VScaler:Autonomic Virtual Machine Scaling

Lenar Yazdanov
Faculty of Computer Science
TU-Dresden
Dresden, Germany
lenar@se.inf.tu-dresden.de

Christof Fetzer
Faculty of Computer Science
TU-Dresden
Dresden, Germany
christof.fetzer@inf.tu-dresden.de

***Abstract*—Recent research results in cloud community found that cloud users increasingly force providers to shift from fixed bundle instance types(e.g. Amazon instances) to flexible bundles and shrinked billing cycles. This means that cloud applications can dynamically provision the used amount of resources in a more fine-grained fashion. This observation calls for approaches which are able to automatically implement fine granular VM resource allocation with respect to user-provided SLAs. In this work we propose VScaler, a framework which implements autonomic resource allocation using a novel approach to reinforcement learning.**

***Index Terms*—scalability; performance; measurement;**

I. Introduction

Recent observations[1] of IaaS trends state, that model of fixed bundles, so called ”instance types” will eventually change to flexible bundles. This change is mostly economically driven. The reason is that cloud user does not want to rent 6 CPU cores when, it is required only 5 of them. Moreover, model of ’fixed bundles’ forces cloud users to provision applications with time varying resource demand for peak load. This strategy leads to high resource under-utilization. Users have to pay for resources which are not used. Another observation is a size of cloud billing cycles. Nowadays, most of cloud providers have 1 hour billing period. Hence, user has to pay for the whole hour, even if VM was running only for 10 minutes. Therefore, authors of [1] conclude that IaaS providers would also shrink billing periods. Third observation is that cloud users can be differentiated, Some of them require high QoS, others have soft QoS requirements.

Data centers can adapt to workload changes by scaling an application horizontally. Horizontal scaling means adding or removing worker nodes(VMs) to and from application. Virtulization enables another way to allocate resources to VMs. The process of adding/removing certain resource from VM is called vertical scaling. Most of modern hypervisors support on-the-fly VM resizing, which means that certain resource, such as CPU, memory or I/O bandwidth, can be added/removed to/from a VM without shutting it down. Cloud trends described in [1] create space for possible resource usage optimization. Vertical scaling fits better to this task for the following reasons. Vertical scaling has lower reconfiguration latency in comparison to horizontal scaling. It does not introduce license costs. Recent work [2] about cloud computing research states that rapid increase in RAM available in a single machine together with large number of CPU cores per machine makes it economical to run application locally rather than in distributed manner. The lack of autonomic scaling approaches based on vertical scaling is calling for solutions which are able to implement fine-granular resource allocation.

Last years industry and academia paid a lot of attention to automatic resource provisioning. There are different approaches in this direction. One of the popular one’s is a threshold based scaling, where application scaling is implemented when resource utilization reaches certain threshold. It mostly exploited by public cloud providers such as RightScale, Amazon. This method tends to focus on scaling at the machine or VM level. But threshold based technique does not facilitate the definition of higher business function and user provided SLA, which leads to resource over-provisioning. Moreover, it is a duty of a cloud user to define these thresholds, which is not trivial and requires expertise knowledge.

More recently, academia made an effort to overcome this problems and proposed automatic scaling techniques based on reinforcement learning (RL) approach. The benefit of this method is ability to make decisions under uncertainty based on environmental observations. The behaviour of applications running in a cloud can be affected due to modifications or change in the workload request model. The threshold based approach in such situations would require a model change, while RL approach will adapt to suit the environment on the basis of its own experience. But main drawback of RL approaches is dimensionality problem, with each new variable added the size of the state space grows exponentially leading to exponential increase in learning time.

In this paper we propose VScaler framework. As a core of VScaler we use reinforcement learning. We overcome one of problems of this approach by applying parallel learning with assumption which allows to speed up the learning process. Moreover, our framework does not require offline learning, it quickly adapts resource allocation policy.

II. Motivation and RL problem

Many web applications running in a cloud environment such as social networks, online-shops, auctions have a time varying resource demand. One of the ways to provision them is to predict workload and allocate additional resources in advance. But for unplanned events (e.g. due to unexpected

rise of popularity) we need auto-scaling technique to automatically adjust resources allocated to applications based on their requirements at any moment of time. Another challenge in scaling web applications is their ability to change workload type. For example, workload may change from being CPU intensive to memory intensive. Threshold based technique can adapt to this kind of variations, but it requires changing of scaling model used so far. Therefore it is necessary to have self-adaptable scaling approach which is able to implement online model adaptation.

Well known 1 hour billing cycles and fixed instance types lead to resource wastage no matter how good scaling policy is. However, the situation is changing. There are IaaS providers such as *CloudSigma* [3], which already offers their infrastructure with 5 minute billing cycle. Moreover, user is free to define proper VM type by selecting required amount of RAM, CPU or I/O.

