [4] have shown that the main challenges that cloud providers have to face are: (i) costs reduction, (ii) performance levels improvement, and (iii) availability and dependability enhancement.

Nowadays, energy management is becoming a priority in the design and operation of complex service-based systems, as the impact of energy consumption associated with physical infrastructures is steadily increasing. IT analysts predict that by 2012 up to 40% of an enterprise technology budget will be consumed by energy costs [9].

Furthermore, service providers need to comply to Service Level Agreement (SLA) contracts which determine the revenues and penalties on the basis of the achieved performance level. Quality of Service (QoS) guarantees have to be provided despite workload fluctuations which could be of several orders of magnitude within the same business day [7]. Finally, if end-user data and applications will be moved to the cloud, cloud infrastructures have to be as reliable as phone systems [8]: this has been recently the weakest link in the chain with several hours of system outages experienced by early cloud providers [4]. Virtualization is an enabling technology for cloud infrastructures which allows to share the same physical machine by multiple end-users applications with guaranteed performance. From a technological perspective, the consolidation of multiple users workloads on the same physical machine allows to reduce costs but it translates into higher utilization of physical resources. However, unforeseen load fluctuations or hardware failures have an impact among multiple applications.

Therefore, providing a cost efficient, dependable cloud infrastructure with guaranteed QoS is of paramount importance both for Industry and Academia. A number of autonomic computing solutions have been developed such that resources are dynamically allocated among running applications on the basis of short-term demand estimates. However, only performance and energy trade-off have been considered so far, with a lower emphasis on the infrastructure dependability/availability [1], [15].

Note in passing, that, the autonomic management implemented today by virtualization products (e.g., VMWare DRS or Virtuoso VSched) aims at equalizing the use of resources [19] but cannot provide any QoS guarantee. Furthermore, the high availability solutions can only react to failures or implement fault tolerance by active-passive configurations [23], [5], hence wasting some system resources.

The aim of this paper is to fill this literature gap devising resource allocation policies for cloud virtualized environments able to identify performance and energy trade-offs, providing a priori availability guarantees for cloud end-users.

Unlike most of the existing literature, this research will consider jointly a broader set of control variables. The problem will be modelled as a mixed integer non-linear programming problem and heuristic solutions based on non-linear programming techniques and local-search will be developed.

The remainder of the paper is organized as follows. Other approaches proposed in the literature are discussed in Section II. Section III introduces the reference system under study and the related performance model. The optimization problem formulation and the local-search approach are presented in Section IV and V. The experimental results in Section VI demonstrate the quality and efficiency of our solutions. Conclusions are finally drawn in Section VII.

II. Related work

Autonomic management and efficient energy use in modern IT systems are receiving great attention by the research community.

Previous literature has considered four main problems that must be addressed to enforce system management policies: 1) admission control, 2) capacity allocation, 3) load balancing, and 4) energy consumption optimization. Three different approaches have been proposed to design mechanisms to address each of these problems, namely, (i) the feedback loop based on control theory, (ii) machine learning adaptive techniques, and (iii) utility-based optimization techniques.

The feedback loop based on control theory [18], [15], [12] has the main advantage of guaranteeing system stability and, upon a change in workloads, it allows to accurately model transient behaviour. However, this technique is typically implemented by local controllers and, hence, only local optimization objectives can be reached. Authors in [15] have proposed a limited lookahead controller which can determine the set of servers in active state, the corresponding operating frequency, and the placement of virtual machines on physical servers. However, the virtual machine placement and capacity allocation problems are considered separately and the scalability of the approach has not been demonstrated. Recently Kalman filters have been adopted in [12] to track and control the CPU utilization in virtualized environments to guide capacity allocation, but considering a single multi-tier application executed in a single physical host.

Machine learning techniques base their learning sessions on live systems, without a need for an analytical model of the system [21], [13]. The work in [21] has proposed a capacity allocation technique that determines the assignment of physical servers to applications that maximizes the fulfilment of SLAs. Kephart et al. [13] have applied machine learning to coordinate multiple autonomic managers with different goals. In particular, their work has integrated an autonomous performance manager with a power manager in order to satisfy performance constraints while minimizing energy consumption costs by exploiting server frequency scaling. A recognized advantage of machine learning techniques is that they accurately capture system behaviour without any explicit performance or traffic model and with little built-in system-specific knowledge. However, training sessions tend to extend over several hour and, usually, the techniques proposed in the literature consider a limited set of managed applications and apply separate actuation mechanisms.

