Applying Reinforcement Learning Towards Automating Resource Allocation and Application Scalability in the Cloud

Enda Barrett, Enda Howley, Jim Duggan

School of Engineering & Informatics, National University of Ireland Galway, Ireland

SUMMARY

Public Infrastructure as a Service (IaaS) clouds such as Amazon, GoGrid and Rackspace deliver computational resources by means of virtualisation technologies. These technologies allow multiple independent virtual machines to reside in apparent isolation on the same physical host. Dynamically scaling applications running on IaaS clouds can lead to varied and unpredictable results due to the performance interference effects associated with co-located virtual machines. Determining appropriate scaling policies in a dynamic non-stationary environment is non-trivial. One principle advantage exhibited by IaaS clouds over their traditional hosting counterparts is the ability to scale resources on-demand. However a problem arises concerning resource allocation as to which resources should be added and removed when the underlying performance of the resource is in a constant state of flux. Decision theoretic frameworks such as Markov Decision Processes are particularly suited to decision making under uncertainty. By applying a temporal difference reinforcement learning algorithm known as Q-learning, optimal scaling policies can be determined. Additionally reinforcement learning techniques typically suffer from curse of dimensionality problems, where the state space grows exponentially with each additional state variable. To address this challenge we also present a novel parallel Q-learning approach aimed at reducing the time taken to determine optimal policies whilst learning online.

KEY WORDS: Reinforcement Learning, Cloud Computing, Resource Scaling

1. INTRODUCTION

IaaS clouds rely on economies of scale to deliver computational resources to consumers in a cost effective way. Sourcing computational resources from IaaS clouds eradicates the cost associated with maintaining the equivalent resources in-house. Similar to traditional utilities such as electricity and gas [5], consumers typically pay only for what they use, provisioning resources as needed in an on-demand fashion. This elasticity or ability to scale resources as required is one of the principle differences between computational clouds and previous utility computing forms such as computational grids and clusters, which require advanced reservations. In delivering resources to consumers, Infrastructure as a Service (IaaS) providers utilise virtualisation technologies such as Xen [4] and VmWare [31] to partition a single physical server into multiple independent Virtual Machines (VMs). These VMs reside in a co-located manner and have no visibility or control over the host environmental configuration or neighbouring VMs. Figure 1 demonstrates a typical cloud scenario where multiple VMs are co-located on a single physical host server. Depending on its configuration each VM is allocated a portion of the physical host resource i.e. CPU cycles, RAM, Disk and Network bandwidth. A Virtual Machine Monitor (VMM) is installed on the physical host and is responsible for controlling VM access to the host's resources. The VMM attempts to isolate individual VMs with respect to security, failure and their respective environment, but not in respect of performance [13, 22]. Consequently performance unpredictability has been identified as one of the key obstacles facing growth and greater adoption of cloud computing [2]. Much uncertainty exists in relation to how ported applications will perform, and indeed scale, once they are deployed in the cloud.

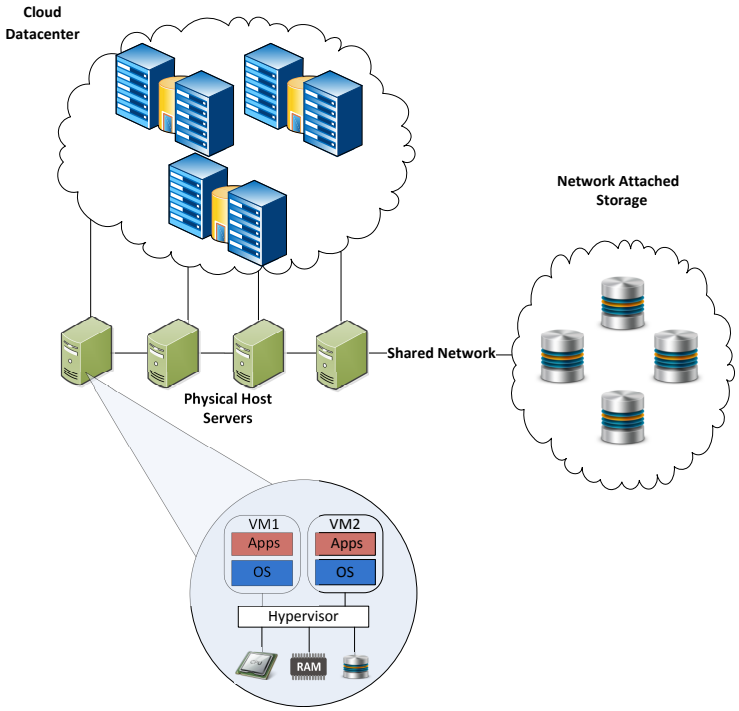


Figure 1. Physical host with three instantiated virtual machines.

Dynamically scaling applications on large IaaS clouds in response to workload or performance changes presents a key challenge for resource planning techniques and application management. An effective scaling solution should allocate resources to optimise factors such as cost, performance and reliability. The solution itself should also be scalable i.e. capable of dealing with large workloads and complex resource allocation decisions. The current approach favoured for allocating resources to applications based on fluctuating numbers of requests and application performance is to define threshold based policies [19, 20]. These are rule based approaches, where upper and lower bounds are defined based on an observable metric such as application response time. With this approach applications can scale up to meet demand at peak times and scale back once demand has subsided. Determining appropriate thresholds however requires expert domain and application knowledge and must often be redefined based on application updates or workload changes. Consider an enterprise application for a large multinational corporation. The application is accessed on a daily basis by a globally distributed workforce. It is powered by three separate data centres, situated in strategic geographical locations for optimal user support. Software such as enterprise applications by their nature are complex entities and can suffer performance differential due to a multitude of factors, including application instance configuration, underlying platform anomalies and resource interference. Defining a threshold based policy for an application as diverse as this would prove extremely difficult as the designer would need extensive knowledge of both the application domain, user usage patterns, seasonal effects and intermittent anomalies. Agent learning techniques such as reinforcement learning are ideally suited to these types of problems. They can learn without prior knowledge of the domain and they update their environmental knowledge based on actual observations. Over time a stationary policy can be found approaching optimality.Since one cannot guarantee the performance of the underlying resource, naively adding similar resources to ensure compliance may not be an optimal strategy.

Recently efforts have been made to develop more adaptive policies towards informing resource allocation decisions. Autoscaling policies should in essence be hyperopic, forgoing short term gains

in an effort to realise greater long term benefits. Policies should also be adaptive to variations in the underlying resource performance and scale in the presence of new or unseen workloads combined with large numbers of resources. Significant work has focussed on decision theoretic planning techniques such as Markov Decision Processes, combined with reinforcement learning techniques. The strength of these approaches is their ability to reason under uncertainty, which maps well onto the stochastic cloud environment. However there are a number of issues that have not been satisfactorily answered by existing research. One of the major drawbacks associated with reinforcement learning techniques when it comes to solving large real world problems, is the length of time it takes to converge to optimal or near optimal policies. In a dynamic scalability context this is the time it takes the learning agent to determine an optimal policy for the given environment. One approach aimed at addressing this problem is to develop a hybrid [29] mechanism in which the learning approach is trained using a good external policy, which potentially could be computed offline using sample data. The problem with this approach is that there are still challenges involved in determining a good initial policy. In addressing these challenges this paper proposes a novel mechanism that takes advantage of the inherent parallelism associated with distributed computational platforms such as computational clouds. The approach involves agents learning in parallel on the same auto-scaling task and sharing information regarding their experiences. This serves two functions, firstly it decreases the length of time its takes agents to determine optimal resource allocations to support application scaling. Secondly the approach is scalable as the number of resources grows, due to the increasing numbers of learners as a function of the number of resources. Finally to facilitate learning in computational clouds we also devise a novel state action space formalism which is capable of learning optimal policies in computational clouds.

