An Approximate Dynamic Programming Approach to Resource Management in Multi-Cloud Scenarios

Pietrabissa Antonio, Delli Priscoli Francesco, Di Giorgio Alessandro, Giuseppi Alessandro, Panfili Martina, Suraci Vincenzo

Department of Computer, Control and Management Engineering "Antonio Ruberti", University of Rome "La Sapienza", Rome, Italy.

A. Pietrabissa is the corresponding author: e-mail: pietrabissa@dis.uniroma1.it, phone: 003977274040, fax: 003977274033.

An Approximate Dynamic Programming Approach to Resource Management in Multi-Cloud Scenarios

The programmability and the virtualization of network resources is crucial to deploy scalable ICT services. The increasing demand of cloud services, mainly devoted to the storage and computing, requires a new functional element, the Cloud Management Broker (CMB), aimed at managing multiple cloud resources to meet the customers' requirements and, simultaneously, to optimize their usage. This paper proposes a multi-cloud resource allocation algorithm that manages the resource requests with the aim of maximizing the CMB revenue over time. The algorithm is based on Markov Decision Process modelling and relies on Reinforcement Learning techniques to find on-line an approximate solution.

Keywords: cloud networks; resource management; reinforcement learning; Markov decision process; approximate dynamic programming.

1 Introduction

The increasing use of cloud infrastructures to rapidly deploy services leads the cloud operators to adopt programmable infrastructure paradigms. Instead of carefully design a dedicated, static, hardware infrastructure, the operators prefer to acquire the hardware (storage, computation and network resources) and manage them using a standard software. The Openflow protocol (McKeown et al., 2008) is a pragmatic example of Software-Defined Networking (SDN), where the logic network managed by Openflow is totally decoupled by the physical infrastructures that host the Openflow controller and the Openflow switches. Openstack (Sefraoui, Aissaoui, & Eleuldj, 2012) is another example of programmable infrastructure. While Openflow virtualizes only the network elements, Openstack allows virtualizing also the storage and the computing services. The virtualization layer adopted by Openflow and Openstack is an abstraction layer that allows the engineers to create logical networks, storage and computing services on top of any hardware configuration, in a technology independent fashion.

Even though the resources abstraction operated by the software-defined paradigms allows the dynamic programmability of the available hardware resources, this paradigm still highly depends on a fine tuning of the underlying physical machines. They must be deployed, configured and maintained, often manually. A further step beyond the resources programmability is the adoption of pure virtualized infrastructures, where the physical machines host a number of virtualized machines, each of which deputed to provide a finite and dedicated set of functionalities. The advantage of resource virtualization is the rapid deployment, maintenance and scaling up of existing environments. For instance, in case of a software or hardware malfunction, a virtual machine can be automatically migrated or rebooted. Network Functions Virtualization (NFV) is an emerging paradigm that exploits the machine virtualization to setup a pure virtualized network management system, in which each functionality runs on top of a virtual machine. The idea behind the NFV is to virtualize any network function (such as DHCP, NAT, Firewall, FTP, Web Hosting, etc.) so that the network configuration can be personalized and dynamically adapted to a specific customer's need automatically or by means of software APIs. The idea of deploying resources (such as storage, networking and computing) on-demand eliminates maintenance costs related to their ownership and,

thus, can radically change the operators' policies and the way they can collaborate to best match their customers' needs.

This paper assumes to operate in a scenario where the cloud operators make use of programmable resources and virtualized functionalities to manage their infrastructures. In this scenario, different cloud providers offer to their customers a variety of resources, either infrastructural or software. These resources are assigned based on the users' demands.

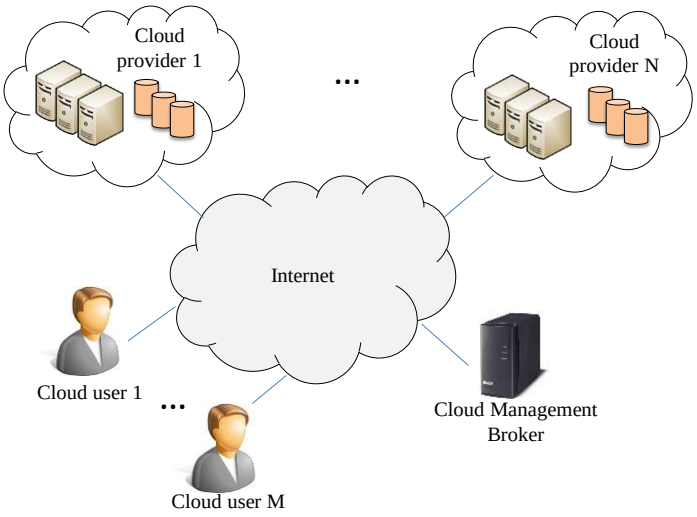

Figure 1. Representation of a multi-cloud system

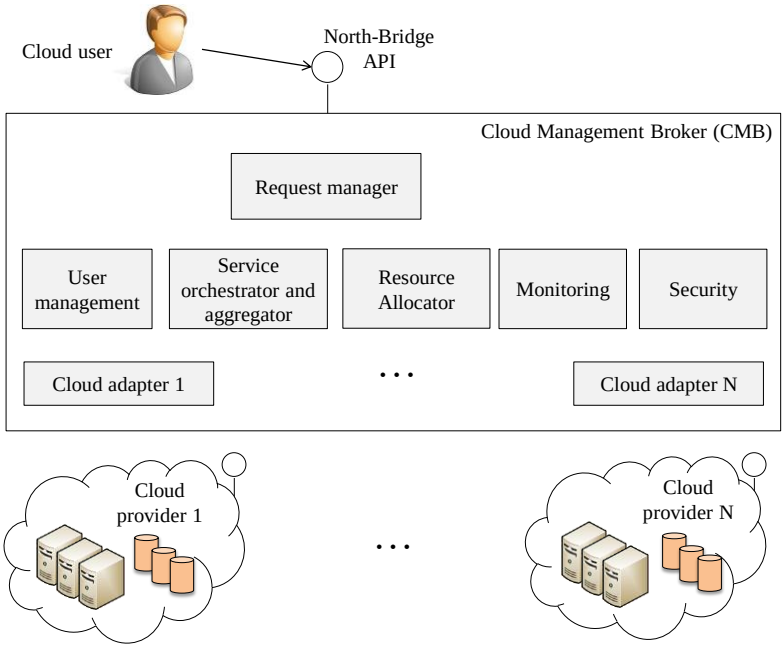

Figure 2. CMB architecture example.

A Cloud Management Broker (NIST, 2013) gives the users an interface to request the resources they need and allocates them on the different available cloud providers depending on various factors, such as availability, cost, and Quality of Service (QoS) requirements. Figures 1 and 2 show a multi-cloud scenario and a possible CMB architecture (see (Oddi, Panfili, Pietrabissa, Zuccaro, & Suraci, 2013) for details); in this scenario, the CMB role is crucial, since it is responsible both of the actual management of the multi-cloud resources and of satisfying the QoS needed by the users. It is then crucial to study algorithms to optimize and control the CMB resource usage.

