URL: A Unified Reinforcement Learning Approach for Autonomic Cloud Management

Cheng-Zhong Xu, Jia Rao, Xiangping Bu
Department of Electrical & Computer Engineering
Wayne State University, Detroit, Michigan 48202
{czxu, jrao, xpbu}@wayne.edu

Abstract

Cloud computing is emerging as an increasingly important service-oriented computing paradigm. Management is a key to providing accurate service availability and performance data, as well as enabling real-time provisioning that automatically provides the capacity needed to meet service demands. In this paper, we present a unified reinforcement learning approach, namely URL, to automate the configuration processes of virtualized machines and appliances running in the virtual machines. The approach lends itself to the application of real-time autoconfiguration of clouds. It also makes it possible to adapt the VM resource budget and appliance parameter settings to the cloud dynamics and the changing workload to provide service quality assurance. In particular, the approach has the flexibility to make a good trade-off between system-wide utilization objectives and appliance-specific SLA optimization goals. Experimental results on Xen VMs with various workloads demonstrate the effectiveness of the approach. It can drive the system into an optimal or near-optimal configuration setting in a few trial-and-error iterations.

1. Introduction

Cloud computing, unlocked by virtualization, is emerging as an increasingly important service-oriented computing paradigm. Management is key to providing accurate service availability and performance data, as well as enabling real-time provisioning that automatically provides the capacity needed to meet service demands. This is because virtualization does not reduce the complexity of a system. In fact, having multiple virtual machines (VMs) running on top of a physical computing infrastructure increases the overall system complexity and poses new challenges in systems management. Recent server market analyses from IDC and Gartner all pointed out the urgent need for deep management and automation technologies that would automate operation operational processes, reduce human errors, and improve the service availability [4, 12]. This is an echo to IBM's early vision for autonomic computing [16]. In this study, we aim to develop machine learning technologies to automate the processes of configuration and reconfiguration of both VMs and VM-based applications (a.k.a. appliances) online.

There are reasons for online VM reconfiguration. When a VM is created from a template or migrated to a new host through live migration [8], its configuration often needs to be adjusted for the new machine to improve resource utilization while meeting the cloud's service level objective (SLO). Because a typical network application has time-varying workloads, there is also a need for dynamic resource allocation in the level of VMs in response to the changing workload.

VM configuration is an error-prone process. In particular, in service consolidation with heterogeneous applications, it is a challenge to figure out the best settings for VMs with different resource demands. Server virtualization has a key requirement for performance isolation. In practice, appliances running on the same physical machine still have chances to interfere with each other. Besides the factor of shared cache, in [20, 10], the authors showed that bad behaviors of an appliance could adversely affect the others' in Xen due to centralized VM scheduling. This phenomenon can also be observed on other virtualization platforms. The interference between VMs would cause performance uncertainty, which makes the VM configuration problem even

harder.

In addition to the VM capacity, application performance is also crucially dependent on its own configuration. It is known that web appliances like Apache and Tomcat often contain more than a hundred parameters to configure when they are deployed. Incorrect settings of the parameters would lead to performance degradation to a large extent. Traditionally, a web system is configured manually, based on operator's experience. Like VM configuration, this is a non-trivial and error-prone task too. Moreover, in multi-component systems like multi-tier websites, the interaction between the components makes performance tuning of the parameters harder. A misconfiguration in one tier may cause misconfiguration in the others. Performance optimization of individual components does not necessarily lead to overall system performance improvement [7]. In [38], the authors demonstrated that in a cluster-based Internet service, when the application server tier was updated with more or less servers, the entire system configuration should be modified to adjust itself to this evolution.

Server virtualization introduces an extra layer of indirection in resource management. The need for dynamic VM configuration/reconfiguration adds one more dimension of challenge to appliance configuration. In particular, the configuration operation must be performed on-line and automatically.

In general, the configuration problem is to find an optimal combination of parameter settings with respect to a performance objective function. There were recent studies on the use of classical combinatorial optimization approaches like hill-climbing and Simplex to automate the tuning process of web applications in a static environment; see [34, 37, 7, 38] for examples. In VM-based dynamic platforms, configurable parameters such as CPU time and memory size are not independent; application parameters often have a concave downward (rather than monotonic) effect on performance. These complicate the optimization problem. The time complexity of the classical optimization approaches prevents them from being applied frequently at run-time for online reconfiguration of the parameters.