The principle contributions of this paper are the design and testing of:

- **Variable workload and performance model**: The development of a model based Q-learning approach, which defines a novel state action space formalism capable of determining optimal resource allocation policies in a realistic cloud setting. Uniquely the output policy reasons across both the variable workload model and the underlying resource performance model.
- **Parallel reinforcement learning**: A parallel reinforcement learning architecture which successfully parallelises Q-learning to speedup convergence rates of agents attempting to auto-scale resources in parallel.

The rest of this paper is structured as follows: *Section 2* explains the causes of performance variability and details our results from microbenchmarking different instance types on Amazon's EC2. *Section 3* provides an overview of relevant and related work in this field. *Section 4* details Markov Decision Processes, the reinforcement learning framework and the parallel reinforcement learning approach. *Section 5* specified our auto-scaling model for both single agent and parallel Q-learning *Section 6* details our experimental findings, leading finally to *Conclusions & Future Work*.

2. CLOUD PERFORMANCE ANALYSIS

The current resource delivery mechanism favoured by IaaS clouds has been largely based on virtualisation technologies. Virtualisation allows for multiple VMs containing disparate or similar guest operating systems, to be deployed in apparent isolation on the same physical host. This multi-tenant environment where agents share and compete for resources on the same host can lead to substantial variability. From the application's performance perspective a variable underlying supportive resource will cause fluctuations in its performance. This section benchmarks a number of IaaS instances on Amazon EC2 to highlight these issues.

2.1. Xen Hypervisor

The sharing of resources amongst the respective VMs is handled by the Virtual Machine Monitor (VMM) [4], an independent domain level monitor which has direct access to the underlying hardware. Xen is a popular open source virtualisation framework, supporting a wide range of

guest operating systems and is used by a large number of cloud providers including Amazon Web Services. Xen facilitates a software layer known as the Xen Hypervisor which is inserted between the server's hardware and the operating system. This allows the physical host to deploy multiple VMs in isolation, decoupling the operating system from the physical host. However whilst virtualisation technologies such as Xen provide excellent security, fault and environmental isolation they do not ensure performance isolation. Koh et. al. [13] state that there are three principle causes of this interference. The first cause is due to the fact that each independent VM on the hypervisor has its own resource scheduler which is attempting to manage shared resources without the visibility of others. Secondly guest operating systems and applications inside a VM have no knowledge about ongoing work in co-located VMs and are unable to adapt in response. Thirdly some hypervisors such as the Xen Hypervisor offload operations such as I/O operations to service VMs. This particularly affects I/O-intensive applications as the Xen hypervisor forces all I/O operations to pass through a special device driver domain, this forces the context to switch into and out of the device driver domain causing interference.

In general the greater the activity of neighbouring VMs, the greater the potential for interference, which directly results in variable application performance. When booting up instances in the cloud an auto-scaling controller will not be able to control the type of VM it is co-located with or the activity of it.

Table I displays three different VM instances and their associated properties currently supported by Amazon EC2. In addition to the performance interference as a result of virtualisation, the type of instance allocated to the application will impact on performance. Amazon rate the I/O performance of the respective instances as *Low*, *Moderate* and *High*, which means the VMs with *High* should receive a greater amount of dedicated I/O bandwidth. As newer hardware is added to the datacenter replacing older models, the physical host your VM resides on could be a determinant in performance also.

2.2. Microbenchmarking EC2

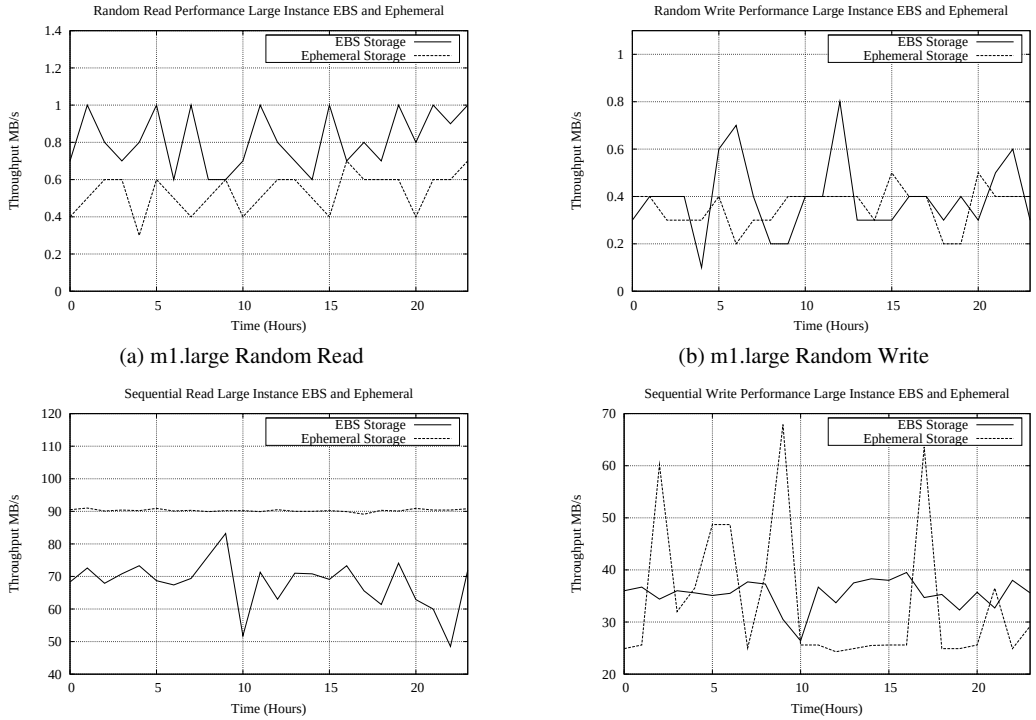
As previously discussed the nature of shared virtualisation instances leads towards performance unpredictability as the underlying CPU, RAM and disk are shared amongst the VMs residing on the physical host. One aspect of the computational resource that is particularly sensitive to interference on virtualised platforms is network and disk I/O. To evaluate this we carried a series of microbenchmarks on a number of EC2 instances to demonstrate disk I/O performance variability exhibited by storage volumes on IaaS clouds. We used the filebench* benchmarking utility to demonstrate sequential/random read/write performance. Filebench is a multi threaded, cross-platform benchmarking tool, capable of running an array of synthetic workloads designed to evaluate disk I/O analysis. The results displayed here highlight the variability deployed applications will observe in relation to I/O performance. Further analyses of performance [11] and performance interference [22] on I/O workloads have previously been published.

Each VM instance on Amazon EC2 can support two different types of storage volume. Instance based or ephemeral storage are storage volumes located within the physical host and shared amongst the resident VMs. Elastic Block Storage (EBS) volumes are network attached storage devices which are connected via a 1Gbps Ethernet connection. We evaluate both of these in terms of performance.