The CMB concept is widely studied in the framework of several Future Internet related initiatives. This paper is based on the work performed by the authors in the framework of the Italian project PLATINO (PLATINO, 2015), of the EU FP7 project T-NOVA (T-NOVA, 2015), and of the European PPP FP7 Future Internet initiative (FI-Core, 2015; FIWARE, 2014). The paper is focused on the resource allocation aspect of the CMB, but the CMB role covers other key aspects, such as user information and preferences management, monitoring of the status of the cloud providers, real time match of security requirements and so on. Generally, a CMB is then not to be considered as a simple request translator between users and cloud providers, but as a more complex actor in the multi cloud system.

The CMB Resource Allocator role is to find and allocate resources to match every user request – e.g., in terms of storage, computing power, bandwidth – over a group of heterogeneous cloud providers. Two options exist to fulfill this task:

(1) allocate the needed resources on-demand, on a pay-per-use cost model;
(2) allocate the needed resources on a set of pre-purchased resources.

Both approaches have their pros and cons. With the former approach, the CMB can achieve high efficiency in the resource usage, whereas the latter approach may lead to higher profits, since pre-assigned resources are likely to be less expansive (e.g., the CMB could benefit from special quantity discounts).

The Resource Allocator task is then to decide how to fulfill the user requests, i.e., which resources have to be mapped (fully or in part) on the pre-purchased resources and which resources need some additional on-demand resources, with the aim of maximizing a given reward function over a time horizon. In the depicted scenario, the reward function is the net profit of the CMB.

1.1 State-of-the-art and paper contribution

For the time being, the multi-cloud problem has been mostly dealt with from an architectural point of view. Compatible One (CompatibleOne, 2015) is a solution which homogenizes language and interfaces for the description of the services offered by multiple cloud providers. This is an example of infrastructure that seems to be perfect to host advanced optimization and control algorithms such as the one proposed in this paper. The architectural frameworks in (Sirocco, 2015) and (Buyya, Ranjan, & Calheiros, 2010) allow a unified usage of resources belonging to different cloud providers; however, no novel optimization techniques are proposed, but only architectural visions. As far as

resource management techniques in cloud environments are concerned, (Wu, Kumar Garg, & Buyya, 2012) proposes an integrated scheduling and admission control algorithm which manages multiple IaaS providers, but it is based on heuristic rules. The papers (Rennie & Mccallum, 1999) and (Konstanteli, Cucinotta, Psychas, & Varvarigou, 2012) propose algorithms for resource allocation and admission control, respectively, but are applied to optimize a single cloud. The recent work in (Woo & Mirkovic, 2014) proposes an ad-hoc algorithm for the application allocation on multiple public clouds. The application workflow is described as a sequence of multiple transactions and tries to minimize the cost as well as to satisfy some Service Level Agreement (SLA) requirements. The solution algorithm in (Woo & Mirkovic, 2014) is very simple (it is based on exhaustive research) but it does not consider users demand distributions and it is only compared to the single cloud scenario.

Resource management problems in communications have been dealt with by numerous operations research (e.g., (Chaisiri, Lee, & Niyato, 2012; Macone, Oddi, Palo, & Suraci, 2013)) and control-theoretical methodologies (e.g., (Manfredi, 2014a, 2014b; Mascolo, 1999)). The proposed approach relies on a Markov Decision Process (MDP) modelling of the problem. MDPs catch both the research for an optimal solution of operations research methodologies and the system dynamics characterizing control-theoretical methods.

The scenario and the problem presented in this paper were introduced in (Oddi et al., 2013), where the CMB Resource Allocator problem is defined as a MDP and solved by Dynamic Programming (DP) algorithm. The utilization of DP algorithms in real networks is however limited by the fact that the probability distribution of the users' demand must be known in advance, and by scalability issues due to the large state-space, whose dimension explodes in realistic scenarios. It is well-known that the optimal control policy can be found off-line by means of DP algorithms and on-line by means of RL algorithms. Differently from the former algorithms, the RL approach does not rely on the knowledge of the transition probabilities and of the reward, which are learnt from direct experience without the need of a complete model of the environment (for this reason, RL methods are often referred to as *model-free* methods).

Reinforcement Learning (RL) approaches, as the one proposed here, are used in adaptive control-theory due to their ability to 'learn' the environment (i.e., the model) on-line (Lewis & Vrabie, 2009; Yang, Liu, & Wang, 2014), and are also applied to resource management problems in many fields (as, for instance, communications (Pietrabissa, 2011a), robotics (Tan, Balajee, & Lynn, 2014), electric vehicles (Di Giorgio, Liberati, & Pietrabissa, 2013) and so on). In (Pietrabissa et al., 2015), two RL algorithms were proposed to solve the CMB Resource Allocator problem. However, the algorithms proposed in (Pietrabissa et al., 2015) are still not scalable enough to be applied to real scenarios, and approximation techniques are then needed (e.g., (Pietrabissa, 2008a, 2008b, 2009)). In this respect, the contribution of this paper is that it adapts the state aggregation policy proposed in (Pietrabissa, 2008a) to the multi-cloud environment described in (Pietrabissa et al., 2015) and develops a RL algorithm on the aggregated model.

1.2 Paper organization

The paper is organized as follows: in Section 2, key concepts on MDP and RL are summarized; Section 3 models the CMB Resource Allocator problem as a MDP presents the proposed approximate RL algorithm; Section 4 shows the results of some numerical simulations; finally, Section 5 draws the conclusions and outlines some future researches.

2 Preliminaries

This Section introduces some key concepts relevant to MDP and RL.

2.1 Markov Decision Processes

A MDP is a discrete-time stochastic control process defined by the tuple $\{S, A, T, r, \Sigma, \gamma\}$, where S is a discrete, finite state set, $A(s)$ is the finite action set in state s, T is the state transition probability matrix, r is the reward function, i.e., $r(s, a, s')$ is the immediate reward obtained in s when action a is taken and state s' is the next state, $\Sigma \in [0; 1]^{|S|}$ is the initial state distribution over the state space S, and $\gamma \in (0,1)$ is the discount factor, that weights immediate rewards versus future rewards. Standard MDP definitions rely on the Markovian (or memory-less) property and on the stationary distribution of the stochastic process. Under these assumptions, T is stationary and its element $t(s, a, s')$ is the probability that the system trajectory transits from state s to s' when action a is taken.

A stationary policy u is a mapping of each state to an action, i.e., $u(s) = a, s \in S, a \in A(s)$[1]. The MDP problem is aimed at finding an optimal policy $u^*: S \to A$ that maximizes, in the long run, the expected discounted reward:

$$R(u) \coloneqq E_{u,\Sigma}\left\{\sum_{t=0,1,\dots,\infty} \gamma^t \cdot r\big(s(t), a(t), s(t+1)\big)\right\}, \tag{1}$$

where $s(t)$ and $a(t)$ denote the state and the action at time t, respectively, and $E_{u,\Sigma}\{\cdot\}$ denotes the expected value under policy u with initial state distribution Σ.

Finally, the value function $V_u(s)$ is the expected discounted reward starting from s and following policy u thereafter, and the action-value function $Q_u(s, a)$ is the expected discounted reward, starting from s, taking action a and following policy u thereafter:

$$V_u(s) = E_u\left\{\sum_{t=0,1,\dots,\infty} \gamma^t \cdot r\big(s(t), a(t), s(t+1)\big) \mid s(0) = s\right\}, \tag{2}$$

$$Q_u(s, a) = E_u\left\{\sum_{t=0,1,\dots,\infty} \gamma^t \cdot r\big(s(t), a(t), s(t+1)\big) \mid s(0) = s, a(0) = a\right\}, \tag{3}$$

where $E_u\{\cdot\}$ denotes the expected value under policy u.