There were other studies on the use of feedback control approaches to adaptively reconfiguring VMs [21], web applications [18], and dynamic resource allocation in static and dynamic environments. Traditional feedback control approaches to QoS-aware resource management achieved the goal of service quality assurance to some extent; see [35] and the references therein. But for controllability, they often restricted themselves to one or two control knobs: `MaxClient` parameter in Apache server [18], CPU shares in web server [1, 19, 31, 32, 35], network-I/O bandwidth in streaming server [39], for examples. In [21], the authors applied adaptive control to autoconfiguration of multi-tier web systems on VM-based dynamic environments. But only results from an integral controller (often with limited stability) were reported; and the controllers for different tiers were tuned independently.

In this paper, we present a unified reinforcement learning approach, namely URL, for autoconfiguration of VMs and appliances. Reinforcement learning (RL) is a process of learning by interactions with dynamic environment, which generates optimal control (or action) policies on a given environment state. Unlike adaptive control, RL does not require a model of either the system or the environment dynamics. Also, RL is able to generate policies optimizing a long-term goal based on immediate rewards of actions. Recent studies demonstrated the feasibility of RL in a wide variety of applications, including the design of computer systems; see [3, 13, 28, 30, 29] for examples. There were few reports so far on the use of RL in VM-level resource management or virtual appliances. Designing a RL-based controller to automate the configuration processes poses unique challenges to be discussed in Section 2.

Because each machine contains a large number of configurable parameters, such as cpu time, memory, and network bandwidth, and the VMs on the same host may interfere with each other, the large design space renders traditional optimization and feedback control approaches impractical in real-time resource configuration. The RL methodology finds a good application in online autoconfiguration. The unified RL (URL) approach is applicable for autoconfiguration of both VMs and multi-tier web appliances. It is able to adapt the VM resource budget and appliance parameter settings in a coordinated way to the changing workload to the provisioning of service quality assurance. In particular, the approach has the flexibility to make a good tradeoff between system-wide utilization objectives and appliance-specific SLA optimizations in different VMs.

The rest of this paper is organized as follows. Section 2 presents scenarios to show the challenges in configuration management in dynamic environments. Section 3 presents basic ideas of the RL approach and its application in autoconfiguration. Enhancement of the approach with model-based initialization policies is given in Section 4. Section 5

and Section 6 present the evaluation methodology, settings, and experimental results. Related work is discussed in Section 7. Section 8 concludes the paper with remarks on limitations of the approach and possible future work.

2. Challenges of Autoconfiguration

In this section, we use Xen virtualization platform and web applications as examples to illustrate the challenges in determining good configurations of VMs and appliances in clouds. Similar challenging issues exist in VMWare, VirtualBox and other virtualization platforms.

2.1. Match Configuration to Changing Workload

It is known that performance of a multi-tier web system heavily depends on the characteristics of its workload. Different types of workload have requirements for different amount of resources of different types. TPC-W benchmark, and its successor TPC-APP (`www.tpc.org`), defines three types of workload: *ordering*, *shopping*, and *browsing*, representing three traffic mixes. Because processing of a request involves multiple system components in different tiers, our past studies showed that saturation of the system in the processing of one type of requests does not necessarily mean it cannot handle the others [24]. Bottleneck may also shift dynamically from tier to tier. Application configuration must match the need of current workload to achieve a good performance.

For instance, `MaxClients` is one of the key performance parameters in Apache, which sets the maximum number of requests to be served simultaneously. A too small setting would lead to low resource utilization, and a high value may drive the system into an overloaded condition. How to set this parameter should be determined by the resource demands, the traffic, and resource capacity of its VM. For a VM resource cap, a configuration of this parameter for heavy load may lead to poor performance under lightly loaded conditions.

