Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: Towards a Fully Automated Workflow

Xavier Dutreilh†, Sergey Kirgizov*, Olga Melekhova*, Jacques Malenfant*, Nicolas Rivierre† and Isis Truck‡
*Université Pierre et Marie Curie – Paris 6 CNRS, UMR 7606 LIP6, 4 place Jussieu, Paris, 75005, France
Email: {Olga.Melekhova, Jacques.Malenfant}@lip6.fr
†Orange Labs, 38-40 rue du Général Leclerc, Issy-les-Moulineaux, 92130, France
Email: {xavier.dutreilh, nicolas.rivierre}@orange-ftgroup.com
‡LIASD – EA 4383, Université Paris 8, 2 rue de la Liberté, Saint-Denis Cedex, 93526, France
Email: truck@ai.univ-paris8.fr

***Abstract*—Dynamic and appropriate resource dimensioning is a crucial issue in cloud computing. As applications go more and more 24/7, online policies must be sought to balance performance with the cost of allocated virtual machines. Most industrial approaches to date use *ad hoc* manual policies, such as threshold-based ones. Providing good thresholds proved to be tricky and hard to automatize to fit every application requirement. Research is being done to apply automatic decision-making approaches, such as reinforcement learning. Yet, they face a lot of problems to go to the field: having good policies in the early phases of learning, time for the learning to converge to an optimal policy and coping with changes in the application performance behavior over time. In this paper, we propose to deal with these problems using appropriate initialization for the early stages as well as convergence speedups applied throughout the learning phases and we present our first experimental results for these. We also introduce a performance model change detection on which we are currently working to complete the learning process management. Even though some of these proposals were known in the reinforcement learning field, the key contribution of this paper is to integrate them in a real cloud controller and to program them as an automated workflow.**

***Keywords*-Cloud computing; virtual machine allocation; reinforcement learning; autonomic computing.**

I. Introduction

Dimensioning resources to applications appropriately is a crucial issue in cloud computing. As applications go more and more 24/7, online policies must be sought to balance performance offered to clients with the cost of virtual machines (VM) for the service provider by making their allocation following closely the workload. Most industrial approaches to date use *ad hoc* manually determined policies, such as threshold-based ones where a low threshold on performance triggers more allocation while a high one triggers a reduction in the number of allocated VMs. Providing good thresholds proved to be tricky and hard to automatize to fit every application requirement [1].

Research is being done to apply automatic decision-making approaches, such as reinforcement learning (RL) [2]. These approaches are particularly well-suited to cloud computing as they don't require the *a priori* knowledge of the application performance model, but rather learn it as the application runs. Yet, RL faces a lot of problems to go to the field [3][4], such as: having good policies in the early phases of learning, time for the learning to converge to an optimal policy and coping with changes in the application performance behavior over time. In this paper, we propose to deal with these problems using appropriate initialization for the early stages, convergence speedups applied throughout the learning phases and performance model change detection. Even though some of these proposals were known in the RL field, the key contribution of this paper is to integrate them in a real cloud controller and to program them as an automated workflow.

We present our first results towards this automated learning management workflow. Section II introduces the resource allocation problem for cloud computing. Section III presents the formulation of the problem in the Q-learning framework, as we have modeled it for a private cloud deployed at Orange Labs. Section IV then presents the core contribution of the paper, the implementation workflow meant to bring RL to real cloud computing infrastructures. Section V then compares to the related work and the conclusion follows. Throughout the paper, experimental results are shown to back up the proposals.

II. Resource allocation in clouds

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [5]. This model promotes availability and is composed of three delivery models: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). IaaS designates the provision of IT and network resources, such as processing, storage and bandwidth as well as management software. PaaS designates the deployment of applications created using particular programming languages and tools supported by a provider onto his own cloud infrastructure.

 ISBN: 978-1-61208-134-2

SaaS designates the use of applications running on a cloud infrastructure. Companies can use clouds either to run data-processing applications (from development tools like continuous integration suites to business tools as video transcoders), transaction-processing software (including social networks and e-commerce websites) or event-processing systems (as fraud detection tools in the financial market). The cloud is appealing to them because of its ability to reduce capital expenditures and increase return on investment since the traditional model where physical hardware and software resources were bought and amortized in the long term is no more.