2.2 Reinforcement Learning

As mentioned in Section 1, the paper interest is in solving the MDP by means of RL algorithms. Let the system be in a given state $s \in S$; RL algorithms take an action $a \in$

---

[1] For the sake of simplicity, we consider only integer policies, i.e., policies which select one action in each state; note that this choice is not restrictive, since it can be shown (Puterman, 1994) that an integer optimal policy always exists for unconstrained MDPs.

$A(s)$ based on a given control rule and then observe the next state $s' \in S$ and the obtained reward $r(s, a, s')$ after the transition. Based on the observations, the RL algorithms update an estimate of the value function of state s or of the action-value function of the couple (s, a).

Different RL algorithms differ by the rule used to decide the control action and by the rule used to update the value (or action-value) function. In this paper, for the sake of simplicity, the Q-learning algorithm is considered, but more complex RL algorithms can be straightforwardly used (e.g., SARSA(λ), as in (Pietrabissa et al., 2015), actor-critic methods (Sutton & Barto, 1998), …). The update rule of Q-Learning is the following:

$$Q\big(s(t), a(t)\big) \leftarrow Q\big(s(t), a(t)\big) + \\ +\alpha(t)\left[r\big(s(t), a(t), s(t+1)\big) + \gamma \max_{a \in A\left(s(t+1)\right)} Q(s(t+1), a) - Q\big(s(t), a(t)\big)\right], \quad (4)$$

where the learning rate $\alpha(t) > 0$ is the key parameter for the algorithm convergence: if $\sum_{t=1,\dots,\infty} \alpha(t) = \infty$ and $\sum_{t=1,\dots,\infty} \big(\alpha(t)\big)^2 < \infty$, the estimate (4) converges to the optimal action-value function as $t \to \infty$ (Sutton & Barto, 1998). The action is then decided based on the current estimate of the state-action value function, and the current best policy is:

$$u(s) = \underset{a \in A(s)}{\operatorname{argmax}}\, Q(\mathrm{s}, \mathrm{a}), s \in S. \quad (5)$$

As the estimate (4) converges to the optimal action-value function, the policy (5) converges to an optimal policy.

To guarantee a certain degree of *exploration* of the state space set, an ε-greedy rule is followed: in state $s \in S$, the current best action (5) is taken by the controller with probability $1 - \varepsilon$, where $\varepsilon \in (0,1)$ is the exploration rate; with probability ε a random action $a \in A(s)$ is chosen, i.e.:

$$u(s) \leftarrow \begin{cases} \underset{a \in A(s)}{\operatorname{argmax}}\, Q(\mathrm{s}, \mathrm{a}), \text{with prob. } 1 - \varepsilon \\ \mathrm{rand}\{a \in A(s)\}, \text{with prob. } \varepsilon \end{cases}, s \in S. \quad (6)$$

A large value of ε guarantees that different policies with respect to the current best one are explored, and thus avoids that the system remains stuck in a local minimum. A small value of ε, on the other hand, let the system choose the best action based on the current estimates of the action-value function and favor the exploitation of the current best policy.

The choice of $\alpha(t)$ and ε depends on the specific application.

3 Problem formulation and proposed controller

This Section models the CMB Resource Allocator as a MDP (Section 3.1), describes the proposed approximated MDP (Sections 3.2 and 3.3) and defines the RL control algorithm (Section 3.4).

3.1 Problem formulation

In this Section, the problem model introduced in (Oddi et al., 2013) is summarized.

Let K be the set of different cloud services available to the users, and let C be the set of different cloud providers interfaced with the CMB. Different resources may be provided by the cloud providers; for the sake of simplicity, storage and processing resources are considered.

Each cloud provider is characterized by the amount of pre-purchased resources, hereafter referred to as static resources and denoted with $W_c, c = 1, \dots, C$.

On-demand resources can be purchased from any provider by paying an additional cost, which is proportional to the purchased quantity. Each service is characterized by the amount of resources needed to meet the QoS requirements, denoted with $w_k, k = 1, \dots, K$. Thus, w_k represents the total amount of resources which needs to be allocated by the CMB on a cloud provider.

It is assumed that, for each service type $k = 1, \dots, K$, the service requests arrive according to a Poisson distribution in time with intensity ν_k and their duration is exponentially distributed with mean termination frequency μ_k.

The state $s(t) \in S$ is defined as a vector that represents the number of allocated services of each type k, on each cloud c, at time t:

$$s(t) = \left(n_{ck}(t)\right)_{\substack{c=1,\dots,C \\ k=1,\dots,K}}, \text{ with } n_{ck} = 0,1,2, \dots . \tag{7}$$

The total amount of resources required at the time *t* by all the allocated services on cloud provider *c* is then:

$$\eta_c\left(s(t)\right) = \sum_{k=1,\dots,K} n_{ck}(t) \cdot w_k \,, c = 1, \dots, C. \tag{8}$$

The state space *S* is defined as

$$S = \left\{ s = (n_{ck})_{\substack{c=1,\dots,C \\ k=1,\dots,K}} \middle| \eta_c(s) = \sum_{k=1,\dots,K} n_{ck} \cdot w_k \leq W_c \right\} = \{s_1, s_2, \dots, s_N\}. \tag{9}$$

Since the considered resources (storage and processing) are additive, and since the static resources W_c are finite, the discrete state space S is finite as well, with N states. It is also considerd an initial state distribution Σ over the state space S. With little abuse of notation, the number of services of type k on cloud c in state s is denoted with $n_{ck}(s)$.

Considering that the CMB can request additional on-demand resources, in each state s it is always possible to accept a new request of a service k by allocating additional on-demand resources on a given cloud provider (i.e., a new request is never rejected). Let δ_{ck} be a $K \times C$ vector of zeros but the element associated to the cloud c and the type k equal to 1. Then, in each state s a request of service k can be allocated on the static resources of cloud c if and only if $s + \delta_{ck} \in S$; otherwise, on-demand resources have to be purchased.

The action space A is defined as follows:

$$A(s_i) = \Bigg\{ a = \left(a_{ck}^{j}\right)_{\substack{c=1,\dots,C \\ k=1,\dots,K \\ j\in\{p,o\}}} \Big| a_{ck}^{p} \in \{0,1\} \text{ if } s_i + \delta_{ck} \in S, a_{ck}^{p} = 0 \text{ if } s_i + \delta_{ck} \notin S, a_{ck}^{o} \in$$

$$\{0,1\}, c = 1, \dots, C, \sum_{i\in\{p,o\}} \sum_{c=1,\dots,C} a_{ck}^{i} = 1, k = 1, \dots, K \Bigg\}, i = 1, \dots, N. \quad (10)$$

where a_{ck}^{p} denotes the action of mapping a request of service k on the static resources of cloud c, and a_{ck}^{o} denotes the action of mapping a request of service k on on-demand resources of cloud c. By fixing one action for each state, the controller defines a policy. The policy u is admissible if $u(s_i) \in A(s_i), i = 1, \dots, N$. The policy space U is the set of all the admissible policies:

$$U = \left\{ \left(u(s_i)\right)_{i=1,\dots,N} | u(s_i) \in A(s_i), i = 1, \dots, N \right\}. \quad (11)$$

The *greedy policy*, denoted with $u_g = \left(u_g(s_i)\right)_{i=1,\dots,N}$, is the policy which, in every state, selects the action that mapping each service on the available cloud with the maximum reward. The greedy policy is an admissible policy, i.e., $u_g \in U$.