In order to adapt to workload variations application can be scaled horizontally or vertically. Vertical scaling in comparison to horizontal allows to allocate resource with lower overhead in terms of time and cost. In [4] authors compared this types of scaling. Horizontal scaling requires up to 5 minutes, while vertical scaling requires less than 0.5 second to reconfigure a VM. Moreover, vertical scaling does not introduce license cost which is present in horizontal scaling. Additionally in case a slight workload increase horizontal scaling leads to high resource wastage. Another advantage of vertical scaling was observed in [5]. Authors state that vertically scaled application has better response time in comparison to horizontally scaled one. Furthermore, authors in [2] pointed out that amount of memory available in single machine increases fast and with combination of large number of CPU cores per machine makes more economically efficient to run an application locally instead running it in a distributed fashion.

There are already approaches in the direction of vertical scaling. Solution proposed by authors of [4] is based on resource usage history, which allows to create optimal scaling model. But the history is not always available. For example, if an application is deployed for first time, then cloud provider has no knowledge about behaviour of the application. Last years one of machine learning approaches became the focus of attention from researches working in the domain of cloud scaling. This technique is called reinforcement learning, it is able to perform self-adaptation of the model. RL has been successfully applied in the domain of resource allocation [6], [7], [8], [8], [9]. It is based on knowledge-free trial-and-error methodology, where agent gets punishment and reward for each observation. The general policy is to move towards state which gives better utility.

Our motivation in this paper is to develop a self-adaptable system based on reinforcement learning approach.

III. RL PROBLEM

RL problems can modeled as a Markov Decision Process (MDP). A MDP can typically be represented as a four tuple consisting of states, actions, transitions probabilities and rewards.

- S, represents environmental state space
- A, represents total action space
- $p(. \mid s, a)$, defines a probability distribution governing state transitions $s_{t+1} \sim p(. \mid s_t, a_t)$
- $q(. \mid s, a)$, defines a probability distribution governing the rewards received $R(s_t, a_t) \sim q(. \mid s_t, a_t)$

S, the set of all possible states represents the agents observable world. At each, step t, the agent perceives its current state $s_t \in S$ and the available action set $A(s_t)$. By taking an actions $a_t \in A(s_t)$, the agent moves to the next state s_{t+1} and receives an immediate reward r_{t+1} from the environment.

In cases when environmental model is not known model-free reinforcement learning algorithms are used. Q-learning belongs to the collection of these algorithms. The update rule in Q-learning of taking action a in state s defined as:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_a Q(s_{t+1}, a_{t+1}) - Q(s,a)] \tag{1}$$

where α is learning rate and γ is a discount factor. The interactions in RL consist of exploitations and explorations. Exploitation is to follow the policy obtained so far. The policy is to take an action which has highest Q-value. Exploration is the selection of random actions to capture the change of environment to prevent from maximizing the short-term reward. An agent is designed by following ε-greedy policy. With a small probability, the agent picks up a random action, and follows the best policy it has found for the rest of the time.

To facilitate agent learning for a cloud resource allocation problem, one must define an appropriate state-action space formalism. In cloud resource management state s corresponds to the amount of resources allocated. In horizontal scaling solutions state is represented as number of running VMs [8], [9] and action a is amount of VMs which can be added, removed or maintained. In vertical scaling state usually is defined as a number of CPUs and memory size assigned to a VM [6], [7]. Action a is considered as management operation. For example, action can be shown as *(increase, decrease)*, where each resource is changed by predefined step size. The action indicates an increase of a number of CPUs and decrease of memory size. Resource requirements of applications with highly fluctuated workload may change greatly. Therefore it is necessary to have certain level of over-provisioning, which is achieved by changing number of CPUs and large memory sizes allocated to a VM. In our work we would like to decrease a level of resource over-provisioning by having smaller resource step sizes, such CPU and memory.

RL approaches generally suffer from so called curse of dimensionality problems, as the size of the state space grows exponentially with each new state variable added. In our approach we want to apply smaller resource step size which eventually increases state-action space and as consequence increases a model learning time. Therefore previous works[6], [7] consider only scaling by number of CPUs and large

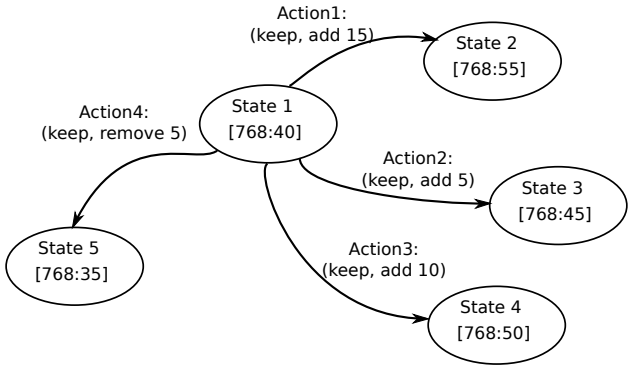

Fig. 1. Markov Decision Process with 5 states and 4 actions

memory sizes. In this work we introduce parallel learning with assumption with allows to overcome this problem.