Many web applications (like social networks) face workload and system changes that affect their performance during their lifetime. First, they have to cope with large fluctuating loads. In predictable situations (like the promotion of a new feature), resources can be provisioned in advance through the use of proven techniques like capacity planning. However, for unplanned spikes (due to slashdotting for instance) and unplannable events (e.g., a rise in popularity after a natural disaster), auto-scaling appears to be the way to automatically adjust resources allocated to applications based on their needs at any given time. In addition, auto-scaling seems to be a lot more justified when it comes to handle changes in applications (following software improvements) and the cloud platform itself (e.g., hardware and software upgrades). The core features of auto-scaling are a pool of available resources that can be pulled or released on-demand and a control loop to monitor the system and decide in real time whether it needs to grow or shrink. Auto-scaling is offered in PaaS environments by providers like Google App Engine and Heroku but applications developed specifically for these platforms are tied to them. In that way, IaaS appears more flexible since users are given free access to virtualized hardware, relying on providers like Amazon [6] and Rackspace or open-source projects like OpenNebula and OpenStack to instantiate VMs. IaaS issues however are the lack of widely adopted standards, although initiatives, such as DMTF and OGF OCCI, are active toward the definition of virtualization formats and IaaS APIs; and automation since developers must build the machinery, or use third party tools, such as RightScale and Claudia . This paper focuses primarily on resource allocation policies that could be used in IaaS and PaaS management layers to perform auto-scaling.

Resource allocation and policies

Fostered by autonomic computing concepts, allocating resources to applications in clouds has been the subject of several works in recent years. As a decision-making problem, the allocation of VMs to an application consists in regularly observing the workload w (in request per second), the current number of allocated VMs u and the current performance p as the average waiting time of requests in seconds and from that, decides to allocate more VMs or deallocate some, in order to maintain the performance p as close as possible to a target performance P given by the SLA of the application while minimizing the costs for the service provider.

Two types of policies have drawn attention in recent works:

1) Threshold-based policies, where upper and lower bounds on the performance trigger adaptations, and where some amount of resources are allocated or deallocated (typically one VM at a time);
2) Sequential decision policies based on Markovian decision processes (MDP) models and computed using, for example, reinforcement learning.

Threshold-based policies are very popular among on-the-field cloud managers. The simplicity and intuitive nature of these policies make them very appealing. However, with the growing complexity of applications, setting thresholds is a per-application task and can be very tricky, especially when it comes to find corrective actions for all possible states [1][7] and deal with performance model changes. As an alternative to manual threshold-based policies, modeling the system as an MDP allows computing policies that can take into account the inertia of the system, such as fixed costs to allocate or deallocate VMs, which favors retaining the same number of VMs when the variation in the workload does not last enough time to amortize these fixed costs. Such compromises are the cornerstones of sequential decision making.

III. The basic RL problem

We now introduce the formulation of the resource allocation problem for clouds as an MDP and the corresponding Q-learning resolution approach implemented in our VirtRL controller [1].

A. Resource allocation as an MDP

With long lasting executions, applications become more and more subject to changes, affecting their end-user performance. To cope with such changes in a decent amount of time and minimize SLA violations, our controller takes resource allocation decisions regularly. As such and as the decisions themselves influence each others over time, the decision-making is modeled as an MDP [8]. Coarsely speaking, an MDP involves a decision agent that repeatedly observes the current state s of the controlled system, takes a decision a among the ones allowed in that state and then observes a transition to a new state s' and a reward r that will drive its decisions. As their name indicates, MDPs are stochastic. Hence, the new state and sometimes also the reward observed from the transition obey probability distributions that characterize the behavior of the underlying controlled system.

The MDP that models our approach to the VM allocation problem is defined as $\mathcal{M} = \langle S, A, T, R, \beta \rangle$ where:

- $S = \{(w, u, p) \mid 0 \leq w \leq W_{max} \wedge 0 \leq u \leq U_{max} \wedge 0 \leq p \leq P_{max}\}$ is the state of the MDP where:
  - $w \in \mathbb{N}$ is the workload in number of requests per second, bounded by $W_{max} = 40$;
  - $u \in \mathbb{N}$ is the current number of homogeneous VMs allocated to the application, bounded by $U_{max} = 10$;
  - $p \in \mathbb{R}$ is the performance expressed as the average response time to requests in seconds, bounded by a value P_{max} chosen from experimental observations.