The transitions between states occur according to the arrival and termination frequencies and to the admission decisions:

- when the system under policy u is in state s and the action $u(s)$ is to allocate the service k on the static resources of cloud c, the transition from state s to state $s + \delta_{ck}$ occurs with frequency λ_k (arrival frequency of service k request);
- since a mapped service k on cloud c terminates with termination frequency μ_k, when the system is in state s the transition from state s to state $s - \delta_{ck}$ occurs with frequency $n_{ck}(s) \cdot \mu_k$, regardless of the policy u;
- when the system under policy u is in state s and the action $u(s)$ is to allocate the service k on the on-demand resources of cloud c, the self-transition occurs with frequency λ_k (arrival frequency of service k request).

Finally, the reward $r(s, a, s')$ gained by the system when it is in state s, takes action a and arrival state is s' is defined as follows:

$$r\left(s_i, a, s_j\right) = \begin{cases} r_{ck}^{p} \text{ if } s_j = s_i + \delta_{ck}, c = 1, \dots, C, k = 1, \dots, K \\ r_{ck}^{o} \text{ if a service } k \text{ request occurs}, s_j = s_i, a_{ck}^{o} = 1 \\ \qquad\qquad c = 1, \dots, C, k = 1, \dots, K \\ 0 \text{ otherwise} \end{cases}$$

$$, i, j = 1, \dots, N, a \in A(s_i), \quad (13)$$

where r_{ck}^{p} and r_{ck}^{o} are the profits obtained allocating a service of class k on cloud c on static and on-demand resources, respectively.

The MDP described so far has continuous-time transitions determined by the arrival and termination frequencies. It is possible to define discrete-time transitions, i.e., to define the transition matrix T, by following a two-step procedure:

(1) divide the transition frequencies of each state by a constant f such that $f > \max_{i=1,\dots,N}\left(\sum_{k=1,\dots,K}\frac{\lambda_k}{f} + \sum_{\substack{k=1,\dots,K\\c=1,\dots,C}}\frac{n_{ck}(s_i)\cdot\mu_k}{f}\right)$;

(2) add self-transitions in such a way that the outgoing probability is 1 for all the states.

The resulting transition probabilities are:

$$t(s_i,a,s_j) = \begin{cases} \frac{\lambda_k}{f}a_{ck}^p \text{ if } s_j = s_i + \delta_{ck}, c = 1,\dots,C, k = 1,\dots,K \\ \frac{n_{ck}(s)\cdot\mu_k}{f} \text{ if } s_j = s_i - \delta_{ck}, c = 1,\dots,C, k = 1,\dots,K, \\ 1 - \sum_{\substack{l=1,\dots,N\\ l\neq i}} t(s_i,a,s_l) \text{ if } s_j = s_i \end{cases}$$

$$i,j = 1,\dots,N, a \in A(s_i). \quad (12)$$

The described procedure is known as *uniformization*, and it can be shown (see, e.g., (Bertsekas, 1987)) that the discrete-time MDP with transition probabilities (12) is equivalent, from the long-term properties viewpoint, to the originating continuous-time MDP.

In this paper, the interest is in maximizing the expected discounted reward (1)[2], which can be also computed as:

$$R(u) = (1-\gamma)\cdot\sum_{i=1,\dots,N} y_u(s_i) r\big(s_i, u(s_i)\big), u \in U, \quad (14)$$

where $y_u(s)$ is the expected discounted sojourn time that the system spends in state s under policy u. The described stationary MDP is ergodic unichain (i.e., under all stationary policies, it is aperiodic and has a single recurrent class and possibly a non-empty set of transient states, see (Puterman, 1994) for details), since the transitions are aperiodic and the transitions due to service terminations are always positive and are independent of the policy (see also (Pietrabissa, 2011b)). Therefore, the expected sojourn times y_u's exist and are finite, and their values can be computed by means of the linear programming formulation of the MPD (Puterman, 1994), which requires the knowledge of the transition matrix.

3.2 Policy restriction

As mentioned in Section 1, this paper adapts the policy restriction mechanism policy proposed in (Pietrabissa, 2008a) to the multi-cloud scenario.

[2] With some awareness, the proposed method is applicable also to the undiscounted and finite-horizon cases.

As observed by simulations, when the load of the clouds is scarce, it is reasonable to accept all the new services, since in those conditions the optimal policy most likely enforces a greedy policy. Thus, if the policy space is restricted by forcing the *greedy-allocation* of all the services when the load of the clouds is below a given threshold, the optimal policy is likely to be included in the restricted policy space. Let $\hat{S}_\rho \subseteq S$ be the subset of the highly-loaded states:

$$\hat{S}_\rho = \left\{ s = (n_{ck})_{\substack{c\in C \\ k\in K}} | \eta_c(s) = \sum_{k\in K} n_{ck} \cdot w_k > \rho \cdot W_c \right\} = \{s_1, s_2, \dots, s_{M(\rho)}\}, \quad (15)$$

where $\rho \in [0,1]$, referred to as *aggregation factor*, is the parameter which establishes the lower limit of the fraction of the cloud resources which are controlled by the greedy policy. In equation (15), it is assumed that the first $M(\rho)$ states are the border states and the remaining $N - M(\rho)$ state are the lightly-loaded states. Clearly, $M(\rho)$ is a non-increasing function of ρ, with $M(0) = N$ and $M(1) = 0$. The states $s \in \hat{S}_\rho$, where decisions are still to be taken by the controller, will be hereafter referred to as *border states*.

In the lightly-loaded states $s_i, i = M(\rho) + 1, \dots, N$, services are mapped in a greedy fashion: whenever it is possible to allocate a service on static resources, the service is mapped on the cloud which has the largest static reward, otherwise it is mapped on the cloud which has the largest on-demand reward. Since no decision has to be taken in the lightly-loaded states, the action space $\hat{A}_\rho(s)$ is restricted to the border states, where it is equal to the original action space (10). Thus it follows that:

$$\hat{A}_\rho(s_i) = A(s_i), i = 1, \dots, M(\rho). \quad (16)$$

The admissible policies, collected in the restricted policy space $\hat{U}_\rho$, are the following:

$$\hat{u} = \left(u(s_1), u(s_1), \dots, u\left(s_{M(\rho)}\right)\right) | u(s_i) \in A(s_i), i = 1, \dots, M(\rho). \quad (17)$$

The transition probabilities of the restricted transition matrix $\hat{T}_\rho$ are the same as the ones in the original MDP (12), except the transitions out-going from the lightly-loaded states, which are now independent of the restricted policy and coincide with the transition probabilities (12) under the greedy action:

$$\hat{t}_\rho(s_i, a, s_j) = \begin{cases} t\left(s_i, u_g(s_i), s_j\right) \text{ if } i > M(\rho) \\ t(s_i, a, s_j) \text{ otherwise} \end{cases}, i = 1, \dots N, a \in A(s_i). \quad (18)$$

Considering the MDP $\hat{\Gamma}_\rho = \{S, \hat{A}_\rho, \hat{T}, r\}$, the expected average reward under policy $\hat{u}$ is:

$$\hat{R}(\hat{u}) = (1-\gamma)\cdot\sum_{i=1,\dots,M(\rho)} y_{\hat{u}}(s_i) r\big(s_i, \hat{u}(s_i)\big) + (1-\gamma)\cdot$$
$$\sum_{s=M(\rho)+1,\dots,N} y_{\hat{u}}(s_i) r\left(s_i, u_g(s_i)\right), \hat{u} \in \hat{U}_\rho. \quad (19)$$

In (19) it is highlighted that the greedy policy u_g is enforced in the lightly-loades states $s_i, i = M(\rho)+1, \dots, N$.