IV. Parallel learning with assumption

Agent in RL approach learns an environment by visiting each state-action pair in a given environment. Therefore the learning process time is mainly depends on the size of state-action space. For example, if we have a VM with two configurable parameters, assuming 10 different settings for each parameter and 10 actions for each state. Then minimum number of interactions required to observe reward for all state-action pairs will be $2^{10} * 10 = 10240$

One of the approaches to speed up agent's learning process of approximated model is to learn in parallel[10]. It means that agent does not have to visit every state-action pair in a given environment. Instead, it could learn values of state-action pairs it did not visit by taking them from neighbouring agents. Each agent learns by working on individual task and periodically shares own observations with other agents. This approach allows greatly reduce environment approximation time. Parallel learning in RL has been already applied to cloud resource management. In previous work [8] this technique is used to implement autonomic horizontal scaling. The authors defined the environment as a number of VMs divided in a groups. Each group is a observable environment for one agent. Agents periodically share their observations with each other. The work results show that this technique allows significantly speed up the learning process. But solution presented in this paper can not be simply transferred to vertical scaling. Because it would require to run more than one VM to parallelize learning process, which will increase a cost.

Our solution is based on the idea that there is more to learn from the observation of one state-action interaction, than just Q-value. Every time when action is taken we can observe amount of resources required for this particular workload. Especially when capacity provisioned to a VM is higher than average utilization over period when action was taken. Hence we can have an assumption that VM would be able to serve observed workload while satisfying service level agreement(SLA), if it would have a capacity which is higher than average utilization. Consider the following simple example presented on figure 1. Each state represents a VM resource allocation. First value is memory size in MB and second CPU capacity expressed in Xen Credit Scheduler[11] cap value, which fixes the maximum amount of CPU the domain will be able to consume. The cap is expressed in percentage of one physical CPU. In *state 1* the VM was allocated 768 MB of memory and 40% of physical CPU. Then *action 1* was taken and the VM got additionally 15% of physical CPU, which means that the VM was reconfigured to 768 MB RAM and 55% of physical CPU. After fixed time interval agent observes application feedback and resources utilization over this period. Memory usage is 83%, virtual CPU usage is 78% and the application performance satisfies SLA. We could assume, that ideal resource allocation scheme is 637.44 MB of memory and 42.9% of physical CPU. Therefore we can update all state-action pairs which move the VM resource allocation toward the states where it has enough capacity to serve observed workload. In the example we will update *(state1, action3)* and *(state1, action2)*, because in *state 3* and *state 4* the VM allocation is higher than average utilization. Following approach allows to speed up the learning process, because after each agent's action more than one state-action pair is updated.

V. VScaler design

In this section we provide an overview of VScaler's architecture and working flow. We describe MDP state definition, each phase of RL and provide description of the algorithm used by VScaler.

A. Overview

We implemented VScaler to automatically control resource allocation to an application running in a virtualized environment without having any prior knowledge about an application behaviour.

Figure 2 shows overall architecture of VScaler system and interactions between components. VScaler uses a *proxy* monitoring capabilities to get incoming request rate and an application performance feedback. *The host daemon* collects the VM resource usage statistics and implements host's resources allocation to the VM. *The predictor* inside VScaler tracks incoming request rate and predicts workload for the next reconfiguration interval. In order to implement automatic VM capacity management VScaler makes decisions based on the VM resource consumption, the application performance feedback and predicted workload.

The management process runs in a following way. VScaler submits resource allocation scheme to *the host daemon*. *The host daemon* allocates resources to the VM. After fixed interval VScaler requests resource usage statistics from *the host daemon* and the application performance feedback from *the proxy*. This data is used to calculate reward and update capacity management policy. Then VScaler takes workload prediction and calculates the best resource allocation scheme for the next reconfiguration interval.

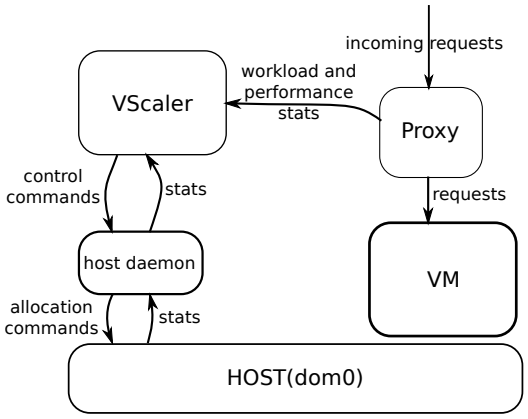

Fig. 2. Architecture of VScaler

B. Design solutions

1) state description: We define MDP which models our approach to VM resource allocation problem as $S = \langle m, c, w, g \rangle$, where:

- $m \in \mathbb{N}$ is memory in MB allocated to the VM;
- $c \in \mathbb{N}$ is CPU allocated to the VM, expressed in a Xen Credit Scheduler[11] cap value);
- $w \in \mathbb{N}$ is a total number of user requests observed per time period and which was served within SLA. This value changes between time steps.
- $g \in \mathbb{N}$ is guess about total number of user requests which can be served in this state while satisfying SLA. This value is updated using *update alternatives* algorithm.