 ISBN: 978-1-61208-134-2

- $A = \{a \in \mathbb{Z} \mid A_{min} \leq a \leq A_{max}\}$ is the action set which consists in adding, maintaining or reducing the number of homogeneous VMs allocated to the application. The actions have been bounded between $A_{min} = -1$ and $A_{max} = 10$ in our experimental setup;
- $T : S \times A \times S \rightarrow [0, 1]$ is the probability distribution $P(s'|s, a)$ of a transition to new state s' given that the system is in state s and action a is chosen;
- $R : S \times A \rightarrow \mathbb{R}$ is the cost function expressing the expected reward when the system is in state s and action a is taken. When stochastic, it can be expressed as $R : S \times A \times \mathbb{R} \rightarrow [0, 1]$, the probability distribution $P(r|s, a)$ of observing a reward r when the system is in state s and action a is taken;
- $\beta, 0 < \beta < 1$ is a discount factor used to finitely evaluate the overall expected reward for an infinite sequence of decisions. The value $\beta = 0.45$ has been used throughout our experiments.

When the functions T and R can be determined prior to the execution of the controlled system, traditional dynamic programming (DP) algorithms, such as value iteration [2][8] can be applied to find an optimal policy. Value iteration solves the following equation expressing the expected total reward over infinite horizon (for the deterministic reward case):

$$V^*(s) = \max_a \left[R(s, a) + \beta \int_{s' \in S} T(s, a, s') V^*(s') \right] \quad (1)$$

and then the optimal allocation policy is given by:

$$\pi^*(s) = \operatorname*{argmax}_a \left[R(s, a) + \beta \int_{s' \in S} T(s, a, s') V^*(s') \right] \quad (2)$$

The value iteration algorithm does this by successive approximations until a predefined error bound ϵ is reached:

$(\forall s \in S)$, initialize $V_0(s)$
t := 0
loop
  $t := t + 1$
  foreach $s \in S$
    foreach $a \in A$
      $Q_t(s, a) := R(s, a) + \beta \int_{s' \in S} T(s, a, s') V_{t-1}(s)$
  $\pi_t(s) := \operatorname{argmax}_a \ Q_t(s, a)$
  $V_t(s) := Q_t(s, \pi_t(s))$
until $\sup_s \ |V_t(s) - V_{t-1}(s)| < \epsilon$
return π_t

B. Resolution through Q-learning

The advantage of traditional DP algorithms is that policies are computed offline. The decision-making at runtime then simply amounts to applying the precomputed policy π^* to the sequence of observed states to provide the corresponding actions. However, T and R are often very difficult to estimate. This can require lengthy experimentation and measurement processes upon the actual controlled system and it must be redone each time a modification to the system may change the probability distributions of its transitions or rewards.

To address these limitations, reinforcement learning has been proposed to learn these as the controlled system operates and as the controller is making decisions to learn from experience. Among the different reinforcement learning approaches [2], the Q-learning is based on the equation for $Q(s, a)$ derived from the value function (see equation 1) and appearing in the inner loop of the value iteration algorithm. It turns out that the function $Q(s, a)$, or Q-function, is easy to learn from experience. Given a controlled system, the learning agent repeatedly observes the current state s, takes an action a and then a transition occurs and it observes the new state s' and the reward r. From these observations, it can update its estimation of the Q-function for state s and action a with:

$$Q[s, a] := (1 - \alpha) Q[s, a] + \alpha \left(r + \beta \max_a Q[s', a'] \right) \quad (3)$$

where α is the rate of learning, balancing the weight of what has already been learned with the weight of the new observation. Throughout our experiments, we have used the value $\alpha = 0.8$. The basic Q-learning algorithm is then [2]:

$(\forall s \in S)(\forall a \in A(s))$, initialize $Q(s, a)$
s := the initial observed state
loop
  Choose $a \in A(s)$ according to a policy derived from Q
  Take action a and observe next state s' and reward r
  $Q[s, a] := (1 - \alpha) Q[s, a] + \alpha \left(r + \beta \max_a Q[s', a'] \right)$
  $s := s'$
end loop
return $\pi(s) = \operatorname{argmax}_a Q(s, a)$

C. Experimental results

In order to experiment the Q-learning, we have defined a reward function as follows. Given a state $s = (w, u, p)$, an action a, the next state $s' = (w', u', p')$ and a target performance P_{SLA} (notice that $u' = u + a$), $CO(a)$ represents the cost of acquiring and renting the VMs, while $PE(s')$ captures the penalties imposed when the target performance is violated:

$$R(s', a) = CO(a) + PE(s')$$

$$CO(a) = \begin{cases} c_i \times a & \text{if } a > 0 \\ 0 & \text{else} \end{cases} + c_f \times u' \times \Delta t$$

$$PE(s') = \frac{p_c}{3600} \times \Delta t \times \begin{cases} \left(1 + \frac{p' - P_{SLA}}{P_{SLA}}\right) & \text{if } p' > P_{SLA} \\ 0 & \text{else} \end{cases}$$

where:

- $c_r = 0.095$ (US$ per hour) is the rental cost of VMs per unit of time (it corresponds to the price of a standard on-demand VM on Amazon EC2 [6]);
- $c_i = \frac{1}{60} c_r$ is the initial one-shot cost of getting a new VM when allocated. It is fixed as a fraction of the rental cost for an hour;
- Δt is the length of the time interval between decisions (40 seconds in our experiments);
- $p_c = 10$ (US$ per hour) is the penalty for the application per unit of time for not providing the level of performance

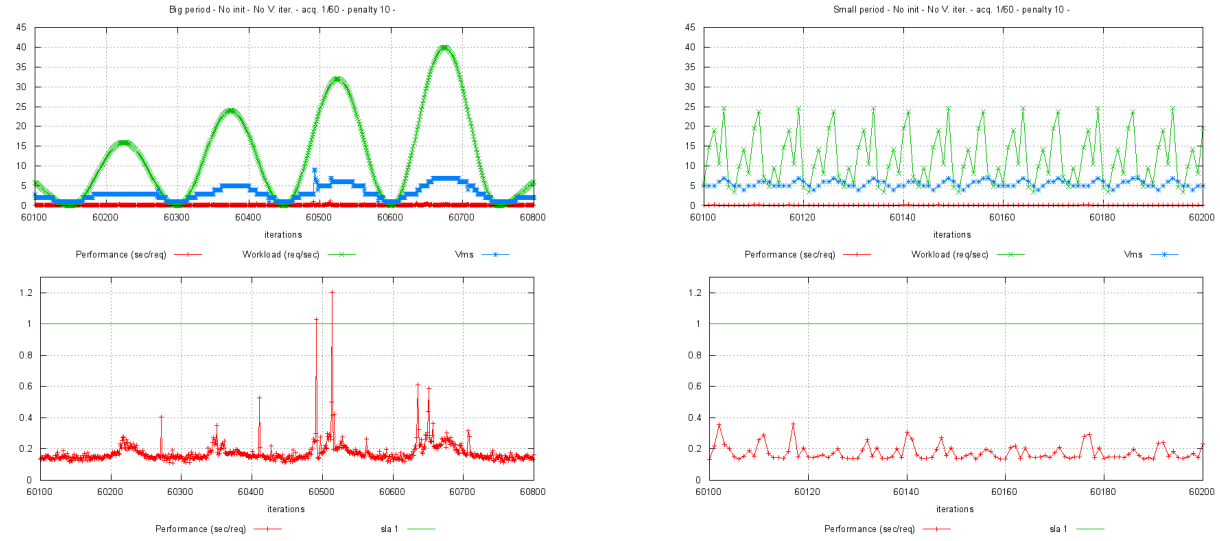

Fig. 1. Comparison of the policies obtained at convergence for two different workload patterns. On the left, a sinusoidal with a dynamic oscillation slope on a long period is used while, on the right, one with a shorter period and a random noise is applied. Beware that the scales are not the same for both cases.

agreed in the SLA. Beware that such a value has direct influence on the aggressiveness and the conservativeness of resource allocation strategies captured by the Q-learning.

This reward function has been used to simulate a cloud executing Olio [9] and the VirtRL decision agent. In this setup, we assume that all VMs used by Olio share a common size which is 1 compute unit (equivalent to 2.66 GHz guaranteed), 256 MB of memory and 20 GB of storage. Figure 1 shows the policies that have been obtained at convergence given two different workload patterns. The first pattern is a long-period sinusoidal function that was used as a baseline for the learning algorithm as its characteristics were easy to learn. The second is a short-period sinusoidal function with random noise, meant to represent the characteristics of real workloads more accurately. Even though these workloads are meant to mimic the cyclic variations of many real-world application workloads, their period has been forced shorter than real ones to put more stress on the learning process.

As we can see from the plots, in both cases, the Q-learning algorithm has converged to an allocation policy that follows the workload with a similar period. The bottom plots give the corresponding performance of the application, the green line giving the target performance while the red line gives the actual one. Besides, to date, our experiments have shown that the time spent in computations and the memory space used to store the representation of the functions are negligible in the cloud computing context.

IV. VirtRL Reinforcement Learning Workflow

Besides the basic learning algorithm, the VirtRL workflow introduces three new activities discussed below:

- Initialization of the Q-function;
- Convergence speedup phases at regular intervals of observations;
- Performance model change detection.