*http://sourceforge.net/projects/filebench/

Table I. Instance types and costs for US-East Virginia

Instance Type	**Memory**	**ECUs**	**Disk**	**Cost** (per/hr)	**I/O Performance**
m1.small	1.7GB	1	160GB	$0.085	Moderate
c1.medium	1.7GB	5	350GB	$0.17	Moderate
m1.large	7.5GB	4	850GB	$0.34	High



(a) m1.large Random Read

(b) m1.large Random Write

(c) m1.large Sequential Read

(d) m1.large Sequential Write

Figure 2. Sequential read/write and random read/write performance variability observed for 2 m1.large instances running on Amazon EC2

In testing disk I/O performance on EC2, we selected four workload profiles to evaluate the random read/write and sequential read/write performance for an m1.small, c1.medium and m1.large instance on the Amazon EC2 cloud. For each experiment we created two instances of each type in a separate availability zone in the US-East Virginia datacentre. The experiments began on a Tuesday evening at $19:00$ UTC and ran for a total of 24 hours. We chose a midweek starting point in order to emulate as closely as possible the variability which would occur during an average working week. The total number of experiments ran over the 24 hours is 1152. The following four workload profiles were designed to evaluate each instance type:

- **Sequential Read**. This workload profile evaluates large sequential file reads. For this experiment the file size was set to be larger than the allocated RAM, 6 Gb in the case of the m1.small/c1.medium instances and 10 Gb in the case of the m1.large instance. This eliminated possible interference due to memory caching. To get a true reflection of the underlying performance, caching was disabled, the iosize was set to 1 MB, and single threaded execution. The experiment was ran for 20 mins, for both EBS and Ephemeral storage each hour, allowing for a steady state evaluation of performance.
- **Sequential Write** This workload profile evaluates large sequential file writes. The individual file write sizes were set to 1 MB each. Caching was again disabled, with syncing enabled. This was executed single threaded. The file sizes generated through writing were 6 Gb for m1.small,c1.medium and 10 Gb for m1.large.
- **Random Read** This workload profile evaluates random file read performance. Caching was disabled, individual read sizes (iosize) was set to 2K, with single threading. Each run generated 128 MB of read data.
- **Random Write** This workload profile evaluates random write performance. Caching was disabled, synchronisation was enabled. The file sizes were set to be larger than the available RAM at 6 GB for m1.small, c1.medium instances and 10 GB for m1.large instances.

Table II. Summary

Profile	Instance Type	Average EBS (MB/s)	Average Ephemeral (MB/s)
Sequential Read	m1.small	78.22	62.01
Sequential Write	m1.small	17.48	30.19
Random Read	m1.small	0.27	0.3
Random Write	m1.small	0.2	0.19
Sequential Read	c1.medium	74.04	107.81
Sequential Write	c1.medium	32.63	29.09
Random Read	c1.medium	0.57	0.3
Random Write	c1.medium	0.2	0.2
Sequential Read	m1.large	60.02	90.26
Sequential Write	m1.large	35.38	33.98
Random Read	m1.large	0.81	0.53
Random Write	m1.large	0.36	0.39

Figure 2 illustrates the sequential/random read/write performance of the m1.large instance for both the EBS and Ephemeral storage volumes. It clearly demonstrates hourly deviations in the total throughput exhibited in both read and write performance. Table II details a tabular breakdown of all the instances tested. The average throughput in both EBS and Ephemeral storage is displayed. According to Amazon both the small and medium instances should achieve a *Moderate* I/O performance whilst the large should be *High*. However the results clearly show high variability across all the instances with the small and medium instances outperforming the large instance for the *Sequential Read* workload profile. This was most surprising as theoretically the large instance should have had a greater share of local disk bandwidth and thus should have had a much higher performance on the ephemeral storage across all the workload profiles. In relation to EBS storage the large instance had a superior read and write performance over the small and medium instance in all profiles except the *Sequential Read* profile. Interestingly EBS volumes also suffer from multi-tenancy issues, as the storage volumes are also virtualised over many devices. In addition the EBS volumes are also bound by network bandwidth and can suffer from network related problems such as congestion. With EBS the user also has no control over the location of the storage volume or positioning relative to other tenants. The results clearly demonstrate the high degree of uncertainty present in relation to I/O performance in the cloud, even on an hourly granularity. Cloud providers such as Amazon do provide Service Level Agreements regarding resource availability but they do not provide any QoS guarantees on performance.

In order to handle this variability an auto-scaling technique must be capable of reasoning across changing workloads and resource performance. It should also be dynamic to fluctuations in real time and capable of handling scenarios it has no prior experience of.

3. BACKGROUND RESEARCH

Threshold based policies are one of the most popular mechanisms for the auto scaling of applications in the cloud both from a commercial and research perspective. Our background research examines these with respect to the most relevant in application scalability in computational clouds. We also examine reinforcement learning approaches to resource allocation and application support, to contextualise the contributions of this paper over previous automated control learning approaches.

3.1. Dynamically Scaling Applications on Clouds : Threshold Based Approaches

Public computational clouds such as Amazon EC2 [1] provide commercial auto scaling solutions to facilitate resource allocation based on predefined metrics. Amazon's Auto Scaling [3] software in conjunction with their proprietary CloudWatch [7] monitoring system can be used to control automated resource allocation decisions based on pre-defined metrics. These metrics stipulate

decision points at which an action is triggered i.e. to add or remove a particular instance type. RightScale [24] similarly allows for the definition of process load metrics which trigger allocation responses once a given metric has been breached. Rightscale facilitates the allocation of resources to applications running across multiple clouds. With this approach a mapping must exist denoting the representative action which must be executed once a specific threshold has been breached. These rule based systems are generally called threshold based approaches in the literature. Threshold based policies such as those employed by RightScale and Amazon's Auto Scaling, tend to focus on scaling at the machine or VM level. They do not facilitate the definition of higher business functions and objectives when scaling a given service or application running on the cloud. Instead the application's resources are altered through allocation and deallocation of VMs. Rodero-Merino et. al. [25] proposes a dynamically scaling solution aimed at addressing this issue. Their proposed solution operates at a higher level of abstraction than those offered currently by IaaS providers. It decomposes high level objectives specified in a Service Description File, which also contains the scalability rules defined by the Service Provider. The paper examines three different scalability mechanisms which facilitate a holistic approach to service management on federated clouds. Their developed application layer is called Claudia and it additionally employs a Virtual Infrastructure Management solution which avoids vendor lock-in and can interface between different clouds. Moran et. al. argued that the mechanisms employed by Claudia were not expressive enough to enable a fine grained control of the service at runtime [17]. The scalability rules defined are specified in an *ad hoc* manner and are not designed with generality. To interchange the abstract level languages used to specify applications behaviour in clouds, they propose the usage of the Rule Interchange Format, in conjunction with the Production Rule Dialect. This generates an appropriate mapping to industry rule engines such as JBoss Drools or Java Jess. These approaches improve upon the industrial led approaches offered by Amazon and Rightscale in attempt to auto scale applications in a more holistic manner but they still require a certain amount of domain knowledge, with rules and conditions required to be defined for different environmental states. Planning in advance the appropriate corrective action can prove extremely difficult especially when one has to consider a large number of possible states [8, 26].