The agent's action space consists of all allowed actions within current state. The agent can choose to add, remove or keep the CPU and memory allocation. For each resource we assign an action set, which is $A = \{a \in \mathbb{Z}, A_{min} < a < A_{max}\}$. Actions are discretized by setting step a_{st} on each resource. In our experiments memory allocation is bounded between $A_{min} = 512MB$ and $A_{max} = 1536MB$, for CPU we have $A_{min} = 10$ and $A_{max} = 50$.

VScaler uses workload predictor which takes request history as input and applies AR model to predict expected number of requests for the next reconfiguration interval. We do not need to define all possible request numbers for each state as it is implemented in [8], [9]. Hence the last to variables(w and g) in the state definition are not fixed within a state. Therefore state-action space size in VScaler depends only on allocation step size for CPU and memory. This design solution allows to reduce agent's environment size and Q-learning lookup table size.

2) reward calculation: We use application performance feedback and VM resources usage statistics to calculate reward. Reward function is defined as ratio between $perfFeedback$ and $resUtil$ which allows to guide agent towards state which gives higher utility:

$$reward = \frac{perfFeedback}{resUtil} \tag{2}$$

$$perfFeedback = \begin{cases} 1, & \text{if respTime<SLA} \\ e^{-p|\frac{respTime-SLA}{SLA}|} - 1, & \text{otherwise} \end{cases} \tag{3}$$

$$resUtil = \frac{\sum_{i=1}^{n}(1 - U_i)}{n} \tag{4}$$

$resUtil$ is a resource usage efficiency, where U_i is a utilization status of certain resource. We consider tho resource types CPU and memory. With increase of resource usage $resUtil$ value decreases, this allows to encourage agent to take actions which give higher resource utilization. We also include SLA penalty in reward calculation, in order to avoid situations when agent moves to the states with lowest over-provisioning. Reward becomes negative, whenever SLA is violated. Therefore an agent gets punished for any resource allocation scheme which violates SLA.

C. Initializing Q learning

Q-learning is a model-free RL technique, where agent learns an environment online. In order to apply control operations during the learning process one has to follow some policy from which decision will be chosen and resource management operations will be taken on controlled system. Defining such policy is complicated, because it requires some knowledge about an application in virtualized environment. In cloud computing context such information may not be available, when an application is deployed for the first time. Therefore only standard policy can be applied. In our approach we use greedy policy, which assumes, that for the next reconfiguration interval an application resource consumption will double. Hence, we allocate two times more resources than current resource utilization. This leads to over-provisioning, but having under utilization during this phase allows to update alternative state-action pairs using parallel learning with assumption. VScaler starts to exploit obtained policy, as soon as predictor is ready to forecast workload. In VScaler we use 100 samples of recent observed user requests number to predict the workload for the next reconfiguration interval.

D. Model learning and exploitation

Our parallel learning approach with assumption considers two types of agents: one real and number of alternative agents. Actions taken by real agent are used to implement resource allocation operations and affect system capacity, while alternative agents observations are used to update management policy for state-action pairs which satisfy resource usage requirements. In the example presented on figure 1 two alternative agents will be launched after reconfiguration. They start from *state 1* and move toward *state 3* and *state 4* respectively. Alternative agent is launched from real agent's previous state every time, when action taken by real agent is not optimal. Termination of alternative agent is called in two cases. First, if there is no action which connects alternative agent's current state with state where the VM has enough capacity to serve observed workload. Second, if real agent took an action which

```
repeat
    s_t ← getCurrentState()
    a_t ← chooseNextAction(s_t, Q)
    U_{t+1} ← getResourceUsage()
    r_{t+1} ← calculateReward(U_{t+1})
    w_{t+1} ← getObservedRequests()
    updateModel(s_t, a_t, r_{t+1}, w_{t+1} Q)
    if respTime < SLA then
        updateAlternatives(s_t, a_t, w_{t+1}, Q, U_{t+1})
    else
        terminateAllAlternatives()
    end if
    updateRequests(s_{t+1}, w_{t+1})
    t ← t + 1
until Agent is terminated
```

Fig. 3. Agent learning algorithm

cased resource under-provisioning and SLA violation. In this case all alternative agents are terminated, because we have only lower resource utilization bound. We don't know what was the best resource allocation scheme.