A. Initialization of the Q-learning

The Q-learning algorithm presented above involves an unspecified initialization step for the Q-function. In theory, provided that the mathematical conditions are observed [10], convergence is guaranteed. Pragmatically, however, the distance between the initial Q-function and the one at convergence has two major impacts on the learning process: decisions made during the early phases and time to converge.

In order to apply the control as the learning process is being done, a policy must be followed from which decisions will be chosen and taken on the controlled system. Defining such a policy can be complicated and it also must allow for some exploration of the different possible actions in order for the learning to get the outcome of these actions to discover which is the best one for each state. In the cloud computing context, without prior information about the application, only a standard policy can be applied, such as an ϵ-greedy policy [2], which is a kind of stochastic policy defined as:

$$\pi(s) = \begin{cases} \text{argmax}_a\, Q(s,a) & \text{with probability } 1-\epsilon \\ \text{choose } a \in A(s) \text{ randomly} & \text{with probability } \epsilon \end{cases}$$

For such a policy to give good results, $Q(s,a)$ must be a good approximation of the optimal Q-function at convergence. Until each state has been visited enough times for the learning to update the Q-function to a value approaching the optimal, the current Q-function provides no better information to choose the action than what provides the initial Q-function itself. Hence, for an ϵ-greedy policy to give good results during the first phase of the learning, a good initial Q-function must be found.

As the above value iteration algorithm shows, there is a tight relationship between the value function $V(s)$ and the Q-function. For a given policy π, we have:

$$Q^{\pi}(s,\pi(s)) = R(s,\pi(s)) + \beta \int_{s' \in S} T(s,\pi(s),s') V^{\pi}(s') \quad (4)$$

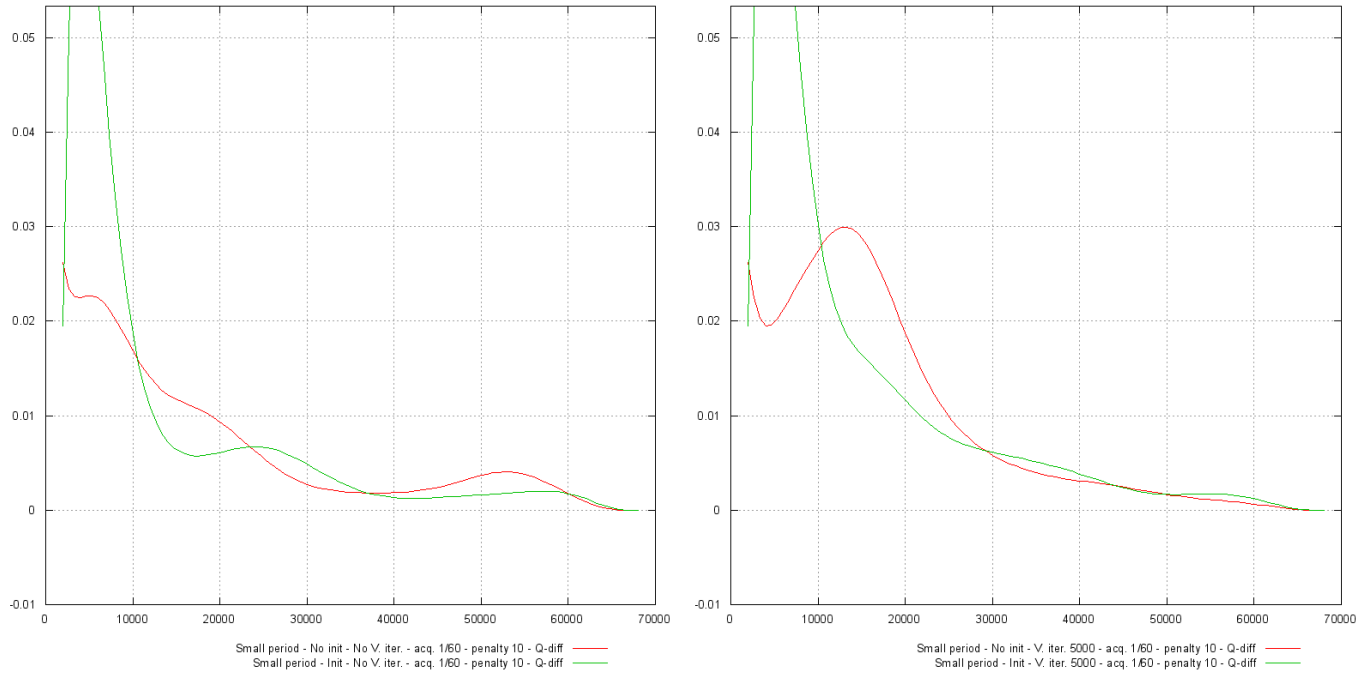

Fig. 2. The two plots show the average difference between two successive Q-functions as learning observations are processed. On the left, no convergence speedup is applied, while on the right, convergence speedups are applied every 5.000 observations. Green curves show the case with initialization, while red ones show learning without initialization. Convergence speedups accentuate the differences between successive Q-functions during the early phases of learning, but diminish them in the middle phase (before 30.000 observations). After 30.000 observations, speedups make little difference.