Threshold based approaches have also been developed towards elastic storage solutions. Lim et al. developed an automated controller for elastic storage in a cloud computing IaaS environment based on proportional thresholding [15]. The controller was broken down into three components; a Horizontal Scale Controller for adding and removing nodes; a Data Rebalance Controller, for controlling data transfers; and a State Machine to coordinate the actions. This approach demonstrated speedy adaption to fast changing events, however this is not always optimal given a particular event may be very short lived. Also in the absence of a formalised state space it lacks the predictive power to respond to highly varying workloads.

The benefits of reinforcement learning methods is their ability to reason under uncertainty based only on environmental observations. A modification to the application, or change in the workload request model would possibly require a model change for the threshold based approach. The reinforcement learning approach will adapt to suit the environment based on its own experience.

3.2. Auto-Scaling Resources : Decision Theoretic Approaches

Tesauro investigated the use of a hybrid reinforcement learning technique for autonomic resource allocation [29]. He applied this research to optimizing server allocation in data centres, where homogenous application servers were added and removed based on a hybrid reinforcement learning technique. This work demonstrated the learning approach's capability to maintain sufficient response times, governed by a Service Level Agreement, across a number of applications. David Vengerov combined reinforcement learning with a fuzzy rulebase to allocate CPU and memory from a common resource pool to multiple entities [30]. This work focussed on the problem of distributing resources from a common resource pool to multiple entities, such as in a grid or data centre.

J. Perez et al. [21] applied reinforcement learning to optimise resource allocation in Grid computing. This work focused on optimising the utility of both the users submitting jobs and the institutions providing the resources through Virtual Organisations. Virtual Organisations consist

of a set of individuals or organisations that share resources and data in a conditional and controlled manner [9]. Galstyan et al. [10] implemented a decentralised multiagent reinforcement learning approach to Grid resource allocation. With incomplete global knowledge, and no agent communication, the authors showed that the reinforcement learning approach was capable of improving the performance of large scale grid.

More recently [8] a Q-learning approach was developed for allocating resources to applications in the cloud. This work developed an automated controller capable of adding and removing VMs based on a variable workload model. The author presented an adaptive approach capable of keeping up with a variable workload model driven by a sinusoidal function. Using convergence speedups, Q function initialisation and model change detection mechanisms, the author was able to fine tune the approach. Rao et al [23] developed a reinforcement learning approach to VM autoconfiguration in clouds. In response to changes in demand for applications the VM itself is reconfigured. The approach was able to determine near optimal solutions in small scale systems.

The key difference between our approach and these works is that they focus on determining policies for allocating resources to match a variable workload or user request model. Our approach supports multiple criteria in that the outputted policy considers both the variable workload and the underlying performance model. The approach also facilitates learning across geographical regions where it is capable of reasoning about the temporal performance variances associated with the cloud. In addition to improve the time taken to approximate optimal or near optimal policies we have devised a parallel learning approach. This is first time a parallelised reinforcement learning approach has been applied in this context. Previous approaches to reducing the state space size and improving convergence times involve hybrid [29] learning approaches and utilising function approximation [30] techniques.

4. REINFORCEMENT LEARNING - THEORETICAL FOUNDATIONS

Reinforcement learning has been applied successfully across a range of domains supporting the automated control and allocation of resources [6, 12, 27, 33]. It operates on the basic premise of punishment and reward, with agents biased towards actions which yield the greatest utility. Much of reinforcement learning theory is based on determining optimal policies for Markov Decision Processes.

4.1. Markov Decision Processes

Reinforcement learning problems can generally be modelled using Markov Decision Processes (MDPs). In fact reinforcement learning methods facilitate solutions to MDPs in the absence of a complete environmental model. This is particularly useful when dealing with real world problems as the model can often be unknown or difficult to approximate.

MDPs are a particular mathematical framework suited to modelling decision making under uncertainty. A MDP can typically be represented as a four tuple consisting of states, actions, transition probabilities and rewards.

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

S the set of all possible states represents the agent's observable world. At the end of each time period t the agent occupies state $s_t \in S$. The agent must then choose an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of all possible actions within state s_t. The execution of the chosen action, results in a state transition to s_{t+1} and an immediate numerical reward $R(s_t, a_t)$. Equation 1 represents the reward function, defining the environmental distribution of rewards . The learning agent's objective is to optimise its expected long term discounted reward.

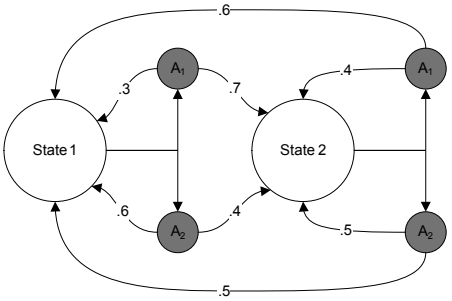


Figure 3. Markov Decision Process with two states and two actions

$$R^a_{s,s'} = E\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\} \tag{1}$$

The state transition probability $p(s_{t+1}|s_t, a_t)$ governs the likelyhood that the agent will transition to state s_{t+1} as a result of choosing a_t in s_t.

$$P^a_{s,s'} = Pr\{s_{t+1} = s'|s_t = s, a_t = a\} \tag{2}$$

The numerical reward received upon arrival at the next state is governed by a probability distribution $q(s_{t+1}|s_t, a_t)$ and is indicative as to the benefit of choosing a_t whilst in s_t. To illustrate the workings a simple MDP, Figure 3 depicts a simple two state, two action MDP. In Figure 3, choosing action A_1 in *State 1* will lead you to *State 2* with a transition probability of 0.7 and back to *State 1* with a transition probability of 0.3. Choosing A_2 will lead you to *State 2* with a transition probability of 0.4 and back to *State 1* with a transition probability of 0.6. An agent currently in *State 1* wishing to transition to *State 2* has a greater probability of doing so should they choose action A_1.

In the specific case where a complete environmental model is known, i.e. (S, A, p, q) are fully observable, the problem reduces to a planning problem [18] and can be solved using traditional dynamic programming techniques such as value iteration. However if there is no complete model available, then one must either attempt to approximate the missing model (model based reinforcement learning) or directly estimate the value function or policy (model free reinforcement learning).

4.2. Q-learning

In the absence of a complete environmental model, model free reinforcement learning algorithms such as Q-learning [32] can be used to generate optimal policies. Q-learning belongs to a collection of algorithms called Temporal Difference (TD) methods. Not requiring a complete model of the environment, TD methods possess a significant advantage. TD methods have the capability of being able to make predictions incrementally and in an online fashion.
The update rule for Q-learning is defined as

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \tag{3}$$

and calculated each time a state is reached which is nonterminal. Approximations of $Q^\pi(s, a)$ which are indicative as to the benefit of taking action a while in state s, are calculated after each time interval. Actions are chosen based on π, the policy being followed. In this research we use an ϵ-greedy policy to decide what action to select whilst occupying a particular state. This means that the agents choose the action which presents it with the greatest amount of reward, most of the time. Let $A'(s) \subseteq A(s)$, be the set of all non-greedy actions. The probability of selection for each non-greedy action is reduced to $\frac{\epsilon}{|A'(s)|}$, resulting in a probability of $1 - \epsilon$ for the greedy strategy.