The agent learning algorithm presented on figure 3. Each reconfiguration interval real agent records previous state, chooses next action and observes reward for the taken reconfiguration action. The next action is selected by algorithm presented on figure 5. Real agent uses predicted workload value to select states which have guessed requests number g higher than predicted value. Then it takes an action which connects current real agent's state with one of these states and has highest Q-value. If after reconfiguration SLA was not violated, then the algorithm updates alternative agents states, otherwise it terminates all alternative agents. At the end of each iteration observed requests number w for the state s_{t+1} is overwritten by w_{t+1}, if $w < w_{t+1}$. If SLA was violated then g is overwritten by w, because guess was wrong and amount of resources allocated in state s_{t+1} was not enough to serve observer requests number without violating SLA.

Pseudo-code of updating alternative agents algorithm is presented on figure 4. First, the algorithm looks for already running alternative agents which can move to a state where the VM is assigned more capacity than observed resource utilization. Otherwise, alternative agent is terminated. Second, the algorithm launches new alternative agents from real agent's state s_t. Third, the algorithm updates all running alternative agents by calculating reward for taken action, updates guessed requests number g and updates the model.

E. Environment exploration

It is known that RL agents cooperate with managed environment by applying two types of interaction: exploration and exploitation. Exploitation is to follow optimal policy, while exploration is the selection random actions to capture system dynamics and refine the existing policy. Basic RL algorithm uses ε - greedy policy to select an action, where agent makes random selection with a probability ε. There are

```
for each alter in alternatives do
    s_alter ← getCurrentState(alter)
    if AlternativeCanMove(s_alter, U_{t+1}) then
        MoveAlternativeTo(alter, s_next)
    else
        terminateAlternative(alter)
    end if
end for
for each state s_next connected to s_t do
    if getResourceAllocation(s_next)> U_t then
        alternatives.append(s_t, s_next)
    end if
end for
for each alter in alternatives do
    s_alter ← getCurrentState(alter)
    a_alter ← getLastAction(alter)
    r_{t+1} ← calculateReward(U_{t+1})
    updateRequests(s_alter, w_{t+1})
    updateModel(s_alter, a_alter, r_{t+1}, w_{t+1}, Q)
end for
```

Fig. 4. Update alternatives

```
predValue ← predictWorkload()
for each state s_next connected to s_t do
    g ←getRequests(s_next)
    if g > predValue then
        selectedStates.append(s_next)
    end if
end for
return getBestAction(s_t, selectedStates)
```

Fig. 5. Choose next action

different schemes for an action selection during the exploration phase. In [12] authors state that selecting untried actions requires $O(n^2)$ interactions to observe all state-action pairs, while random selection policy requires $O(2^n)$ interactions. Applying RL in cloud computing context creates additional requirement to the exploration process. One has to ensure that exploration action does not violate SLA. Hence during exploration phase VScaler selects actions which allocate enough resources to serve predicted workload and then among these actions chooses the one which was executed less frequently. In all our experiments VScaler implements exploration with a probability $\varepsilon = 0.05$

VI. Evaluation

The testbed for our experiments is hosted on quad-core Xeon 2.66GHz with 16 GB memory, 100 Mbps network and Ubuntu 12.04 with Xen 4.1. We use RUBiS[13] benchmark to evaluate VScaler. RUBiS is online auction benchmark. In our experiments PHP version of the benchmark is applied, which consists of web front-end and database back-end. We run Apache 2.2 as a web server and MySQL as a database server. The web-server and the database run inside VMs with