while the value function itself can be computed for a given policy using the following equation:

$$V^{\pi}(s) = R(s, \pi(s)) + \beta \int_{s' \in S} T(s, \pi(s), s') V^{\pi}(s') \qquad (5)$$

But, of course, we need definitions for the functions R and T, as well as a policy π. Our approach to initialization considers approximate functions $\overline{R}$ and $\overline{T}$, and applies a greedy policy (always choose the best possible action) to find an approximate value function $\overline{V}^{\pi}$, which in turn is put into equation 4 to give an initial Q-function $\overline{Q}$.

Figures 2 and 3 show the result of applying such an initialization, along with convergence speedups to which we return in the next subsection. The approximate reward function that we have chosen is the above reward function truncated of its penalty term (that is much more difficult to estimate). The approximate transition function retains the determinism in the new number of VMs, but considers the next workload and the next performance values as normal random variables centered on the current workload and performance values respectively. The no-initialization case in fact takes $Q \equiv 0$ as the initial Q-function. In Figure 2, we can see that the initialized case exhibits larger average differences at first, but then converges faster to lower differences during a second phase until the overall convergence becomes the same after 30.000 observations approximately. In Figure 3 that is explained in section IV-B, we can see that, after beginning the execution with 10 VMs, the decisions make the allocation come to a less costly number of VMs much faster, though the performance is more volatile. The fact that our current estimated reward function completely neglects the penalties for bad performance, explains this behavior. A bit more guidance than a simple ϵ-greedy policy during this first phase or a more precise estimated reward function, quite simply overcomes these negative effects.

B. Convergence speedups

As presented above, Q-learning learns very progressively the Q-function, updating it only for the visited states and only when they are visited. Compared to the value iteration algorithm, the Q-learning updates the Q-function only for the visited state at each observation, while the value iteration updates the V-function for all states at each iteration. Is it possible to speed up the convergence of Q-learning by using ideas coming from the value iteration algorithm? This is the basic idea behind model-based Q-learning [2][11], which recognizes that there is more to learn from the observations $\langle s, a, r, s' \rangle$ than just from the value of $Q(s, a)$.

Model-based Q-learning uses these observations to estimate, in the statistical sense, the functions R and T. Consider the following measures:

- $C[s, a]$ is the number of times the state s has been observed and the action a taken;
- $T_C[s, a, s']$ is the number of times a transition from state s and action a to the state s' has been observed;
- $R_C[s, a]$ is the sum of the rewards that has been obtained when the state s has been observed and action a taken.

Hence, for each observation $\langle s, a, r, s' \rangle$, these counters are updated as:

$$T_C[s, a, s'] := T_C[s, a, s'] + 1 \quad (6)$$

$$R_C[s, a] := R_C[s, a] + r \quad (7)$$

$$C[s, a] := C[s, a] + 1 \quad (8)$$

Given these statistics, estimators $\bar{T}$ and $\bar{R}$ of the functions T and R are computed as:

$$\bar{T}(s, a, s') = \frac{T_C[s, a, s']}{C[s, a]} \quad (9)$$

$$\bar{R}(s, a) = \frac{R_C[s, a]}{C[s, a]} \quad (10)$$

With these estimators, it becomes possible to compute an estimator $\bar{V}^*$ of the optimal V-function V^* and use this estimator to update the current Q-function. This approach to convergence speedups is the most aggressive compared to other partial sweeps of the state space, such as eligibility traces and sample backups model-based techniques [2].

Figure 3 compares the decisions made during the learning process given that convergence speedups are applied or not. For these experiments, we present results for speedups made at each 5,000 observations. At the top, during the early phases of learning, the differences are due to the initialization, as explained in the previous subsection, as no convergence speedup has been performed yet. In the middle, we see that when convergence speedups have been made during the first period of learning, the decisions follow more closely the workload and the performance is more stable. After 30,000 observations, there is no longer a noticeable difference between the two. Convergence speedups therefore help to have a better policy earlier in the learning process.