Utility-based approaches have been introduced to optimize the degree of user satisfaction by expressing their goals in terms of user-level performance metrics [11], [19], [20], [26]. Typically, the system is modelled by means of a performance model embedded within an optimization framework. Optimization can provide global optimal solutions or sub-optima by means of heuristics, depending on the complexity of the optimization model. Optimization is typically applied to each one of the four actuators separately. A first class of research studies deals with admission control for overload protection of servers [11]. Capacity allocation is typically viewed as a separate optimization activity that operates under the assumption that servers are protected from overload. A first set of studies focus on service differentiation in server farms by physically partitioning the farm into separate clusters, each serving one of the request classes (see, e.g., [22]). New technology trends advocate the sharing of physical resources among multiple request classes [3], [19], [20] supported also by virtualization and server consolidation technologies. In [19] a virtualization framework able to support heterogeneous workloads (batch and web interactive system) is presented, but energy saving policies are not considered. Authors in [26] have presented a multi-layer and multi-time scale solution for the management of virtualized systems, but they do not consider the trade-off between performance and system costs. Finally, in [14], a framework able to coordinate independent resource managers running on distributed physical nodes of a virtualized data center has been presented.

All of the above mentioned approaches considered only system performance (in terms of throughput or end-to-end response time) and, sometimes, energy trade-offs without providing however any availability guarantee for the running applications.

In this paper we extend our previous work [3] and we use the utility-based optimization approach to design capacity allocation, load balancing and energy saving policies, jointly, while fulfilling a priori availability constraints.

III. Reference System Infrastructure

Figure 1 shows the architecture of the cloud service center under study. The system includes a set I of heterogeneous servers which run a Virtual Machine Monitor (VMM) configured under work-conserving mode. Server physical resources are partitioned among multiple virtual machines each running a dedicated application. All the service requests are divided into classes, depending on the VMs required to support their executions, their SLA contracts and workload profiles. Overall the system serves K request classes, each one involving a set N_k of applications. As in [20], among the many server resources, we considered CPU and memory as representative for the resource allocation problem. For the sake of simplicity, we consider servers with a single CPU which supports Dynamic Voltage and Frequency Scaling (DVFS) by varying both its supply voltage and operating frequency from a limited set of values. However, our model can be easily extended to include modern multi-core systems. Let $\overline{c}_i$ be the time unit cost for server i when it is in low power sleep state. Each server i has a set H_i of operating frequencies at which it can work. When server i is working at frequency h has associated

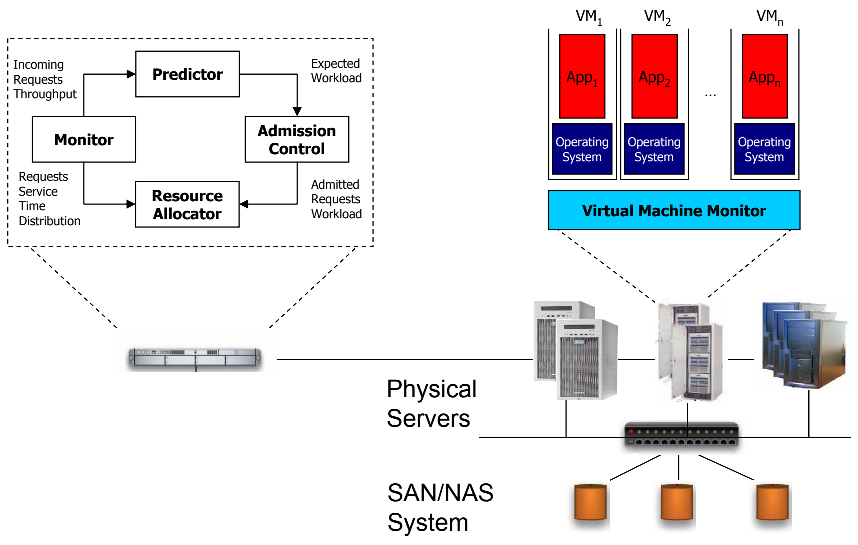

Fig. 1. Reference infrastructure

a capacity $C_{i,h}$ and an extra cost $c_{i,h}$. We assume as a first approximation, as in [15], that the overall power consumption of a server depends on its operating frequency/voltage. As in [20] we also take into account the costs cs_i representing the overhead when i is switched from the low power sleep to the active state, and cm representing the overhead when a VM is moved to a different server. Besides, to each request class k, is associated a linear utility function which specifies the per request revenue (or penalty) V_k, given the average response time R_k, as shown in Figure 2. $\overline{R}_k$ identifies the revenue/penalty region threshold and $-m_k$ is the utility function slope. Finally, SLA contracts specify, for each class, the minimum level of availability that has to be provided for each class k request, $\overline{A}_k$. Each physical server i has availability a_i, and in order to provide availability guarantees, the VMs running the same logical applications are replicated on multiple physical servers and work under the load-sharing configuration [5]. In our system, resource management is performed periodically by a centralized network controller which includes a monitor, a predictor, an admission controller, and a resource allocator [6]. The system monitor measures the workload and performance metrics of each application, identifies multiple requests classes, and estimates requests service demands [16]. The predictor forecasts future system load conditions based on historical data [3]. The admission controller decides whether to accept or reject new requests according to the current system workload [11]. Finally, the allocator determines the best system configuration and VMs to servers assignment maximizing SLA net profits, while providing availability guarantees. This paper focuses on the design of the resource allocator component. We will assume that the incoming workload is obtained as the result of prediction and admission control and service demands are obtained by the monitoring infrastructure (see, e.g. [16]). Resource allocation decision are applied in the systems periodically, e.g., every half an hour [20].