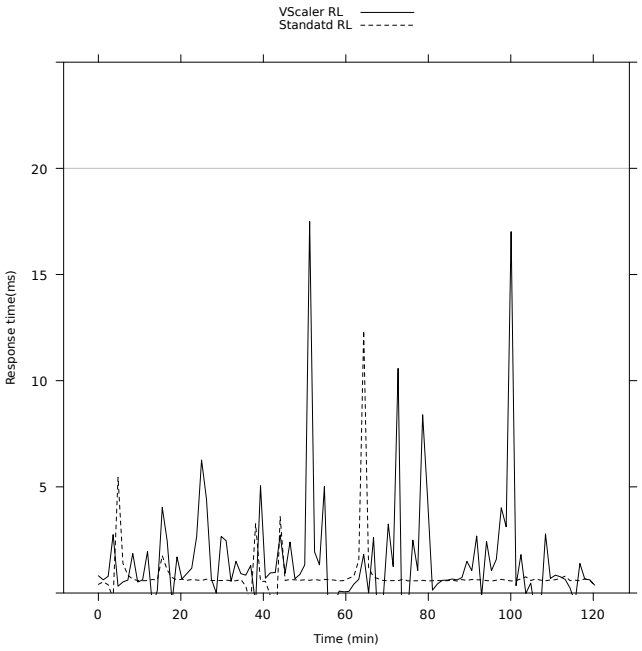

Fig. 6. Average response time: Standard RL vs VScaler RL

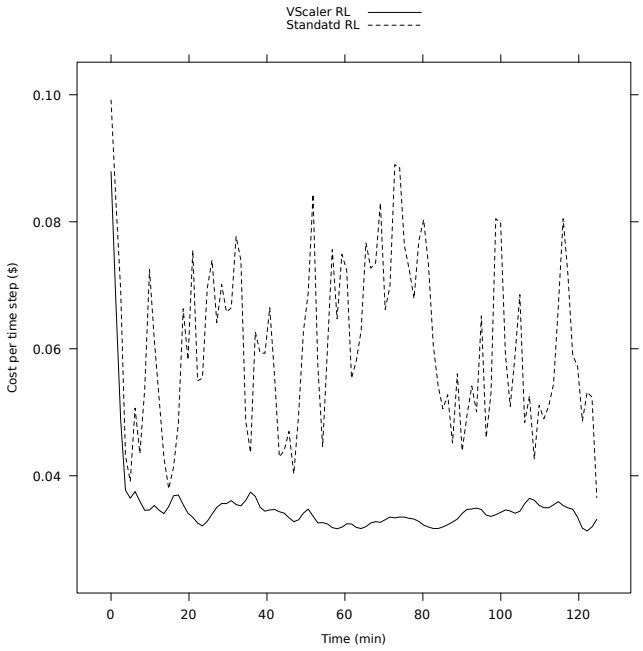

Fig. 7. Average costs: Standard RL vs VScaler RL. The greater size of VM in terms of CPU and memory, the greater the cost

64-bit CentOS 6.3. Throughout all experiments database VM is over-provisioned.

A. Convergence speedup

To analyse the advantage of the approach presented in this paper, we evaluated performance of two Q-learning approaches. The first approach uses parallel learning with assumption. Second approach is standard Q-learning approach, where agent after each observation updates only one state-action pair such as in [9]. To make clear comparison we used the same state-action formalism in parallel learning with assumption as for standard Q-learning approach. Therefore we did not use prediction, instead each state was assigned fixed workload value. We also did not use special initialization policy. In each approach an agent learned environment using standard policy, which assumes that in exploration phase the agent takes random action, in exploitation phase the agent takes an action which gives higher utility. In this experiment we run constant workload. 1000 clients were emulated using RUBiS benchmark.

On figure 6 presented average response achieved by described approaches. VScaler RL quickly adapts to the workload, while Standard RL needs more time to learn the environment and adapt to the workload. One has to notice that both approaches do not violate SLA, but Standard RL in comparison to VScaler RL achieves this performance for higher cost. It can be observed on graph 7. We define cost of the VM as directly proportional to its configuration.

B. Real world scenario

To evaluate VScaler performance in real cloud environment with dynamic resource demands variations we instrumented RUBiS client emulator to modulate request rate of RUBiS benchmark. The RUBiS client emulator reads clients request rate from trace file. The trace consists of per-minute workload intensity observed during WorldCup 98[14]. We used 6 hour trace starting at 1998-05-10:03.00.

The goal of this experiment is to measure performance and resource utilization for 5 different schemes:

- static allocation
- dynamic allocation
  - fixed step allocation
    - CPU step 2, memory step 64 MB
    - CPU step 5, memory step 64 MB
  - dynamic step allocation
    - scale up 100%, scale down 100%
    - scale up 100%, scale down 20%

The static allocation is a VM template which capacity is not changed during the test. We configured the VM template with 1536 MB of memory and 1 virtual CPU with cap value of 50. The VM template approximates Amazon EC2 m1.small instance. The rest schemes change a VM capacity during the test. These schemes have maximum resource allocation bound which is the VM template size. Scheme with fixed step allocation is allowed to change CPU and memory only by one step. Dynamic step allocation scheme changes current VM capacity by scale up or scale down factor. We consider dynamic allocation scheme, because it is not trivial task to find right step size for resource allocation, when an application resource demand changes dynamically. Too big step size leads to over-provisioning, while small step causes resource saturation and as result SLA violation. In our experiment we

set SLA to 20 ms.

On figure 8 presented average resource utilisation. Static allocation has lowest CPU and memory utilization. This result demonstrates how much resources are wasted when web application runs in fixed instance types. The rest schemes have significantly higher CPU and memory utilisation. Schemes with step scaling have a bit higher utilization in comparison to dynamic step scaling schemes. The reason is that dynamic step allocation schemes have more freedom in scaling. It is interesting to notice that dynamic step scaling with smaller scaling down factor has lower utilization. This scheme is more pessimistic for releasing resources. In our experiments we fixed lower memory allocation size to 512 MB due to OS minimal requirements, therefore memory utilisation is lower than CPU utilization.

Figure 10 shows 99 percentile response time. Two schemes do not fit in SLA requirements, which is 20 ms. As we can see fixed step scaling with smaller CPU allocation size violated SLA, while scheme with bigger CPU allocation step satisfied SLA. The result shows that it is difficult to find 'right' step size when there is no knowledge about application running inside the VM. Another interesting observation is that one of dynamic step allocation schemes also violated SLA. Scheme with 100% scaling down implements aggressive capacity management, therefore during unexpected spikes it does not allocate enough resources.