Estimated action values of each state action pair $Q^\pi(s, a)$ are stored in lookup table form. The goal of the learning agent is to maximize its returns in the long run, often forgoing short term gains in place of long term benefits. By introducing a discount factor γ, $(0 < \gamma < 1)$, an agent's degree

of myopia can be controlled. A value close to 1 for γ assigns a greater weight to future rewards, while a value close to 0 considers only the most recent rewards. This represents a key benefit of policies determined through reinforcement learning compared with threshold based policies. The reinforcement learning based approaches are capable of reasoning over multiple actions, choosing only those which yield the greatest cumulative reward over the entire duration of the episode. The steps involved in Q-learning are depicted by Algorithm 1.

Algorithm 1 Reinforcement Learning Algorithm (Q-learning)

Initialize $Q(s, a)$ arbitrarily
  Repeat (for each episode)
  Initialize s
  repeat
    Choose a from s using policy derived from Q (ϵ-greedy)
    Take action a and observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$;
  until s is terminal

Q-learning can often require significant experience within a given environment in order to learn a good policy. Whilst it is easy to implement and can operate successfully in the absence of a complete environmental model, it does not make efficient use of the data that it gathers as a result of learning [16]. It can also take significant time to approximate the true value function Q^*. In an environment where computational resources are relatively cheap and gathering real world experience costly, an alternative approach is to parallelise the learning process amongst multiple independent learning agents.

4.3. Parallel Reinforcement Learning

A learning agent can speed up the time it takes to learn an approximate model of the environment if it does not have to visit every state and action in the given environment. If instead it could learn the value of states it had not previously visited from neighbouring agents, then the time taken to approximate $Q*$ would be greatly reduced. Parallel learning approaches generally comprise one of the following two approaches. Agents learn individually operating on the same task or agents learn on a subset of the given task. Our approach is an example of the former, where all agents attempt to allocate resources to support the scaling of the same application type. Whilst the agents operate on the same learning task, they will all have different learning experiences due to the stochastic nature of the environment i.e. they will visit different states, choose different actions and observe different rewards. Previous work by Kretchmar [14] has demonstrated the convergence speedups made possible by applying a parallel reinforcement learning approach in a general setting. Each Q-learning agent independently maintains a local Q_l and global estimate Q_g of the approximate Q-values. Q_g is the agent's global representation of Q. It consists of the combined experience of all other learning agents exclusive of their own. This separation of personal experience from that of all the other agents facilitates a weighted aggregation of experience. In environments exhibiting a high degree of randomness an agent may weight its own experience over that of the global experience. Q_g is calculated by aggregating the weighted sum of Q-value estimates of all other agents according to Equation 4.

$$Q(s, a) = \frac{Q(s, a)_l \times Expc_l + Q(s, a)_g \times Expc_g}{Expc_l + Expc_g} \tag{4}$$

The agent makes decisions based on Q(s,a) the weighted aggregation of the local and global estimates of Q. Algorithm 2 depicts the steps involved in our parallel learning approach. Firstly both the local $Q(s, a)_l$ and global $Q(s, a)_g$ value estimates are initialised to 0. This is an optimistic initialisation and encourages exploration in the early stages of learning. The communications arrays

$comms_{in}$ and $comms_{out}$ are initially set to $\emptyset$. For all states visited the agent chooses an action a using an ϵ-greedy policy π with respect to Q the combined global and local estimate. This policy ensures that not all the agent's actions are greedy with respect to Q. Sometimes the agent will choose to act randomly, this balances the tradeoff between exploration and exploitation. A high value of epsilon will bias the agent's decisions towards exploration and a low value allowing the agent to exploit its current knowledge. Based on the policy the agent executes the action a, observes the reward r and next state s'. The agent then updates its estimate of $Q(s,a)_l$ in accordance with Equation 3. If the difference between the Q value estimates are greater than a predefined threshold θ then agent's local estimate is added to the outgoing communications array $comms_{out}$. This information is then transmitted to all other learning agents. Initially quite a lot of data is transmitted between agents, but as the local estimates converge to the global estimates the agents do not transmit anymore information.

Algorithm 2 Parallel Q-Learning

Initialise $Q(s,a)_l = 0$, $Q(s,a)_g = 0$, $Q(s,a) = 0$
$comms_{out}$, $comms_{in} \leftarrow \emptyset$
$\pi \leftarrow$ an arbitrary ϵ-greedy policy w.r.t to Q

repeat
  for all $s \in S$ **do**
    Choose a from s using policy π
    Take action a, observe r, s'
    $Q(s,a)_l \leftarrow Q(s,a)_l + \alpha[r + \gamma \max_{a'} Q(s',a')_l - Q(s,a)_l]$
    $s \leftarrow s'$;
    if $\parallel (Q(s,a)_l - Q(s,a)_g) \parallel > \theta$ **then**
      Add $Q(s,a)_l$ to $comms_{out}$
    end if
    Transmit $comms_{out}$ to all agents
    Receive $comms_{in}$ from other agents
    if $comms_{in} \neq \emptyset$ **then**
      for all $Q(s,a)_g \in comms_{in}$ **do**
        $Q(s,a) = \frac{Q(s,a) \times Expc_l + Q(s,a)_g \times Expc_g}{Expc_l + Expc_g}$
      end for
    end if
  end for
until s is terminal

5. MODEL FOR THE AUTO-SCALING OF APPLICATIONS IN CLOUDS

To facilitate agent learning for a cloud resource allocation problem one must define an appropriate state action space formalism. The revised state action space formalism is designed specifically for obtaining better policies within computational clouds. Our state space representation is tailored to suit the performance related variabilities and the geographical distribution of resources. We define the state space S as the conjunction of three state variables $S = \{u, v, time\}$.

- u is the total number of user requests observed per time period. This value varies between time steps.
- v is the total number of virtual machines allocated to the application, where each virtual machine instance
  $V_i \in \{\{t_1, l_1\}, ...\{t_n, l_m\}\}$. n represents the total number of virtual machine types and m is the total number of geographic regions. t is the virtual machine type and l is the region.

- $time$ is UTC time. It allows the agent to reason about possible performance related effects such as peak time of day in a data centre in a specific region.

The agents action set A contains the set of all possible actions within the current state. The agent can choose to add, remove or maintain the amount of virtual machines allocated to the application. Rewards are determined based on a defined Service Level Agreement (SLA) which is related to performance. The overall reward allocated per time step is given by the following equations.

$$C(a) = C_r \times V_a + \Big\{ \sum_{i=1}^{v} (C_r \times V_i) \Big\} \tag{5}$$

$$H(a) = P_c \times \begin{cases} (1 + \frac{p' - sla}{sla} & if \;\; p' > SLA \\ 0 & else \end{cases} \tag{6}$$

$$R(s', a) = C(a) + H(a) \tag{7}$$

C_r is the cost of the resource, this is variable depending on the type, specific configuration and region. V represents an individual virtual machine instance, with V_a representing the specific virtual machine allocated, deallocated or maintained as a result of action a. H is the overall penalty applied as a result of violating the specified SLA. $P_c \in \Re$ represents a defined penalty constant incurred through violation of the SLA. The total reward $R(s', a)$ for choosing action a, resulting in s' is the combination of the cost of execution and any associated penalties. Whilst many more state observations could be included in this model (CPU utilisation, Memory utilisation, average response time), the approach works surprisingly well given relatively modest state information. In fact previous allocation approaches have had comparable performance to heavily researched open-loop queuing theoretic models, using only current demand as the single observable state variable [28].