In order to estimate requests class performance, the service center is modelled by a queueing network which includes a set of multi-class single-server queues and a delay center (see [3] for further details). Each layer of queues represents the collection N_k of applications supporting requests execution at each tier. The delay center represents the client-based delays, or think time, between the server completion of one request and the arrival of the subsequent request within a session. User sessions begin with a class k request arriving to the service center from an exogenous source with rate λ_k. Upon completion the request either returns to the system as a class k' request with probability $\pi_{k,k'}$ or it completes with probability $1-\sum\limits_{l\in K}\pi_{k,l}$. The aggregate rate of arrivals for class k requests, denoted Λ_k, is given by the exogenous arrival rates and the transition probabilities: $\Lambda_k = \lambda_k + \sum\limits_{k'\in K}\Lambda_{k'}\pi_{k',k}$. Let $\mu_{k,j}$ denote the maximum service rate of a capacity 1 server dedicated to the execution of the application at tier j of request class k. We assume that $\mu_{k,j}$ parameters are estimated at run time by the monitoring component (see e.g., [16]). Moreover, requests at different application tiers within each class and on every server are executed according to the processor sharing scheduling discipline (which is common among Web servers and application containers) and service time of class k requests at server i follows a general distribution with mean $(C_{i,h}\mu_{k,j})^{-1}$.

Once a capacity assignment has been performed, VMMs guarantee to a given VM and the hosted application to receive as much resource (in term of CPU capacity and RAM) as it has been assigned to, regardless of the load imposed by other applications. For example, if two VMs have been assigned the same fraction of capacity, then they are guaranteed to receive 50% of the CPU each. Since the VMM has been

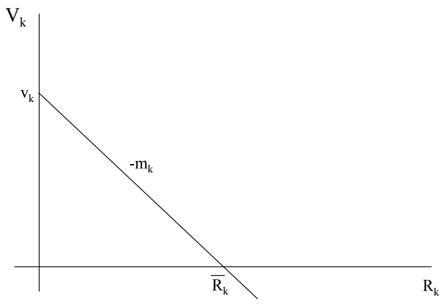

Fig. 2. Utility function

configured to support work-conserving mode [10], if one of the two aforementioned VMs is blocked for an I/O operation (or simply because no requests are running in the system), then the other one can consume the entire CPU.

The service discipline implemented by a VMM configured in work-conserving mode can be modelled by the Generalized Processor Sharing (GPS) scheduling [25]. Under GPS, the server capacity devoted to class k request for application tier j at time t (if any) when the server is running at frequency h is $C_{i,h}\phi_{i,k,j}/\sum_{k'\in\mathcal{K}(t),j\in N_{k'}}\phi_{i,k',j}$, where $\mathcal{K}(t)$ is the set of classes with waiting requests on server i at time t, and $\phi_{i,k,j}$ denotes the fraction of capacity devoted for executing VM at tier j for request class k at server i.

To evaluate the requests response time, we adopt the approximation discussed in [25] and widely used in the literature [1], [24]. Let us denote with $\lambda_{i,k,j}$ the rate of execution for class k requests at tier j on server i. Under this approximation, the average response time $R_{i,k,j}$ for the execution of the application at tier j of class k request at server i working at frequency h can be evaluated as:

$$R_{i,k,j}=\frac{1}{C_{i,h}\mu_{k,j}\phi_{i,k,j}-\lambda_{i,k,j}}. \tag{1}$$

The average response time for class k requests is the sum of the average response times at each tier computed over all servers:

$$R_k=\frac{1}{\Lambda_k}\left(\sum_{i\in I,j\in N_k}\lambda_{i,k,j}R_{i,k,j}\right). \tag{2}$$

System performance are evaluated by adopting analytical models as in [17], [1], [22], [24]. There is a trade-off between the accuracy of the model and the time required to estimate system performance which have to be evaluated with strict time constraints. Accurate performance models have been provided in the literature for Web systems, e.g. [11]. However, the high complexity for analysing even small size models has prevented us from using them here.

IV. Optimization Model Formulation

In this section, we formulate the resource allocation problem with availability constraints. The focus is on determining resource allocation policies that allow to maximize the total profit, that is the difference between the revenues from SLA contracts and the total costs. However, when maximizing the profit, we should guarantee availability of the services provided.