We presented achieved throughput on figure 9. The throughput achieved by dynamic step allocation scheme with 100% scaling up and 20% scale down factor is close to fixed step allocation scheme.

Experiment results show fixed step allocation allows to achieve good performance, but only if allocation step has a proper size, but it requires some knowledge about application behaviour. Scheme with dynamic step allocation achieves same result and does not require knowledge about application.

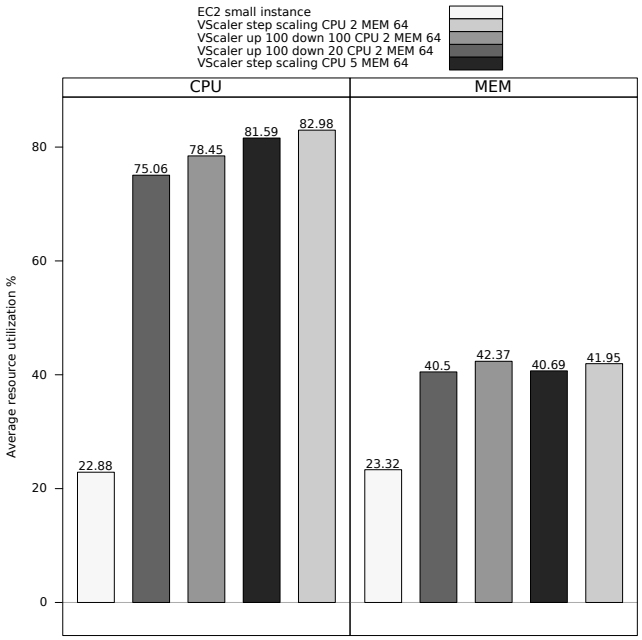

Fig. 8. Average resource utilization

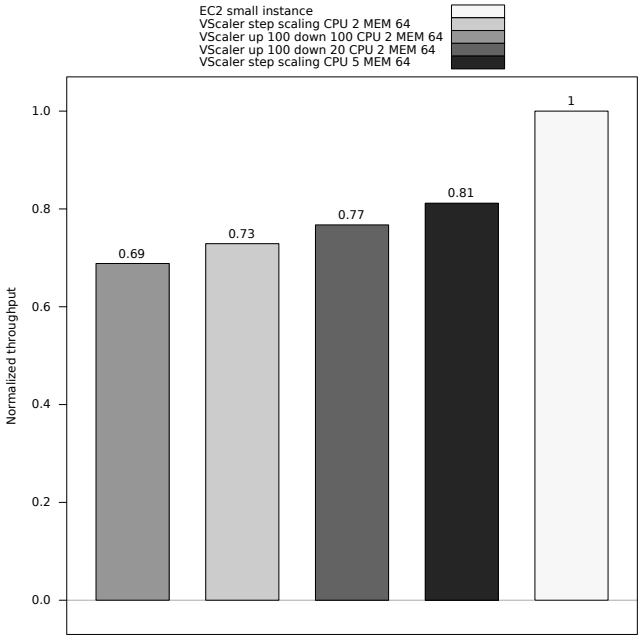

Fig. 9. Normalized throughput

VII. Related work

Cloud computing industry (e.g. Amazon[15], RightScale[16]) use a predefined policies(rules) to manage scaling of virtualized applications. Usually users have to specify scaling rules after deployment of application. These rules define upper and lower limit of servers and scaling conditions which usually defined as resource utilisation or request rate. This approach assumes that cloud users have expertise knowledge about application behaviour, so they can define proper rules. However, it is not always true.

Several works apply auto-scaling techniques to VM capacity management. We consider works which apply vertical scaling in resource allocation problem.

SmartScale[4] uses a combination of horizontal and vertical scaling. In each reconfiguration phase authors propose to consider trade-off between these two types of scaling. SmartScale chooses the one which allows better utilize resources and fit SLA requirements. The framework implements reconfiguration with 1 hour period. The chosen period fits to billing cycles of most of cloud providers. But SmartScale would require model adaptation in order to achieve high resource usage efficiency in new conditions, when IaaS providers shrink billing periods.

Authors in [7] use RL in VM resource management. They addressed the problem of applying vertical scaling to multi-tier web applications capacity management. System presented in this paper uses fixed step scaling which leads to SLA violation when workload change requires higher scaling step. In our work we show that better results can be achieved with