C. Model changes detection

With long-lasting application executions, it is less and less possible to ignore that the behavior of applications will change over time, a key assumption when applying MDPs to model them. Indeed, applications can be updated with patches or major software releases that change their performance. Moreover, the workload can also change, for example with a different mix of the possible requests hence also changing its average performance. Reinforcement learning, if the learning is pursued during the whole lifetime of the application, can adapt the policy to relatively smooth changes in its behavior [2]. However, the learning will not be able to react to major changes in the behavioral model promptly, especially if the rate of random actions used for learning is decreased over time, as suggested to ease the convergence.

Our proposal is to use again the information gathered with the model-based reinforcement learning approach to do statistical testing upon the observations used to estimate R and T, between older and newer observations to decide whether a model change has occurred or not. As the form of the probability distributions is not known, non-parametric testing shall be used. And because the two samplings are not independent, appropriate tests such as Wilcoxon signed rank test [12] shall be used. We conjecture that such tests can detect major changes in the behavior of the application and that the more subtle changes undetectable with them can be dealt with by using continuous learning all over the application execution. Experiments to back up this claim are currently undertaken.

V. Related work

Several works apply resource allocation techniques in cloud computing. We concentrate here on the ones that take a disciplined rather than an *ad hoc* approach to the problem.

Among the different works on threshold-based policies, Lim *et al.* propose proportional thresholding to adapt policy parameters at runtime [13]. It consists in modifying the range of thresholds in order to trigger more frequent decisions when necessary. This approach adapts very well to fast changing conditions and is directly integrable into automated agents with stability mechanisms. Although the aforementioned paper gives answers to the latency instability, it still lacks of an adaptation of the power of the decisions to fluctuating workloads.

Xu *et al.* propose a two-level controller [14]. A first level applies fuzzy logic techniques to learn the relationship between workload, resources and performance. It then controls the resource allocation by deciding at a regular interval the level of resources needed by each application, thanks to the fuzzy rules previously learned. The second level applies a simply knapsack technique to allocate the resources to the different applications. Though interesting to learn the behavior of applications, this approach limits itself to homogeneous applications by computing only one set of rules and does not really care about stability. Rao *et al.* with their VCONF [4] apply reinforcement learning but in the context of neural networks, in a way that parallels the work of Xu *et al.* Their neural network represents the relationship between workload, resources and performance used to perform vertical scaling.

Tesauro *et al.* explore the application of reinforcement learning in a sequential decision process [3]. The paper presents two novel ideas: the use of a predetermined policy for the initial period of the learning and the use of an approximation of the Q-function as a neural network. The results are interesting, though dependent on the form of the reward function. Besides that, the initial learning with a predetermined policy appears less promising than an initialization using a precomputing of the Q-function through the traditional value-iteration algorithms in a model-based learning approach [11]. Amoui *et al.* have also applied RL successfully to the management of quality attributes in a news web application to optimize the application throughput [15]. An interesting point of this work is the use of simulation to initialize the learning functions.

Bahati and Bauer [16] propose to use RL to manage threshold-based rules. A first controller applies these rules to a target application in order to enforce its quality attributes. A second controller monitors these rules, adapts its thresholds to changing conditions and disables irrelevant rules. This approach is interesting since it limits the state space to appropriate state-action pairs and allows the reuse of learned models

 ISBN: 978-1-61208-134-2

Fig. 3. Comparison between the decisions made during the learning process as observations pass given that convergence speedups are applied or not. On the left, plots show decisions and the resulting performance when no speedup is made, while on the right convergence speedups are applied every 5,000 observations.

from one rule set to the next. Nonetheless, this approach reintroduces the burden of setting up efficient thresholds, does not qualify learned models nor give much information about the time it takes to achieve convergence.

Zhang *et al.* propose a pragmatic approach to resource allocation which consists in preallocating enough resources to match up to 95% of the observed workload and then allocates more resources on another cloud when this threshold is passed [17]. No real automatic control approach is applied to prevent instability, however.

Kalyvianaki *et al.* design controllers built on top of statistical and Kalman filtering to track CPU utilization and allocate pieces of processing units to VMs [18]. Their agents scale individual VMs as well as multi-tier applications while coping with workload changes. However, authors limit themselves to vertical scaling within the bounds of three physical servers; their controllers only work on applications with immediate rewards and do not care about stability.