To evaluate the optimal resource allocation policies, we need to determine the servers to be switched to active or low power sleep state with their relative operating frequencies, the set of applications executed by each server, the request volumes at different servers, and the capacity devoted to the applications at each server.

To this end we introduce the following decision variables:

x_i:= 1 if server i is running, 0 otherwise

$y_{i,h}$:= 1 if server i is working at frequency h, 0 otherwise

$z_{i,k,j}$:= 1 if the application tier j (i.e. its hosting VM) for class k request is assigned to server i, 0 otherwise

$w_{i,k}$:= 1 if at least an application tier j for class k request is assigned to server i, 0 otherwise

$\lambda_{i,k,j}$:= rate of execution for class k requests at tier j on server i

$\phi_{i,k,j}$:= fraction of capacity devoted for executing the VM hosting application tier j for request class k at server i.

Let us also denote with $\overline{x}_i$ and $\overline{z}_{i,k,j}$ the value of the variables x_i and $z_{i,k,j}$ in the solution of the previous control time period.

The optimization problem which involves the joint decision of servers switching, frequency scaling, application placement, load balancing, and capacity allocation is reported in Figure 3. The goal is to maximize the difference between revenues from SLAs and the costs associated with servers switching and moving VMs: in the objective function, the first and second terms indicate the revenues from SLA, the third term is the cost of servers in low power sleep state, the fourth is the cost of using servers according to their operating frequency, followed by a penalty term incurred for switching servers from the low power sleep to the active state, and a penalty term associated with moving VMs. Note that the term $\sum_{k\in K}u_k\Lambda_k-\sum_{i\in I}\overline{c}_i$ in the objective function, can be dropped since it is independent of the decision variables.

Constraint family (3) entails that the traffic assigned to individual servers, and for every application tier, equals the overall load predicted for class k jobs. Note that, for a given request class k, the overall load Λ_k is the same at every tier. Constraint family (4) expresses the bounds for the VMs capacity allocation (i.e., at most 100% of the server capacity can be used). Constraint family (5) guarantees that, if the server i is in active state, then exactly one frequency is selected in the set H_i, i.e., only one variable $y_{i,h}$ is set to 1. Hence, the extra cost of a server working at frequency h and its corresponding capacity can be expressed by $\sum_{h\in H_i}c_{i,h}y_{i,h}$ and $\sum_{h\in H_i}C_{i,h}y_{i,h}$, respectively. Constraint family (6) allows assigning application tiers to servers that are in active state. Constraint family (7) allows running requests at a server only if the corresponding application tier has been assigned to it. Constraints family (8)

(P1)

$$\max \sum_{k\in K}\left(-m_k \sum_{i\in I, j\in N_k} \frac{\lambda_{i,k,j}}{\left(\sum\limits_{h\in H_i} C_{i,h}y_{i,h}\right)\mu_{k,j}\phi_{i,k,j}-\lambda_{i,k,j}}\right)+\sum_{k\in K} u_k\Lambda_k - \sum_{i\in I}\overline{c}_i - \sum_{i\in I,h\in H_i} c_{i,h}y_{i,h} +$$

$$-\sum_{i\in I} cs_i \max(0, x_i - \overline{x}_i) - \sum_{i\in I,k\in K,j\in N_k} cm \max(0, z_{i,k,j} - \overline{z}_{i,k,j})$$

subject to

$$\sum_{i\in I}\lambda_{i,k,j} = \Lambda_k \qquad \forall k\in K, j\in N_k \tag{3}$$

$$\sum_{k\in K,j\in N_k}\phi_{i,k,j} \le 1 \qquad \forall i\in I \tag{4}$$

$$\sum_{h\in H_i} y_{i,h} = x_i \qquad \forall i\in I \tag{5}$$

$$z_{i,k,j} \le x_i \qquad \forall i\in I, k\in K, j\in N_k \tag{6}$$

$$\lambda_{i,k,j} \le \Lambda_k z_{i,k,j} \qquad \forall i\in I, k\in K, j\in N_k \tag{7}$$

$$\lambda_{i,k,j} \le \left(\sum_{h\in H_i} C_{i,h}y_{i,h}\right)\mu_{k,j}\phi_{i,k,j} - \epsilon \qquad \forall i\in I, k\in K, j\in N_k \tag{8}$$

$$\sum_{j\in N_k} RAM_{k,j} z_{i,k,j} \le \overline{RAM}_i \qquad \forall i\in I, k\in K \tag{9}$$

$$\prod_{j=1}^{N_k}\left(1-\prod_{i=1}^{M}(1-a_i)^{w_{i,k}}\right) \ge \overline{A}_k \qquad \forall k\in K \tag{10}$$

$$\sum_{j=1}^{N_k} z_{i,k,j} \ge N_k w_{i,k} \qquad \forall i\in I, k\in K \tag{11}$$

$$\lambda_{i,k,j}, \phi_{i,k,j} \ge 0 \qquad \forall i\in I, k\in K, j\in N_k$$

$$x_i, y_{i,h}, z_{i,k,j}, w_{i,k} \in \{0,1\} \qquad \forall i\in I, k\in K, j\in N_k$$