Hereafter, the MDP $\hat{\Gamma}_\rho$ will be referred to as *restricted* MDP, and an optimal policy will be denoted as $\hat{u}_\rho^*$. The following Theorem holds:

***Theorem* 1**. Considering the original MDP Γ under optimal policy $u^* \in U$ and the restricted MDP $\hat{\Gamma}_\rho$ under optimal policy $\hat{u}_\rho^* \in \hat{U}_\rho$, the following inequality holds:

$$R\big(u_g\big) \le \hat{\mathrm{R}}\big(\hat{u}_\rho^*\big) \le R(u^*). \quad (20)$$

Moreover, it follows that $\hat{\mathrm{R}}(\hat{u}_0^*) = R(u^*)$, $\hat{\mathrm{R}}(\hat{u}_1^*) = R\big(u_g\big)$ and $\hat{\mathrm{R}}\big(\hat{u}_{\rho_1}^*\big) \le \hat{\mathrm{R}}\big(\hat{u}_{\rho_2}^*\big)$ if $\rho_1 < \rho_2, \forall \rho_1, \rho_2 \in [0,1]$. ■

Proof. Consider the following subsets of the original policy space, where the greedy policy is enforced in the lightly-loaded states:

$$U_{M(\rho)} = \big\{u \in U | u(s_i) = u_g(s_i), i = M(\rho)+1, \dots, N\big\} \subseteq U, \rho \in [0,1]. \quad (21)$$

By definition of the restricted MDP $\hat{\Gamma}_\rho$, the greedy policy is enforced in the lightly-loaded states of $\hat{\Gamma}_\rho$. Therefore, $\hat{\Gamma}_\rho$ under the restricted policy $\hat{u} = (\hat{u}_1, \dots, \hat{u}_M) \in \hat{U}_\rho$ is equivalent to the original MDP Γ under the policy $u_{eq} \coloneqq \big(\hat{u}_1, \dots, \hat{u}_{M(\rho)}, u_g(s_{M+1}), \dots, u_g(s_N)\big) \in U_{M(\rho)}$.

By comparing equations (14) and (19), it follows that $R\big(u_{eq}\big) = \hat{\mathrm{R}}(\hat{u})$. Given that $u_g \in U_{M(\rho)} \subseteq U$, equation (20) follows.

By definition (21), since $M(\rho)$ is a non-increasing function of ρ with $M(0) = N$ and $M(1) = 0$, it follows that $U_{M(0)} = U$, $U_{M(1)} = \big\{u_g\big\}$ and $U_{M(\rho_2)} \subseteq U_{M(\rho_1)}$ if $\rho_1 > \rho_2, \rho_1, \rho_2 \in [0,1]$; therefore, the second part of the Theorem holds. ■

3.3 State aggregation

The restricted MDP improves the scalability by reducing the policy space. To have a more effective scalability improvement, an *aggregated* MDP is considered, defined as $\bar{\Gamma}_\rho = \big\{\bar{S}_\rho, \hat{A}_\rho, \bar{T}_\rho, \bar{r}\big\}$, where the lightly-loaded states $s_i, i = M(\rho)+1, \dots, N$, are aggregated into one single state s_{agg}.

The aggregated state space is then:

$$\bar{S}_\rho = \{s_1, s_2, \dots, s_{M(\rho)}\} \cup \big\{s_{agg}\big\}. \quad (22)$$

The aggregated action space is defined over the border states only (as with the lightly-loaded states in the restricted case, the greedy actions are always performed in the aggregated state s_{agg}). Therefore, the aggregated action and policy spaces are equal to the restricted action space $\hat{A}_\rho$ and to the restricted policy space $\hat{U}_\rho$, respectively.

Unfortunately, for the aggregated state s_{agg} it is not possible to explicitly define the transition probabilities and the associated reward, since, when the system is in state s_{agg}, it is actually in one of the aggregated states $s_i, i = M(\rho)+1, \dots, N$. Hereafter, the transition matrix and the reward function for the aggregated MDP under a restricted policy $\hat{u} \in \hat{U}$ will be defined by assuming that the sojourn times of $\hat{\Gamma}_\rho$ under $\hat{u}$ are known.

The aggregated transition matrix is denoted as $\bar{T}_\rho = \left(\bar{t}_\rho(s,a,s')\right)_{\substack{s,s' \in \bar{S}_\rho \\ a \in \hat{A}_\rho(s)}}$; the transitions between border states are the same as the transitions $\hat{t}_\rho(s,a,s')$ defined by eq. (18).

The sojourn time in the aggregate state under policy $\hat{u}$ is equal to:

$$y_{\hat{u}}(s_{agg}) = \sum_{i=M(\rho)+1,\dots,N} y_{\hat{u}}(s_i). \tag{23}$$

Then, the out-going transition probabilities are computed as:

$$\left[\bar{t}_\rho(s_{agg}, s_j)\right]_{\hat{u}} = \sum_{i=M(\rho)+1,\dots,N} \left(\frac{y_{\hat{u}}(s_i)}{y_{\hat{u}}(s_{agg})} \cdot \hat{t}_\rho\left(s_i, u_g(s_i), s_j\right) \right), j = 1, \dots, M(\rho), \tag{24}$$

where $[t]_{\hat{u}}$ denotes the transition probability t under policy $\hat{u}$. The in-going transitions are computed as:

$$\bar{t}_\rho\left(s_j, \hat{u}(s_j), s_{agg}\right) = \sum_{i=M(\rho)+1,\dots,N} \hat{t}_\rho\left(s_j, \hat{u}(s_j), s_i\right), j = 1, \dots, M(\rho). \tag{25}$$

The self-transition probability $\left[\bar{t}_\rho(s_{agg}, s_j)\right]_{\hat{u}}$ is always computed as in eq. (12) by following the uniformization procedure.

Similarly, the reward of the border states is the same as the reward of the original MDP, i.e., $\bar{r}\left(s_i, a(s_i)\right) = r\left(s_i, a(s_i)\right), i = 1, \dots, M$, whereas the sojourn times of the $\hat{\Gamma}_\rho$ should be known in advance to compute the reward of the aggregate state under policy $\hat{u}$:

$$\left[\bar{r}(s_{agg})\right]_{\hat{u}} = \sum_{i=M(\rho)+1,\dots,N} \left(\frac{y_{\hat{u}}(s_i)}{y_{\hat{u}}(s_{agg})} \cdot \left(s_i, u_g(s_i)\right) \right), \tag{26}$$

where $[r]_{\hat{u}}$ denotes the reward r under policy $\hat{u}$.