Reinforcement learning approaches generally suffer from curse of dimensionality problems, as the size of the state space grows exponentially with each new state variable added. This limitation prevents reinforcement learning techniques from handling environments consisting of very large state and action spaces. To prove the viability of using reinforcement learning to handle application scalability in clouds in lieu of potentially large state and action spaces we have devised a learning architecture aimed at parallelising Q-learning in the context of auto-scaling resources.

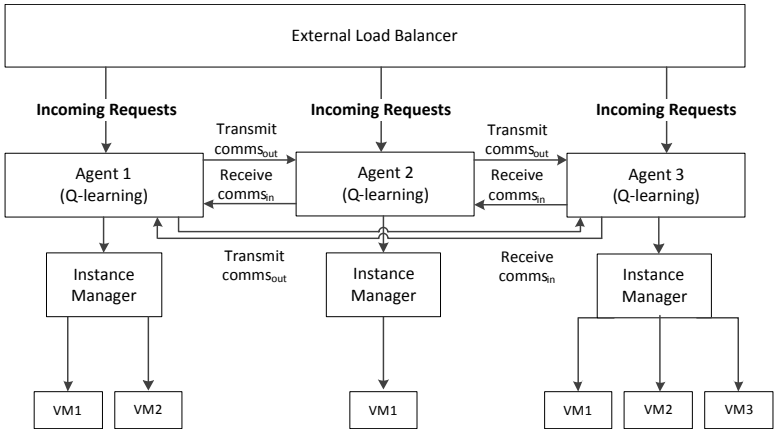


Figure 4. Parallel Q-learning Architecture

Figure 4 presents a high level architecture of parallel Q-learning in a typical cloud environment. Each agent makes its own decisions about the incoming user requests and experiences penalties and rewards independently. Each agent's environment is insular, this means that multiple independent agents do not introduce non-stationarity in each others environments as a direct result of learning in parallel. Based on the allocated numbers of requests the agent must attempt to learn an approximate

individually optimal policy. The agents then share information regarding their observations while operating in the environment. Each agent communicates directly with all the other agents in the system. Actions of whether to add, remove or maintain the existing amount of allocated VMs are executed by the instance manager based on the instructions of the learning agent.

6. EXPERIMENTAL RESULTS

In this section we examine algorithmic performance from two different perspectives. Firstly we investigate the performance of our proposed formalism in the presence of a variable underlying resource and workload model. Secondly we evaluate our parallel reinforcement learning approach and examine its performance with respect to the time taken to converge to optimal policies.

6.1. Experimental Setup

We develop an experimental testbed in Matlab to evaluate our results. Unless stated in the individual experimental sections the following parameters are applied across all experiments.

1. The user request models are generated by a workload simulator. The simulator simulates user requests in an open-loop mode. The open-loop mode generates Poisson requests with an adjustable mean arrival rate ranging from $10 - 150$ reqs/sec.
2. A Service Level Agreement (SLA) of 250 (msecs) governs the maximum allowed response time per request. Each request exceeding this value is deemed in violation of the SLA and incurs a penalty according to $P_c = 1$. The value of the penalty P_c has a direct impact on the distance the policy maintains from the SLA.
3. Q-learning is initialised with the following parametric settings. A value of $\alpha = 0.5$ for the learning rate, ensures that the majority of the error in the estimate is backed up. The discount factor $\gamma = 0.85$, discounts the value of future states. A value of $\epsilon = 0.1$ was chosen, to facilitate adequate environmental exploration. The experimental analysis of Q-learning has the same parametric settings.
4. Four separate data centres are simulated in four disparate geographic regions, closely emulating the data centre regions supported by Amazon's EC2. Our simulations also emulate EC2's instance pricing model, where prices per VM varies between types and regions. In our experiments we define the price of the VM as directly proportional to its configuration, in terms of CPU, memory and disk size i.e. the greater the size of the configuration the greater the cost. Table I in Section 3 outlines the different instance types and their associated configurations.
5. A performance model distribution is constructed by discretising the observed benchmark results into time steps. This allows us to model variable performance over each time step. Taking the lowest and highest observed values per time step, a random performance sample is generated uniformly. We assign a specific peak time in each region when performance variability is increased across all the virtual machines instantiated within that region.

The agent's knowledge is stored in a lookup table (Q-table) and is used to inform decisions over the entire learning episode.

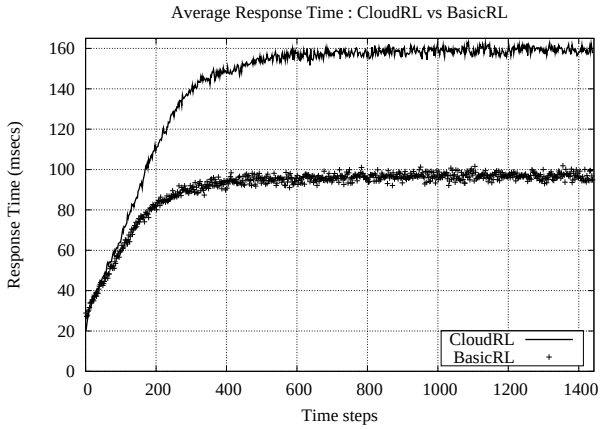
6.2. Optimising for Variable Resource Performance

This experiment analyses the proposed state action space formalism in contrast to previous work where agents ignore resource performance variability. VMs are instantiated with respect to type and location configurations. Learning intervals are discretised into time steps, with each lasting for 60 seconds. Each interval constitutes a decision point, where the agent chooses an action a, is presented with a reward r and the next state s'. Throughout all the experiments the agents action set is limited to the addition or substraction of a single virtual machine each time. Individual VMs are configured in terms of CPU, memory and disk, however the focus of this paper is on I/O variability. A I/O

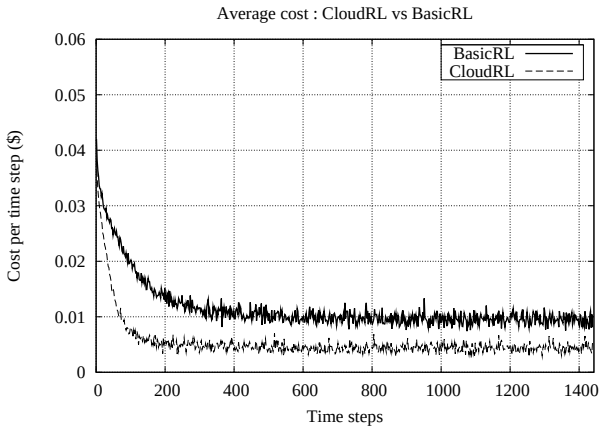
performance model for each VM type is generated based on the observed I/O performance, through the benchmarking of instances. The simulations carried out are based on data gathered through benchmarking live Amazon EC2 instances as outlined in Section 2.