Fig. 3. Optimization Problem

guarantees that resources are not saturated. Actually, these constraints should be formulated as a strict inequality: the small positive parameter ϵ is introduced in order to have a well stated problem. Constraint family (9) guarantees that the memory available at a given server is sufficient to support all the applications assigned to it. Constraint family (10) forces the availability, obtained assigning servers to class k request, to be at least equal to the minimum level agreed according to the load-sharing fault tolerance scheme [5]. Finally, (11) relates variables $z_{i,k,j}$ and $w_{i,k}$. (P1) is a mixed integer non-linear programming problem. Even if the set of server in active state is fixed, with servers operating at pre-selected frequencies, and fixed assignments of applications running on each server (i.e., the value of variables x_i, $y_{i,h}$, and $z_{i,j,k}$ have been determined), the joint load balancing and capacity allocation problem is still difficult. This is because the objective function is neither concave nor convex. In fact, it is possible to prove by diagonalization techniques, that the eigenvalues of the Hessian of the cost function are mixed in signs (see [3]). Commercial optimization tools can solve only (P1) small size instances. For realistic problems of reasonable size, a heuristic approach has to be considered. We provide a solution based on a local-search approach.

V. Solution Algorithm

The method proposed is a generalization of the method described in [3], [2]. In particular, we modified the local-search strategy to take into account the availability constraints. The algorithm is composed of two parts: a method for finding an initial solution (Algorithm 1, step 1) and a local-search method for improving that solution (Algorithm 1, steps 5 and 9). The local-search procedure is constructed upon two main procedures. The first is devoted to the optimization of the discrete variables (tier assignment $z_{i,k,j}$, and server use established by x_i and $y_{i,h}$): given a current solution s, a neighbourhood $N(s)$ is defined as the set of solutions that can be constructed applying a "move" to s; at each step, the neighbourhood is explored and the best solution in $N(s)$ is chosen. The latter is devoted to optimize the continuous variables ($\lambda_{i,k,j}$ and $\phi_{i,k,j}$): the method (fixed point iteration)

searches for a local improvement changing, in a cyclic fashion, variables λ, while keeping ϕ fixed, and vice-versa, until a stationary point is found (see [3] for further details). The two tasks are used in sequence, in fact every time a new current solution is found exploring the neighbourhood, a fixed point iteration is performed, this allows to balance the load after the server and VMs configuration is changed.
Particular care must be taken in the generalization of the moves from the original algorithm. In fact the availability constraints make necessary to add several restriction to possible moves to avoid the loose of feasibility. We describe briefly the set of moves and the changes added to take into account the availability constraints:

- **Increase/decrease of frequency**. For each server we can decide to increase/decrease the working frequency according to the new application loads. This move has been kept unchanged, in fact starting from a feasible solution with given loads, a change in frequency compatible with the new loads does not affect the availability.
- **Server switch to low power sleep/active state**. For each server we can decide whether to switch it to low power sleep state and allocate all the applications of that server on the other available servers. As the server availabilities are in general different, then some of available servers can not be considered for re-allocation. When a new server is switched to active state, some applications in the bottleneck server are duplicated and allocated on that server, thus availability constraints are not violated.
- **Re-allocation of class-tier application**. Similar considerations are necessary in the re-allocation of an application on a new server, that must have enough availability to satisfy the application requests.
- **Servers exchange**. For each server we can decided to switch it to low power sleep state and to move all its applications to a different server to be switched to active state. This move is feasible only is the availability of the new server guarantees the availability requests of all the moved applications.

The modified moves that avoid violation of the availability constraints are a restriction of the original moves. This restriction can lead to a reduced capability of the local-search procedure to explore the solution space and to improve the initial solution. To overcome this difficulty a more complex strategy, that allows a wider exploration, has been developed. When the local-search that keeps feasibility fails to find an improvement, we use the original neighborhood that explores solution without taking into account the availability constraints, and the violation is recorded in a list (step 9 of Algorithm 1). This procedure is repeated for a given number of iterations, (Algorithm 1, step 8). After that we perform a step (Algorithm 1, step 11) in which additional servers are switched to active state for requests classes whose availability constraints are violated. We then repeat the local-search with availability constraints (Algorithm 1, step 5).