The expected average reward of $\bar{\Gamma}_\rho$ under policy $\hat{u}$ is then:

$$\begin{aligned} \bar{R}(\hat{u}) &= (1-\gamma) \cdot \sum_{i=1,\dots,M(\rho)} y_{\hat{u}}(s_i) r\left(s_i, \hat{u}(s_i)\right) \\ &+ (1-\gamma) \cdot y_{\hat{u}}(s_{agg}) \left[\bar{r}(s_{agg})\right]_{\hat{u}}, \hat{u} \in \hat{U}_\rho. \end{aligned} \tag{27}$$

***Property* 1**. The aggregated MDP $\bar{\Gamma}_\rho = \{\bar{S}_\rho, \hat{A}_\rho, \bar{T}_\rho, \bar{r}\}$ under policy $\hat{u}$ generates the same expected reward of the restricted MDP $\hat{\Gamma}_\rho = \{S, \hat{A}_\rho, \hat{T}_\rho, r\}$ under policy $\hat{u}$: $\hat{R}(\hat{u}) = \bar{R}(\hat{u})$. ■

Proof. By substituting equation (26) into equation (27), which computes $\bar{R}(\hat{u})$, equation (19) , which computes $\hat{R}(\hat{u})$, is straightforwardly obtained. ■

From Property 1, the following result follows:

***Corollary* 1**. An optimal policy $\hat{u}_\rho^*$ of the restricted MDP $\hat{\Gamma}_\rho$ is also an optimal policy of the aggregated MDP $\bar{\Gamma}_\rho$. ■

Corollary 2 follows from Theorem 1 and Corollary 1:

***Corollary* 2**. Considering the original MDP Γ under optimal policy $u^*$and the aggregated MDP $\bar{\Gamma}_\rho$ under optimal restricted policy $\hat{u}_\rho^*$, the following inequality holds:

$$R(u_g) \leq \bar{R}(\hat{u}_\rho^*) \leq R(u^*). \tag{28}$$

Moreover, $\bar{R}(\hat{u}_0^*) = R(u^*)$, $\bar{R}(\hat{u}_1^*) = R(u_g)$ and $\bar{R}(\hat{u}_{\rho_1}^*) \leq \bar{R}(\hat{u}_{\rho_2}^*)$ if $\rho_1 > \rho_2, \forall \rho_1, \rho_2 \in [0,1]$. ■

The aggregated MDP $\bar{\Gamma}_\rho$ cannot be solved by DP methods, since the computation of the transition matrix and of the reward under a given policy $\hat{u}$ requires the knowledge of the sojourn times of the restricted MDP $\hat{\Gamma}_\rho$ under the policy $\hat{u}$. However, the results of this Section will be exploited by the RL control algorithm defined in the next Section.

3.4 RL control algorithm

The aggregated MDP $\bar{\Gamma}_\rho$ can be solved by means of RL algorithms, since RL algorithms does not rely on the knowledge of the transition matrix and of the reward: as the controller takes an action in a given state, it observes from the environment both the next state and the immediate reward.

Since RL algorithms converge to the optimal policy, under given assumptions, they can be used to find the optimal restricted policy acting on the aggregated MDP. From Corollary 2, it follows that the value of the aggregation factor ρ determines the trade-off between scalability and performance of the algorithm: the number of aggregated states grows with ρ, while the expected reward generated by the optimal restricted policy decreases with ρ.

In the simulations of Section 4, a Q-learning algorithm is implemented that performs the update rule (4) over the aggregate state space $\bar{S}_\rho$ and the restricted action set $\hat{A}_\rho$. The proposed algorithm implements an ε-greedy policy in the border states. In the aggregate state the controller always performs greedy actions and updates the action-value function of the aggregate state accordingly (since the greedy actions are always taken, there is one action-value function in s_{agg}, and the single action-value function coincides with the value function).

The learning rate $\alpha(t)$ has the following law:

$$\alpha(t) = \alpha_0 \frac{1}{\lfloor t/t_{exp} \rfloor + 1}, \tag{29}$$

where $\lfloor \cdot \rfloor$ is the lower-integer operator, α_0 is the initial learning rate and t_{exp} is the number of iterations during which the learning rate remains equal to α_0. The learning rate (3) verifies the convergence conditions $\sum_{t=1,\dots,\infty} \alpha(t) = \infty$ and $\sum_{t=1,\dots,\infty} \big(\alpha(t)\big)^2 < \infty$. The learning rate (29) is initially large and then decays with time as the number of state visits grows. In fact, since it is assumed that the environment is initially unknown, an initial exploration is crucial for the learning phase; as the system explores the state space, the estimate of the action-value functions becomes more accurate and $\alpha(t)$ is lowered accordingly.

The values of the algorithm parameters have been tuned by simulation runs and then used in the simulations reported in Section 4.

4 Simulations

This Section presents some numerical simulation results. The two proposed RL algorithms are evaluated against the results of a DP algorithm, and also against a simple greedy policy. DP algorithms, scalability problems apart, find the optimal policy provided that the statistical distribution of the demand is known. The greedy policy is the most immediate policy to implement, and chooses the action that leads to the greatest immediate reward; it is a myopic policy, since it does not consider any time horizon: there is no guarantee that a choice that gives an immediate greater reward leads to a satisfactory expected reward over the whole time horizon.

The algorithms are tested in different scenarios implemented in MATLAB®.

In the first simulation, 3 cloud providers are available and the CMB offers 2 classes of services. Table 1 shows the scenario parameters. Algorithm parameters were selected as follows: $\alpha_0 = 1$, $t_{exp} = 10^3$, $\varepsilon = 0.98$ and $\gamma = 0.89$. The simulation length was set to $3 \cdot 10^5$. All the simulation runs start from empty clouds.

Table 1. Scenario 1 parameters.

Parameter	Value
C	3
K	2
$W_c, c = 1,2,3$	$\{0.3,0.3,0.2\}$
$w_k, k = 1,2$	$\{0.1,0.2\}$
$\lambda_k, k = 1,2$	$\{0.05,0.15\}$
$\mu_k, k = 1,2$	$\{0.5,0.15\}$
$r_{1k}^p, k = 1,2$	$\{1,0.9,1.25\}$
$r_{2k}^p, k = 1,2$	$\{2,0,2.5\}$
$r_{1k}^o, k = 1,2$	$\{0.2,0.3,0.25\}$
$r_{2k}^o, k = 1,2$	$\{0.35,0,0.4\}$

In this scenario, the non-aggregated state space size N is small (150 feasible states), so the optimal policy can be computed by DP algorithms.

Fig. 3 shows the results in terms of state space size and of revenue, averaged over 10 iterations. The figure shows that the revenue obtained with the Q-learning algorithm without aggregation is about 92% of the optimal revenue, whereas the revenue obtained with the greedy policy is about 72%. Even in this small scenario, the aggregation appears effective for $\rho = 0.2$ and $\rho = 0.4$: in both cases the revenue is above 80% (about 85% and 82%, respectively) with a state space size equal to 52% and 33% of the non-aggregated state-space size, respectively.