To analyse the benefit of the state space formalism presented in this paper over previous work when dealing with the variabilities of the cloud, the performance of two Q-learning approaches is evaluated. The first approach (hereafter referred to as CloudRL) reasons across both workload and resource variability. The second approach does not reason about the variability of the resource, instead presuming that each additional resource gives a defined performance gain based on its configuration. We refer to this as BasicRL. The addition of the variable resource performance model allows the agent to reason over the addition and removal of resources choosing those which have performed better in the past. Both approaches share the same parametric settings, as outlined in 5 above, however there are significant differences in the respective state space representation. Firstly the BasicRL approach does not incorporate current UTC time into its state space rendering it incapable of reasoning about the possible effects of peak time in a particular region. Secondly it considers all virtual machines to be homogenous/region independent. This approach is consistent with [8] previous research in cloud resource allocation and that of Grids also.

As stated in Section 4 with respect to response times per request, the value specified for P_c has a direct impact on how closely the learned policy approximates the given SLA value. A high penalty will encourage policies that produce lower response times, resulting in increased numbers of VMs



(a) Average Response Time : CloudRL vs BasicRL



(b) Average Costs : CloudRL vs BasicRL

Figure 5. Comparison between the CloudRL and the BasicRL approaches with respect to average costs and average response time

and greater execution costs. The higher overall cost created by the additional VMs is offset by the fact that the SLA violation penalties are so high and the agent will yield a greater reward by decreasing the probability of SLA violation. A low value for P_c will result in greater numbers of SLA violations but the relatively low penalty applied encourages policies that more closely approximate the given SLA, resulting in lower costs. The objective of each approach is to choose resources which facilitate the combined goals of cost reduction and maintain the SLA.

Figure 5a demonstrates the performance comparison between CloudRL and BasicRL with respect to average response time. The CloudRL approach has a higher average response time per request, standing roughly at 160 (msecs) at convergence, however it is still considerably below the SLA of 250 (msecs). BasicRL is unable to reason about the geographical region or type of the virtual machines deployed. This results in a greater probability of choosing suboptimal resources for a given time period. As a result its policy opts to maintain a much higher number of allocated resources to the application. Hence it has a lower response time than the CloudRL approach, but incurs higher average costs as it has more resources allocated to the application, as is depicted in Figure 5b. The multi-criteria Q-learning approach maintains on average a 47% cost saving over the single-criteria approach, in the simulated cloud environment.

6.3. Adjustable Mean Inter-Arrival Rates

This experiment analyses the performance of Q-learning as the mean-arrival rate parameter λ is adjusted. The performance is compared against a user request model of fixed mean. The fixed workload request model consists of a Poisson distribution with a mean-arrival rate of 20 (reqs/sec). Each VM has a theoretical throughput in the range of $1 - 10$ (reqs/sec) which varies per time step in accordance with the resource performance model.

Figure 6 plots the average number of VMs allocated each time step. The fixed workload ($\lambda = 20$) quickly converges to the optimal allocation of VMs. The plot showing the adjustable inter-arrival rates demonstrates the adaptability of the approach as workloads change. Every 2000 time steps the mean arrival rate is increased. After 2000 time steps the workload switches from 40 to 60 (reqs/sec). Figure 6 clearly demonstrates the speed at which the approach can determine the appropriate amount of resources to allocate given the change in the mean number of user requests. Whilst initially it takes time to converge, once an initial policy has been found the approach converges much faster to subsequent model changes as it has already gained experience from previous time steps. After 4000 time steps the average request arrival-rate (λ) shifts again to 120 (reqs/sec). The approach takes greater time to converge to the larger request model. This due to its limited experience of the newly observed states and the greater resource fluctuations as a result of the larger numbers of allocated VMs.

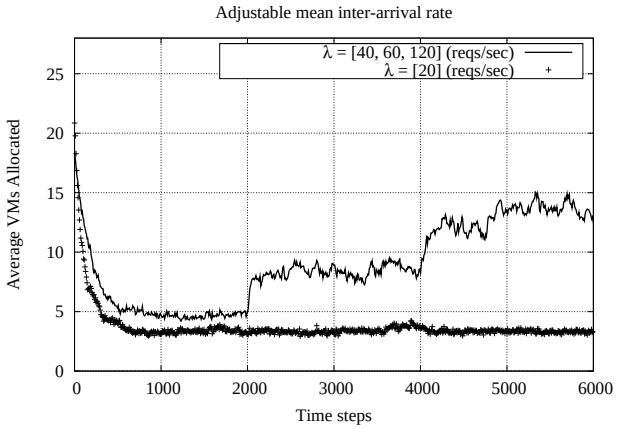
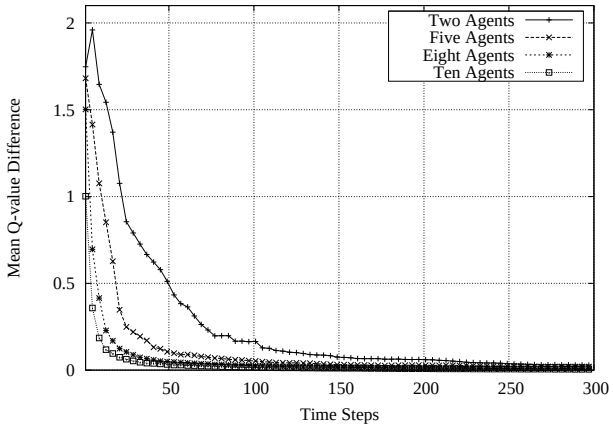
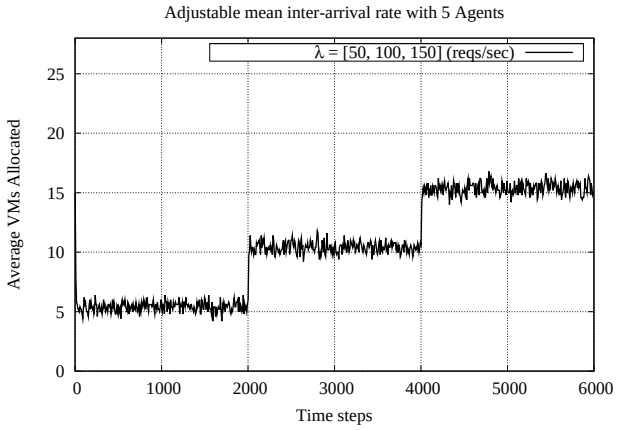


Figure 6. Q-learning performance under varying mean-arrival rates



(a) Average Q-value distance between learning agents



(b) Q-learning performance with 5 parallel agents with adjustable mean arrival-rate

Figure 7. The average Q-value distance between agents learning on the same task in parallel and the performance of 5 parallel agents with adjustable mean inter-arrival rates

6.4. Agents Learning in Parallel

One of the challenges faced when dealing with real world problems with large state spaces is the time it takes to converge to an optimal policy. Usually a substantial number of state visits are required to asymptotically converge to $Q^{\pi}(s, a)$, however in reality often good policies can be determined with far fewer visits. In many real world problems this level of performance is unacceptable. In an auto-scaling context this would potentially lead to expensive allocations in the learning phase that would inhibit the commercial viability of the approach. In order to improve the length of time it takes to converge to an optimal policy we examine the novel parallel learning approach for improving convergence times through the sharing of Q-value estimates.