The strategy combines the two main features of classical

```
1  (x,y,z, $\phi$,$\lambda$)=Initial_Solution();
2  $STOP$=false;
3  while not STOP do
4      while improve do
5          Local_Search_Availability(x,y,z, $\phi$,$\lambda$);
6      end
7      ($\hat{x}$,$\hat{y}$,$\hat{z}$, $\hat{\phi}$,$\hat{\lambda}$)=Save_Best_Solution();
8      while iteration_limit do
9          Local_Search_withoutAvailability(x,y,z, $\phi$,$\lambda$);
10     end
11     Remove_Violations(x,y,z, $\phi$,$\lambda$);
12     if not_feasible_solution then
13         STOP=true;
14         (x,y,z, $\phi$,$\lambda$)=($\hat{x}$,$\hat{y}$,$\hat{z}$, $\hat{\phi}$,$\hat{\lambda}$);
15     end
16 end
17 return (x,y,z, $\phi$,$\lambda$);
```
Algorithm 1: Resource allocation optimization procedure

local-search strategy exploration and refinement. The local-search with availability constraints performs the refinement step of the solution, as it explores its neighbourhood. The original local-search (with no availability) allows larger changes in solution performing the exploration step.

VI. Experimental Results

We have evaluated the performance of our resource management algorithm for a variety of system and workload configurations.

All tests have been performed on a standard 2008 Intel workstation considering a large set of randomly generated instances. The number of servers $|I|$ has been varied between 80 and 400. The number of request classes $|K|$ has been varied between 10 and 100 and the maximum number of tiers $|N_k|$ has been varied between 2 and 4. In each test case, the number of tiers $|N_k|$ for different classes may be different. Service demands and m_k have been uniformly generated in the interval $[0.1, 1]$ sec and $[0.2, 2]$, respectively. With these values of m_k the revenues obtained by using a single CPU for one hour vary between \$1 and \$10 according to the current commercial fees (see for example the *Sun Utility Computing* infrastructure[1] or *Amazon Elastic Cloud*[2]). $\overline{R}_k$ is proportional to the number of tiers $|N_k|$ and to the overall demanding time at various tiers of class k requests (i.e., $\overline{R}_k = 10 \cdot \sum_{j \in N_k} \frac{1}{\mu_{k,j}}$). The server availability a_i has been uniformly generated in the range $[0.95, 0.999]$ and request classes availability constraints $\overline{A}_k$ have been set in the range $[0.999, 0.99999]$. RAM requirements for VMs have been varied between 256 MB and 4 GB, while the service center includes physical machines with RAM capacity between 4 and 64 GB. Servers with 6 P-states (a voltage and frequency setting) have been considered (the working operating frequencies were 2.8, 2.4, 2.2, 2.0, 1.8, and 1.0 GHz which implies 95, 80, 66, 55, 51, and 34W power consumption). The overhead of the cooling system has been

[1]http://www.sun.com/service/sungrid/index.jsp
[2]http://aws.amazon.com/ec2/

set equal to 0.7 (in modern service centres, for \$1 spent in servers energy, \$0.7 are spent for air conditioning [9]). The cost of energy per kWh has been varied between 0.05 and 0.25 \$/kWh.

The switching cost of servers cs_i has been evaluated as in [15] considering the reboot cost in energy consumed and ranges from $\$24 \cdot 10^{-5}$ to $\$48 \cdot 10^{-5}$. The cost associated with VMs movement cm has been evaluated by considering the cost of use of server to suspend, move and start up a VM according to the values reported in [19] and has been estimated in the range $\$24 \cdot 10^{-6}$ to $\$12 \cdot 10^{-4}$.

Half of the tests have been run by considering homogeneous systems (including only servers of capacity 1) and the remaining by considering heterogeneous systems (half servers with capacity 1 and half of capacity 2). In the former case, the load was evenly shared among different tiers, while, more realistically, in the second case application tiers were the system bottleneck.

Our algorithms have been coded in Java and the execution time has been optimized by using Sun Java 1.6 just in time compiler. Tests are run by using a single core of the CPU.

The optimization time of the analyses reported in this section is limited to half an hour. Table I reports the average optimization time obtained for problem instances of different sizes. Within half an hour time constraint, systems with up to 50 classes, 400 servers, and around 1,000 VMs can be solved.

Figure 4 reports the execution trace of a significant run of our algorithm for an heterogeneous system including 50 classes and 400 servers. The x axis reports the execution time, while the y axis reports the objective function value. Solution A depicts the best found feasible solution, while Solution B is the local-search current solution which can sometimes violate constraint family (10). After almost 40 sec, the local-search finds the first local optimum and Solution A is not further improved. Then, constraint (10) is relaxed and the search can continue to explore the solution space. In around 50 sec the search find a better optimum feasible solution and the plot of Solution A coincides with Solution B. Note that, during the search the value of Solution B can increase or decrease. Solution B usually decreases suddenly when the search goes back in the feasible solution set. Indeed, in that case additional servers are turned on to improve the system availability and additional energy costs are incurred accordingly. As a final consideration, the plot shows that the local-search is effective since the transition in the unfeasible region allows to find after 80 sec a local optimum feasible solution which is around 11% greater than the first local optimum at time 40 sec.