VI. Conclusion and Future Works

Reinforcement learning is a promising approach towards an autonomic solution to the problem of dynamically adapting the amount of resources allocated to applications in cloud environments. However, reinforcement learning algorithms require care and expertise to deal with the main requirements of self-adapting cloud infrastructures: good allocation policies from the start, prompt convergence to the optimal policy and capability to deal with evolution in the performance model of applications. In this paper, we have evaluated experimentally, through simulation, the possibilities of two techniques:

- Careful initialization of the learning functions in order to have a good policy from the start;
- Convergence speedups for model-based reinforcement learning which inserts complete policy evaluation steps at regular intervals into the learning phases.

We have simulated Olio, a standard testbed web application, on a cloud and applied to this simulation a reinforcement learning approach to resource allocation. The main conclusions from these experiments are:

- Good initialization made by evaluation standard policies proved successful to begin the learning and the decisions with an already good policy and therefore good performance from the start;
- Convergence speedup techniques did improve a bit the error in the Q-function of the learning process, but more importantly succeeded in making the learned policy approach the optimal one faster.

We are currently conducting experiments to validate the use of statistical testing over the behavior model learned to complete the more traditional use of continuous learning, to take care of large changes in the performance model of the application and the cloud infrastructure by reinitializing the learning process. We are implementing on a cloud prototyped at Orange Labs an automated workflow to put these techniques at work for the allocation of virtual machines to applications and therefore to provide with a truly autonomic solution to this problem on actual industrial-strength clouds.

Much work and experimentation still need to be done. Our experimentations to date show that, even with our proposed workflow, applying reinforcement learning on a per-application basis will still need more information from its environment to be usable in the real world. Reinforcement learning should rather be applied in the context of a larger-scale workflow, where clouds could gain information from applications to applications in order to make the techniques much more successful. We plan to experiment now with higher level descriptions of applications and their need for adaptation in order to select from past applications learned policies from which the learning can be initialized more accurately.

References

[1] X. Dutreilh, N. Rivierre, A. Moreau, J. Malenfant, and I. Truck, "From Data Center Resource Allocation to Control Theory and Back," in *Proc. of the 3rd IEEE Int. Conf. on Cloud Computing, CLOUD 2010, application and industry track*. IEEE, 2010, pp. 410–417.

[2] R. Sutton and A. Barto, *Reinforcement learning — an introduction*. MIT Press, 1998.

[3] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation," in *Proc. of the 2006 IEEE Int. Conf. on Autonomic Computing (ICAC)*. IEEE Computer Society, 2006, pp. 65–73.

[4] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "Vconf: a reinforcement learning approach to virtual machines auto-configuration," in *Proc. of the 6th Int. conf. on Autonomic computing (ICAC)*, 2009, pp. 137–146.

[5] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Tech. Rep., July 2009. [Online]. Available: http://www.csrc.nist.gov/groups/SNS/cloud-computing/

[6] "Amazon EC2," http://aws.amazon.com/ec2/.

[7] J. A. Rolia, L. Cherkasova, and C. McCarthy, "Configuring workload manager control parameters for resource pools," in *NOMS*, 2006, pp. 127–137.

[8] D. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd ed. Athena Scientific, 1995, volume 1 and 2.

[9] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, O. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," 2008.

[10] D. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[11] M. L. Littman, "Algorithms for Sequential Decision Making," Ph.D. dissertation, Dep. of Computer Science, Brown U., mars 1996.

[12] D. Sheskin, *Handbook of parametric and non-parametric statistical procedures*. Chapman and Hall, 2007.

[13] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proc. of the 7th Int. Conf. on Autonomic computing (ICAC)*. ACM, 2010, pp. 1–10.

[14] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "On the Use of Fuzzy Modeling in Virtualized Data Center Management," in *Proc. of the 4th Int. Conf. on Autonomic Computing (ICAC)*. IEEE Computer Society, 2007, pp. 25–34.

[15] M. Amoui, M. Salehie, S. Mirarab, and L. Tahvildari, "Adaptive Action Selection in Autonomic Software Using Reinforcement Learning," in *Proc. of the 4th Int. Conf. on Autonomic and Autonomous Systems (ICAS)*. IEEE Computer Society, 2008, pp. 175–181.

[16] R. M. Bahati and M. A. Bauer, "Towards adaptive policy-based management," in *NOMS*. IEEE, 2010, pp. 511–518.

[17] H. Zhang, G. Jiang, K. Yoshihira, H. Chen, and A. Saxena, "Resilient workload manager: taming bursty workload of scaling internet applications," in *Proc. of the 6th Int. Conf. industry session on Autonomic computing and communications*. ACM, 2009, pp. 19–28.

[18] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *Proc. of the 6th Int. Conf. on Autonomic computing (ICAC)*. ACM, 2009, pp. 117–126.

 ISBN: 978-1-61208-134-2