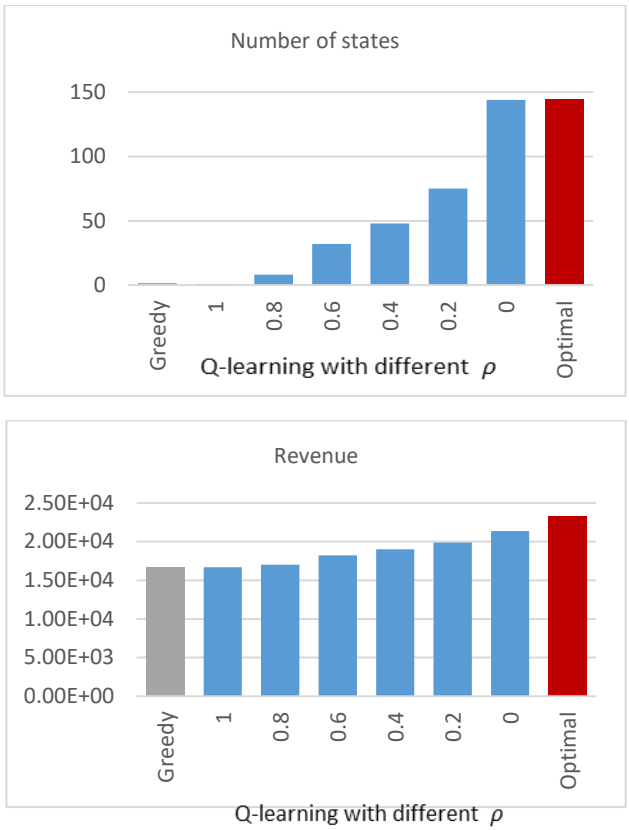

Figure 3. First simulation results: comparison among greedy policy, Q-learning with different levels of aggregation and optimal policy.

The second simulation considers a scenario with more availability of static resources on the third cloud provider states, as indicated in Table 2. Algorithm parameters were selected as follows: $\alpha_0 = 1$, $t_{exp} = 10^5$, $\varepsilon = 0.98$ and $\gamma = 0.89$. The simulation length was set to $6 \cdot 10^5$. All the simulation runs start from empty clouds. In this scenario, the state space size N is much larger ($\sim 10^4$ feasible states), and DP algorithms cannot be used due to the scalability issue. With the proposed aggregation policy, by setting $\rho = 0.6$, the number of states is reduced by one order of magnitude: $M(\rho) + 1 = 2.6 \cdot 10^3$.

Table 2. Scenario 2 parameters.

Parameter	Value
C	3
K	2
$W_c, c = 1,2,3$	$\{0.8,0.8,0.6\}$
$w_k, k = 1,2$	$\{0.1,0.2\}$
$\lambda_k, k = 1,2$	$\{0.5,0.6\}$
$\mu_k, k = 1,2$	$\{0.5,0.15\}$
$r_{1k}^p, k = 1,2$	$\{1,1.2,1.5\}$
$r_{2k}^p, k = 1,2$	$\{2,1.8,1.2\}$
$r_{1k}^o, k = 1,2$	$\{0.2,0.3,0.2\}$
$r_{2k}^o, k = 1,2$	$\{0.25,0.35,0.45\}$

The results, in terms of revenue and of static allocations, averaged over 10 iterations, are shown in Figure 4, which shows similar results to the former simulation; this time, the revenue obtained with the Q-learning policy averaged over 10 simulation runs exceeds the one obtained with the greedy policy by about 22%. The figure also shows that this gain was achieved thanks to a better exploitation of the static resources.

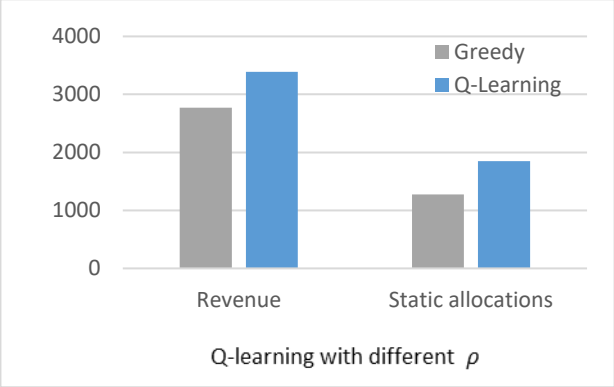

Figure 4. Second simulation results: comparison between greedy and aggregated Q-learning.

5 Conclusions

Cloud Management Brokers (CMB) will play a crucial role in Network Function Virtualization in the near future to allow a more flexible and scalable market. In the considered scenario, the CMB is interfaced with different cloud providers and manages their resources to satisfy the requests of the users while maximizing the CMB owner revenue. Therefore, a multi-cloud resource allocation problem is dealt with in this paper.

Firstly, the resource allocation problem is modelled as a Markov Decision Process (MDP). Then, the MDP is solved by means of a Reinforcement Learning algorithm; finally, a policy reduction and a state aggregation strategies are proposed to cope with the inherent scalability problems of MDPs. Numerical simulation results show that the approximation strategy leads to an effective reduction of the problem size while well-approximating the revenue obtained by the optimal solution.

Currently, the presented algorithm is being implemented in the demonstrator of the T-NOVA research project, but, for a ready-to-market product, further studies are still needed to improve its performances. A research area is to investigate approaches to further improve the algorithm scalability in order to reduce the storage and computational requirements, e.g., by means of machine-learning techniques to approximate the action-value function over the aggregated state space (see (Xu, Zuo, & Huang, 2014) and references therein). This aspect is particularly important due to the so-called *curse of dimensionality* (Bertsekas, 1987), since the state-space dimension may explode in large multi-cloud scenarios. Another area which has to be investigated is the algorithm behavior and the required algorithm modifications in the presence of constraints (e.g., constraints on the distribution of the requests among the cloud providers or on the Quality of Service which has to be grant to the users). A suitable approach may be the lexicographic one (see (Gábor, Kalmár, & Szcpcsvári, 1998; Panfili, Pietrabissa, Oddi, & Suraci, 2016), where the policy improvement step is still aimed at minimizing the cost function but the candidate policies are also evaluated against the constraints, ranked in order of importance.

Acknowledgements

Research supported by the EU project T-NOVA (T-NOVA, 2015), under FP7 ref n 619520, and by the Italian project PLATINO (PLATINO, 2015), under Grant Agreement n°PON01_01007. The authors wish to thank all the members of the project teams of PLATINO and T-NOVA for their valuable contributions to the work.