In order to evaluate the impact of availability constraints on the net revenues we have also considered a case study based on realistic workloads created from a trace of requests registered in the Web site of a large University in Italy. The real system includes almost 100 servers and the trace contains the number of sessions, on a per-hour basis, over a one year period. Realistic workloads are built by assuming request arrivals to follow non-homogeneous Poisson processes with rates changing every hour according to the trace. From this

$(\|K\|\ ,\ \|I\|)$	Homogeneous System	Heterogeneous System
(10,80)	0.72	1.12
(20,160)	7.31	7.25
(30,240)	32.61	25.90
(40,320)	102.53	72.23
(50,400)	314.98	220.90

TABLE I
OPTIMIZATION ALGORITHM AVERAGE EXECUTION TIME (SEC)

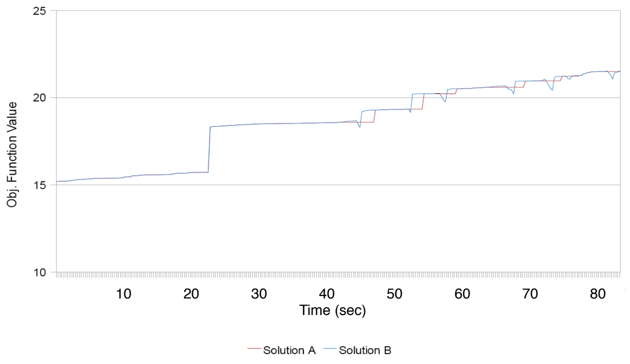

Fig. 4. Execution trace for the solution of a heterogeneous system with $|K| = 50$ and $|I| = 400$

logs, we have extracted 50 requests classes which correspond to the days with the highest workloads experienced during the year. The workload follows a bimodal distribution with two peaks around 11.00 and 16.00 (see Figure 5).

Results are illustrated in Figures 6 and 7 which report the number of on line servers and net revenues for a service center including 400 servers with and without availability constraints. As the plots show, under light load conditions between 1.00 and 9.00, the solution with availability constraints introduces almost 100% additional servers with respect to the solution without availability. Vice versa, under heavy load conditions the system naturally adopts a larger number of servers and, in that case, the solution with availability introduces only 30% additional servers. Furthermore, by inspecting the obtained solutions, results have shown that the local-search procedure is able to use effectively the DVFS capability of new servers and to manage service resources effectively since the net revenues of the solution with availability are only 20% lower on overage over the 24 hours.

VII. CONCLUSION

This paper proposes resource allocation policies for the management of multi-tier virtualized cloud systems with the aim at maximizing the profits associated with multiple-class SLAs. A heuristic solution based on local-search which provides also availability guarantees for the running applications has been developed. Cloud service centers including up to 400 servers and 50 request classes can be managed efficiently. Current work is considering the resource allocation problem at very fine-grained time scales (sec) by adopting control theory models. Furthermore, the adoption of full system power

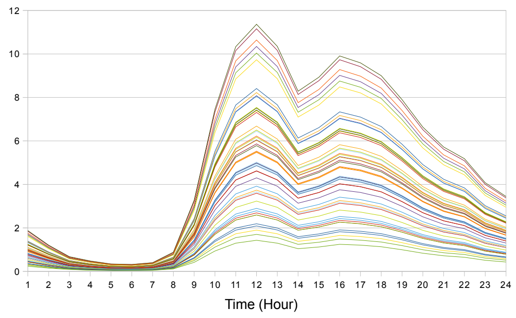

Fig. 5. Case study incoming workload (req/sec)

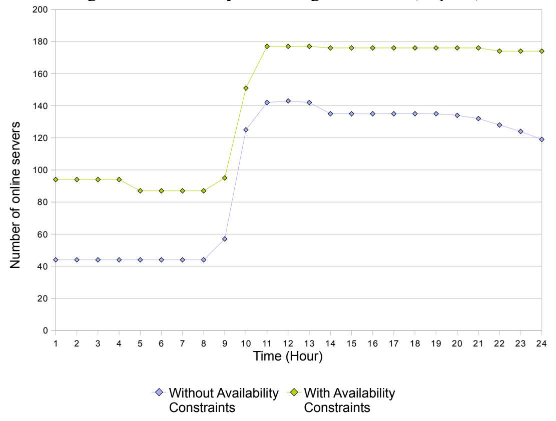

Fig. 6. Case study number of on line servers

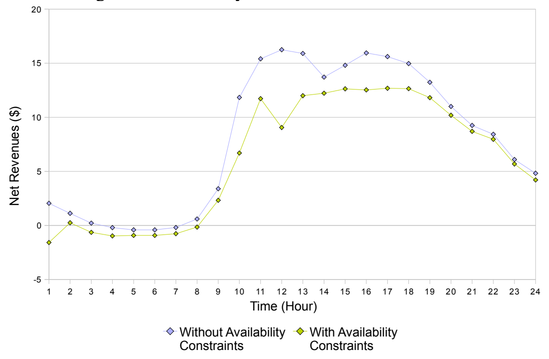

Fig. 7. Case study net revenues

models characterizing servers' energy consumption in terms of the servers' utilization will be considered. Finally, in order to improve the scalability of the approach class and server partitioning techniques will be developed.