Whilst each agent attempts to optimise allocations for their respective numbers of user requests, they will encounter similar states. By sharing estimates of the values of these states amongst each other, they can reduce the length of time it takes to learn an optimal policy. The stochastic nature of the workload request model ensures each agent will have a different learning experience but they will all converge on the same result. The goal of this experiment is to speed up the time it takes to converge to a given policy. In order to facilitate a strict analysis of the parallel learning performance, we homogenise the VMs allocated to the application with respect to location and performance. For simplicity and analysis, the throughput of each VM is set to the maximum of 10 (reqs/sec). Figure 7a plots the average distance between Q-function estimates as the number of parallel agents is increased

from 2 to 10 agents. The graph depicts the average distance between agents' approximation of the value of $Q(s, a)$. The graph shows the reduction in the time it takes to converge as the number of agents is increased. Figure 7b shows the resulting performance of 5 agents learning in parallel where the request mean inter-arrival rates are adjusted every 2000 time steps. For the first 2000 time steps the inter-arrival rate parameter is equal to 50 (reqs/sec). With this setting the initial convergence to the average optimal allocation of VMs per time step takes about 80 timesteps. If you compare this to the single agent learning in Figure 6 albeit with a variable performance model, the convergence time as a result of learning in parallel has dropped significantly. As the rate parameter is shifted to 100 and 150 (reqs/sec) at 2000 and 4000 time steps respectively the time taken to converge is also dramatically reduced as a result of both the combination of prior knowledge and learning in parallel.

7. CONCLUSIONS & FUTURE WORK

Reinforcement learning techniques have been successfully applied to automated control problems across a range of domains including economics, multi-agent systems and grid computing. Utilising reinforcement learning techniques to automatically control the scaling of virtual resources in supporting applications offers advantages with respect to reliability, adaptability and autonomy. Threshold based approaches have predominated industry scaling solutions such as Amazon Auto Scaling and RightScale. However the development of effective thresholds which govern allocation decisions requires extensive domain and application knowledge. Relying only on environmental observations the agent learner can adjust its behaviour over time in respect of changes in workload models and resource variability. This work presents a novel reinforcement learning approach aimed at optimising resource allocation decisions to support application scalability in cloud computing environments. Whilst relatively simple, our novel state action space formalism is capable of guiding a Q-learning based agent towards good VM allocation policies in IaaS clouds with no prior experience. Coupled with variable numbers of user requests and resource performance it can effective reason about multiple virtual machine types dynamically dealing with temporal performance issues. To deal with the curse of dimensionality often associated with real world learning problems this work also presents a parallel reinforcement learning approach which is capable of reducing the time taken to converge to optimal resource allocation policies. Each learning agent attempts to approximate optimal policies based on its experience. Through the sharing of information with other agents the time taken to converge to a stable policy is greatly reduced. The combination of parallel agent learning with our novel state space solution enables advanced uncertainty reasoning capabilities over cloud resources.

In the future we wish to integrate the approach into a live virtualised test-bed environment to evaluate performance outside the simulated environment. We are currently developing a hybrid cloud test-bed utilising OpenStack to manage both the internal hardware virtualisation and public cloud resources. The reinforcement learning approach will be evaluated by deploying a standard benchmark application such as Apache Olio and comparing performance against traditional scalability mechanisms. Whilst approaches to support application scalability have generally involved allocating resources in an effort to support a performance related objective such as application response time, we feel that a more high level approach will be needed to address proper application scalability in a cloud context. In this light we plan to combine the holistic approaches specified by Rodero-Merino et al. [16] and extended by Morn et al. [17] allow for the specification of higher level business functions in conjunction with learning.

ACKNOWLEDGMENT

The authors would like to gratefully acknowledge the continued support of Science Foundation Ireland.

REFERENCES

1. Amazon. Amazon elastic compute cloud (amazon ec2), http://aws.amazon.com/ec2/. 2013.
2. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.
3. AutoScale. Autoscale, http://aws.amazon.com/autoscaling/. 2013.
4. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.
5. R. Buyya. Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility. In *CCGRID*, page 1, 2009.
6. Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, M. Lematre, N. Maudet, J. Padget, S. Phelps, J. A. Rodrguez-aguilar, and P. Sousa. Issues in multiagent resource allocation. *Informatica*, 30:2006, 2006.
7. CloudWatch. Cloudwatch, http://aws.amazon.com/cloudwatch/. 2013.
8. 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*, pages 67–74, 2011.
9. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International journal of high performance computing applications*, 15(3):200–222, 2001.
10. A. Galstyan, K. Czajkowski, and K. Lerman. Resource allocation in the grid using reinforcement learning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1314–1315. IEEE Computer Society, 2004.
11. A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Trans. Parallel Distrib. Syst.*, 22:931–945, June 2011.
12. M. Jacyno, S. Bullock, M. Luck, and T. Payne. Understanding decentralised control of resource allocation in a minimal multi-agent system. In *the 6th International Joint Conference on Autonomous Agents and Multi-agent Systems*, 2007.
13. Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. An analysis of performance interference effects in virtual environments. In *In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS*, 2007.
14. R. M. Kretchmar. Parallel reinforcement learning. In *In The 6th World Conference on Systemics, Cybernetics, and Informatics*, 2002.
15. H. C. Lim, S. Babu, and J. S. Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, pages 1–10. ACM, 2010.
16. M. L. Littman. Algorithms for sequential decision making. In *Ph.D. Thesis, Department of Computer Science, Brown University*, 1996.
17. D. Morn, L. M. Vaquero, and F. Galán. Elastically ruling the cloud: Specifying application's behavior in federated clouds. In L. Liu and M. Parashar, editors, *IEEE CLOUD*, pages 89–96. IEEE, 2011.
18. D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
19. P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 13–26, New York, NY, USA, 2009. ACM.
20. P. Padala, K. G. Shin, X. Z. Mustafa, U. Z. Wang, and S. S. Arif. Adaptive control of virtualized resources in utility computing environments. In *In Proceedings of the European Conference on Computer Systems*, pages 289–302, 2007.
21. J. Perez, C. Germain-Renaud, B. Kégl, and C. Loomis. Grid differentiated services: A reinforcement learning approach. In *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*, pages 287–294. IEEE, 2008.
22. X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu. Understanding performance interference of i/o workload in virtualized cloud environments. pages 51–58, 2010.
23. 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*, pages 137–146. ACM, 2009.
24. RightScale. Rightscale, http://www.rightscale.com/. 2013.
25. L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240, 2010.
26. J. A. Rolia, L. Cherkasova, and C. McCarthy. Configuring workload manager control parameters for resource pools. In *NOMS*, pages 127–137, 2006.
27. E. Scalas, M. Gallegati, E. Guerci, D. Mas, and A. Tedeschi. Growth and allocation of resources in economics: The agent-based approach. Quantitative finance papers, arXiv.org, 2006.
28. G. Tesauro. Online resource allocation using decompositional reinforcement learning. In *Proc. AAAI-05*, pages 9–13, 2005.
29. G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3):287–299, 2007.
30. D. Vengerov. A reinforcement learning approach to dynamic resource allocation. *Engineering Applications of Artificial Intelligence*, 20(3):383–390, 2007.
31. VmWare. Vmware virtualisation software, http://www.vmware.com/. 2012.
32. C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge,England, 1989.

33. R. Wolski, J. Brevik, J. S. Plank, and T. Bryan. Grid resource allocation and control using computational economies. In *Grid Computing: Making the Global Infrastructure a Reality*, pages 747–772. John Wiley & Sons, 2003.