References

Bertsekas, D. P. (1987). *Dynamic Programming deterministic and stochastic models*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Buyya, R., Ranjan, R., & Calheiros, R. (2010). InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In C.-H. Hsu, L. Yang, J. Park, & S.-S. Yeo (Eds.), *Algorithms and Architectures for Parallel Processing SE - 2* (Vol. 6081, pp. 13–31). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-13119-6_2

Chaisiri, S., Lee, B. S., & Niyato, D. (2012). Optimization of resource provisioning cost in cloud computing. *IEEE Transactions on Services Computing*, *5*(2), 164–177. http://doi.org/10.1109/TSC.2011.7

CompatibleOne. (2015). CompatibleOne: the Open Source Cloud Broker. Retrieved May 20, 2001, from http://www.compatibleone.org/

Di Giorgio, A., Liberati, F., & Pietrabissa, A. (2013). On-board stochastic control of Electric Vehicle recharging. In *52nd IEEE Conference on Decision and Control* (pp. 5710–5715). IEEE. http://doi.org/10.1109/CDC.2013.6760789

FI-Core. (2015). (Future Internet - Core Platform), EU FP7-ICT Large-scale Integrating Project (IP), 2014- 2016, grant agreement no. 632893. Retrieved from

http://cordis.europa.eu/project/rcn/192274_en.html

FIWARE. (2014). (Future Internet Ware), EU FP7-ICT Large-scale Integrating Project (IP), 2011- 2014, grant agreement no. 312826. Retrieved from http://www.fi-ware.eu/

Gábor, Z., Kalmár, Z., & Szcpcsvári, C. (1998). Multi-criteria Reinforcement Learning. In *International Conference on Machine Learning (ICML 1998)* (pp. 197–205). Madison, Wisconsin, USA.

Konstanteli, K., Cucinotta, T., Psychas, K., & Varvarigou, T. (2012). Admission Control for Elastic Cloud Services. *2012 IEEE Fifth International Conference on Cloud Computing*, 41–48. http://doi.org/10.1109/CLOUD.2012.63

Lewis, F., & Vrabie, D. (2009). Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits and Systems Magazine*, *9*(3), 32–50. http://doi.org/10.1109/MCAS.2009.933854

Macone, D., Oddi, G., Palo, A., & Suraci, V. (2013). A dynamic load balancing algorithm for Quality of Service and mobility management in next generation home networks. *Telecommunication Systems*, *53*(3), 265–283. http://doi.org/10.1007/s11235-013-9697-y

Manfredi, S. (2014a). A theoretical analysis of multi-hop consensus algorithms for wireless networks: Trade off among reliability, responsiveness and delay tolerance. *Ad Hoc Networks*, *13*, 234–244. http://doi.org/10.1016/j.adhoc.2011.05.005

Manfredi, S. (2014b). A theoretical analysis of multi-hop consensus algorithms for wireless networks: Trade off among reliability, responsiveness and delay tolerance. *Ad Hoc Networks*, *13*, 234–244. http://doi.org/10.1016/j.adhoc.2011.05.005

Mascolo, S. (1999). Congestion control in high-speed communication networks using the Smith principle. *Automatica*, *35*(12), 1921–1935. http://doi.org/10.1016/S0005-1098(99)00128-4

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., … Turner, J. (2008). OpenFlow. *ACM SIGCOMM Computer Communication Review*, *38*(2), 69. http://doi.org/10.1145/1355734.1355746

NIST. (2013). National Institute of Standards and Technology: Cloud Computing Program, Section 6.1. Retrieved March 20, 2003, from http://www.nist.gov/itl/cloud/6_1.cfm

Oddi, G., Panfili, M., Pietrabissa, A., Zuccaro, L., & Suraci, V. (2013). A Resource Allocation Algorithm of Multi-cloud Resources Based on Markov Decision Process. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science* (Vol. 1, pp. 130–135). IEEE. http://doi.org/10.1109/CloudCom.2013.24

Panfili, M., Pietrabissa, A., Oddi, G., & Suraci, V. (2016). A lexicographic approach to constrained MDP admission control. *International Journal of Control*, *89*(2), 235–247. http://doi.org/10.1080/00207179.2015.1068955

Pietrabissa, A. (2008a). Admission Control in UMTS Networks based on Approximate Dynamic Programming. *European Journal of Control*, *14*(1), 62–75. http://doi.org/10.3166/ejc.14.62-75

Pietrabissa, A. (2008b). An Alternative LP Formulation of the Admission Control Problem in Multiclass Networks. *IEEE Transactions on Automatic Control*, *53*(3), 839–845. http://doi.org/10.1109/TAC.2008.919516

Pietrabissa, A. (2009). A policy approximation method for the UMTS connection admission control problem modelled as an MDP. *International Journal of Control*, *82*(10), 1814–1827. http://doi.org/10.1080/00207170902774233

Pietrabissa, A. (2011a). A Reinforcement Learning Approach to Call Admission and Call Dropping Control in Links with Variable Capacity. *European Journal of Control*, *17*(1), 89–103. http://doi.org/10.3166/ejc.17.89-103

Pietrabissa, A. (2011b). A Reinforcement Learning Approach to Call Admission and Call Dropping Control in Links with Variable Capacity. *European Journal of Control*, *17*(1), 89–103. http://doi.org/10.3166/ejc.17.89-103

Pietrabissa, A., Battilotti, S., Facchinei, F., Giuseppi, A., Oddi, G., Panfili, M., & Suraci, V. (2015). Resource management in multi-cloud scenarios via reinforcement learning. In *2015 34th Chinese Control Conference (CCC)* (Vol. 2015-Septe, pp. 9084–9089). IEEE. http://doi.org/10.1109/ChiCC.2015.7261077

PLATINO. (2015). Platform for Innovative Services in Future Internet, Italian Ministry of University and Research (MIUR) PLATINO project, Grant Agreement n° PON01_01007. Retrieved May 20, 2001, from http://www.progettoplatino.it/

Puterman, M. L. (1994). *Markov Decision Processes*. (M. L. Puterman, Ed.). New York: John Wiley & Sons, Inc. http://doi.org/10.1002/9780470316887

Rennie, J., & Mccallum, A. K. (1999). Using Reinforcement Learning to Spider the Web Efficiently. *ICML*, *99*, 335–343.

Sefraoui, O., Aissaoui, M., & Eleuldj, M. (2012). OpenStack: Toward an Open-source Solution for Cloud Computing. *International Journal of Computer Applications*, *55*(3), 38–42. http://doi.org/10.5120/8738-2991

Sirocco. (2015). Sirocco project: an open-source multi-cloud manager. Retrieved May 20, 2001, from http://sirocco.projects.ow2.org/xwiki/bin/view/Main

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: an introduction*. MIT Press, Cambridge, MA.

Tan, H., Balajee, K., & Lynn, D. (2014). Integration of evolutionary computing and reinforcement learning for robotic imitation learning. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 407–412). IEEE. http://doi.org/10.1109/SMC.2014.6973941

T-NOVA. (2015). Network Functions as-a-Service over Virtualised Infrastructures, EU FP7-ICT Large-scale Integrating Project (IP), 2014- 2016. Retrieved May 20, 2011, from http://www.t-nova.eu/

Woo, S. S., & Mirkovic, J. (2014). Optimal application allocation on multiple public clouds. *Computer Networks*, *68*, 138–148. http://doi.org/10.1016/j.comnet.2013.12.001

Wu, L., Kumar Garg, S., & Buyya, R. (2012). SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments. In *Journal of Computer and System Sciences* (Vol. 78, pp. 1280–1299). http://doi.org/10.1016/j.jcss.2011.12.014

Xu, X., Zuo, L., & Huang, Z. (2014). Reinforcement learning algorithms with function approximation: Recent advances and applications. *Information Sciences*, *261*, 1–31. http://doi.org/10.1016/j.ins.2013.08.037

Yang, X., Liu, D., & Wang, D. (2014). Reinforcement learning for adaptive optimal control of unknown continuous-time nonlinear systems with input constraints. *International Journal of Control*, *87*(3), 553–566. http://doi.org/10.1080/00207179.2013.848292