Acknowledgement

The work of Danilo Ardagna and Barbara Panicucci has been partially supported by the GAME-IT research project funded by Politecnico di Milano and by the European Commission, Programme IDEAS- ERC, Project 227977-SMScom.

References

[1] B. Abrahao, V. Almeida, J. Almeida, A. Zhang, D. Beyer, and F. Safai. Self-Adaptive SLA-Driven Capacity Management for Internet Services. In *Proc. NOMS06*, 2006.

[2] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-Aware Autonomic Resource Allocation in Multi-tier Virtualized Environments. *IEEE Trans. on Services Computing,* to appear.

[3] D. Ardagna, M. Trubian, and L. Zhang. SLA based resource allocation policies in autonomic environments. *Journal of Parallel and Distributed Computing*, 67(3):259–270, 2007.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, I. S. A. Rabkin, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf.

[5] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11 – 33, Jan.-March 2004.

[6] A. Chandra, P. Goyal, and P. Shenoy. Quantifying the Benefits of Resource Multiplexing in On-Demand Data Centers. In *In Proc. of the 1st Workshop on Alg. and Arch. for Self-Managing Systems*, 2003.

[7] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centres. In *18th ACM SOSP*, Banff, Canada, October 2001.

[8] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali. Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing*, 13(5):10–13, 2009.

[9] W. Feng, X. Feng, and R. Ge. Green Supercomputing Comes of Age. *IT Professional*, 10(1), 2008.

[10] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing Performance Isolation Across Virtual Machines in Xen. In *Middleware*, 2006.

[11] V. Gupta and M. Harchol-Balter. Self-adaptive Admission Control Policies for Resource-sharing systems. In *SIGMETRICS*, 2009.

[12] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In *ICAC2009 Proc.*, 2009.

[13] J. Kephart, H. Chan, R. Das, D. Levine, G. Tesauro, F. Rawson, and C. Lefurgy. Coordinating Multiple Autonomic Managers to Achieve Specified Power-performance Tradeoffs. In *ICAC Proc.*, June 2007.

[14] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan. vManage: loosely coupled platform and virtualization management in data centers. In *ICAC2009 Proc.*, 2009.

[15] D. Kusic, J. O. Kephart, N. Kandasamy, and G. Jiang. Power and Performance Management of Virtualized Computing Environments Via Lookahead Control. In *ICAC 2008 Proc.*, 2008.

[16] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. Cpu demand for web serving: Measurement analysis and dynamic estimation. *Perform. Eval.*, 65(6-7):531–553, 2008.

[17] G. Pacifici, M. Spreitzer, A. N. Tantawi, and A. Youssef. Performance Management for Cluster-Based Web Services. *IEEE Journal on Selected Areas in Communications*, 23(12), December 2005.

[18] W. Qin and Q. Wang. Modeling and Control Design for Performance Management of Web Servers Via an LPV Approach. *IEEE Transactions on Control Systems Technology*, 13(1), Jan 2002.

[19] M. Steinder, I. Whalley, and D. Chess. Server virtualization in autonomic management of heterogeneous workloads. *SIGOPS Oper. Syst. Rev.*, 42(1), 2008.

[20] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *WWW2007*, 2007.

[21] G. Tesauro, N. R. Das, and M. Bennani. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In *ICAC Proc.*, June 2006.

[22] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi. Analytic modeling of multitier Internet applications. *ACM Transaction on Web*, 1(1), January 2007.

[23] VMware. VMware High Availability. http://www.vmware.com/products/high-availability/features.html.

[24] X. Wang, Z. Du, Y. Chen, and S. Li. Virtualization-based autonomic resource management for multi-tier Web applications in shared data center. *J. Syst. Softw.*, 81(9):1591–1608, 2008.

[25] Z. L. Zhang, D. Towsley, and J. Kurose. Statistical analysis of the generalized processor sharing scheduling discipline. In *IEEE Journal on Selected Areas in Comm.*, 2003.

[26] X. Zhu, D. Young, B. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D.Gmach, R. Gardner, T. Christian, and L. Cherkasova:. 1000 islands: An integrated approach to resource management for virtualized data centers. *Journal of Cluster Computing*, 12(1):45–57, 2009.