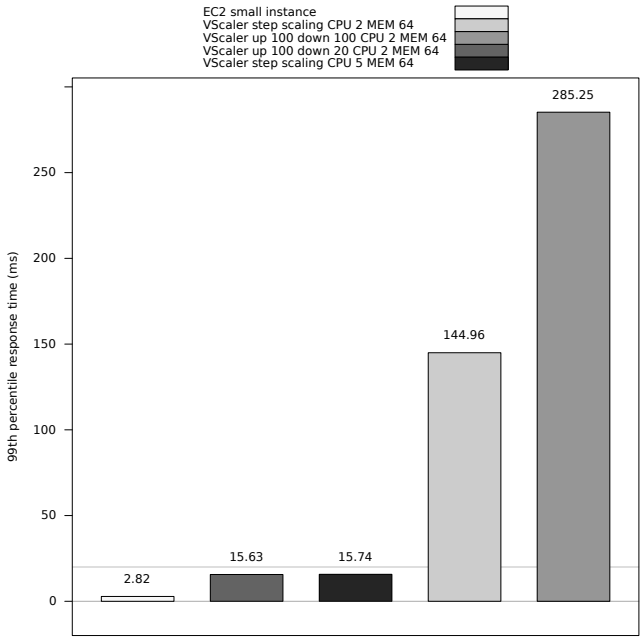

Fig. 10. 99th percentile response time

dynamic step allocation. Moreover, proposed system requires longer learning time in comparison to our approach.

As in [4] vertical scaling is implemented by plugging virtual CPU. Virtualisation provides mechanism to allocate fraction of physical CPU. It is known that RL has scalability problems. Therefore shifting from full CPU to fraction CPU allocation would increase number of state-action pairs in RL approach, which will dramatically increase performance of proposed approach.

The work [17] presents CloudScale system. CloudScale implements fine granular CPU and memory allocation. The system uses two complimentary approaches patter-driven and state-driven demand prediction which allow to keep application performance within SLA. But CloudScale uses predefined thresholds which require a particular knowledge about virtualized application. Moreover, the system needs to adapt scaling thresholds when workload changes from CPU intensive to memory intensive.

VIII. Conclusion

Reinforcement learning is promising approach in the direction of autonomic capacity management. It allows adapt to dynamic changes of resource requirements from applications hosted on a cloud. However it is requires significant learning time when state-action space is big. In this work we presented novel approach to RL which speed ups the learning process. The results show that parallel learning with assumption observes environment faster than standard learning policy. Moreover, proposed approach allows to keep web application performance to ensure SLA requirements. We evaluated VScaler against real world scenario. We observe that VScaler achieves high resource usage efficiency and performance with dynamic allocation scheme.

In this work we implement single VM scaling which can be applied to a number of applications running in one VM. It was mentioned that RL has a scalability issues, having more VMs under control would increase state-action space and model update time. We need a hybrid approach to achieve good scalability. This is a part of our future work.

Acknowledgment

This work is supported by Erasmus Mundus Action 2 Programme of the European Union.

References

[1] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "The resource-as-a-service (raas) cloud," in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, ser. HotCloud'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 12–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342763.2342775

[2] A. Rowstron, D. Narayanan, A. Donnelly, G. O'Shea, and A. Douglas, "Nobody ever got fired for using hadoop on a cluster," in *Proceedings of the 1st International Workshop on Hot Topics in Cloud Data Processing*, ser. HotCDP '12. New York, NY, USA: ACM, 2012, pp. 2:1–2:5. [Online]. Available: http://doi.acm.org/10.1145/2169090.2169092

[3] Cloudsigma(iaas) provider. http://www.cloudsigma.com/.

[4] S. Dutta, S. Gera, A. Verma, and B. Viswanathan, "Smartscale: Automatic application scaling in enterprise clouds," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, june 2012, pp. 221 –228.

[5] C. M. Wesam Dawoud, Ibrahim Takouna, "Elastic virtual machine for fine-grained cloud resource provisioning," in *Proceedings of the 4th International Conference on Recent Trends of Computing, Communication and Information Technologies (ObCom 2011)*, ser. CCIS, no. 0269. Tamil Nadu, India: Springer, 12 2011.

[6] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "Vconf: a reinforcement learning approach to virtual machines auto-configuration," in *Proceedings of the 6th international conference on Autonomic computing*, ser. ICAC '09. New York, NY, USA: ACM, 2009, pp. 137–146. [Online]. Available: http://doi.acm.org/10.1145/1555228.1555263

[7] J. Rao, X. Bu, C.-Z. Xu, and K. Wang, "A distributed self-learning approach for elastic provisioning of virtualized cloud resources," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, july 2011, pp. 45 –54.

[8] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, p. n/a, May 2012. [Online]. Available: http://dx.doi.org/10.1002/cpe.2864

[9] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow," in *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, Venice/Mestre, Italy, May 2011, pp. 67–74. [Online]. Available: http://www.thinkmind.org/download.php?articleid=icas_2011_4_10_20094

[10] R. M. Kretchmar, "Parallel reinforcement learning," in *In The 6th World Conference on Systemics, Cybernetics, and Informatics*, 2002.

[11] Credit-based cpu scheduler. http://wiki.xen.org.

[12] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[13] Rubis online auction system. http://rubis.ow2.org/.

[14] The ircache project. http://www.ircache.net/.

[15] Amazon auto scaling service. http://aws.amazon.com/autoscaling/.

[16] Rightscale web site. http://www.rightscale.com.

[17] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 5:1–5:14. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038921