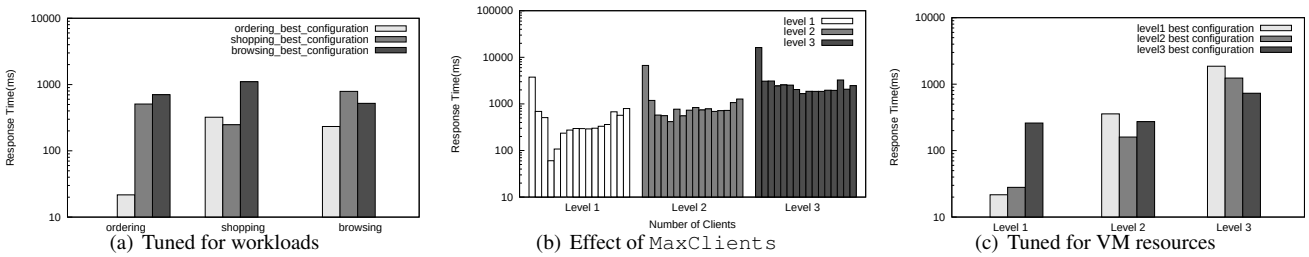
To show the effect of overall configuration, we set up a three-tier Apache/Tomcat/MySQL website, each running on a virtual machine. Recall Apache and Tomcat each has more than a hundred configuration parameters. We restricted our attention to eight performance-critical parameters from different tiers: `MaxClients`, `Keepalive timeout`, `MinSpareServers`, `MaxSpareServers` in Apache server and `MaxThreads`, `Session timeout`, `minSpareThreads`, `maxSpareThreads` in Tomcast server. We assumed the default settings for MySQL parameters.

We tested the application performance using TPC-W benchmark on a cluster of Linux servers, each with two quad-core Intel Xeon processors and eight GB memory. It is expected that each workload has its preferred configuration, under which the system would yield best performance in terms of response time and throughput. We tuned the application configuration manually for each workload. Figure 1(a) shows the performance under different workload mixes. The configuration in each group of bars was best tuned for ordering, shopping, and browsing workload, respectively. From the figure, we can see that there is no single configuration suitable for all kinds of workloads. In particular, the best configuration for shopping or browsing mixes led to extremely poor performance under ordering workload. The first objective of this study is to develop an RL-based autoconfiguration approach to adapt the application configuration to the changing workload.

2.2. Match Configuration to Virtual Machine Dynamics

For a web system hosted on VMs, its throughput is capped by the VMs configurations. Recall that in a cloud computing environment, VMs may need to be reconfigured on-demand in response to the change of underlying computing resources (addition/removal of nodes), fault tolerance, service live migration, and other purposes. Any change of the VM configuration would render the early carefully tuned web system configuration obsolete. Real-time reconfiguration is needed.

In the following, we continue to use the `MaxClients` parameter to show the challenge due to the VM dynamics. In this experiment, we assume an input of fixed workload, but change the VM resource capacity dynamically. We defined three levels of resource capacity: Level-1 (4 virtual CPUs and 4GB memory), Level-2 (3 virtual CPUs and 3GB memory), and Level-3 (2 virtual CPUs and 2GB memory). Figure 1(b) shows the impact of `MaxClients` setting under different VM configurations. We can observe that each VM configuration has its own preferred `MaxClients` setting, leading to the minimum response time. To our surprise, as the VM capacity increases, the best setting of `MaxClients` goes down instead of going up. A possible reason is that because with the VM becoming more and more powerful, it can complete a request in a shorter time.



(a) Tuned for workloads (b) Effect of `MaxClients` (c) Tuned for VM resources

Figure 1: Performance of applications under different settings and different VM configurations.

As a result, the number of concurrent requests will decrease. The measured response time of a request includes its queuing time and processing time. The `MaxClients` parameter controls the balance between these two factors. A large value would help reduce the queueing time, but at the cost of processing time because of the increased level of concurrency. The non-linear relationship between the best choice of `MaxClients` and VM configuration complicates the configuration problem of web applications.

In addition to `MaxClients`, we tested the effects of other parameters under different VM configurations. We observed similar nonlinear relationships between the best settings of the parameters and the VM configuration. Figure 1(c) shows no single configuration is best for all VM configurations. In particular, the performance under Level-2 configuration may even deliver better performance under Level-1 platform. The second objective of this study is to extend the RL-based autoconfiguration approach to adapt the application configuration to VM dynamics.

2.3. Interference of VMs and Heterogeneous Appliances

Server virtualization allows multiple VMs to share computing resources in the same pool. Their performance is hard to be isolated. VM interference poses one more challenge to the autoconfiguration problem. VMs of a physical node are not necessarily homogeneous, running different instances of the same application. VM heterogeneity makes the configuration problem even harder.

Xen virtualization relies on a VM monitor (VMM) to manage the underlying computing resources. It is the lowest level software abstraction, consisting of two components: a hypervisor and a driver domain. The hypervisor provides the guest OS, also called a guest domain in Xen, the illusion of occupying the actual hardware devices, by performing functions such as CPU scheduling, memory mapping and I/O handling for guest domains. The driver domain (dom0) is a privileged VM which manages other guest domains (or VMs) and executes resource allocation policies. Xen provides a control interface in the driver domain to manage the available resources to each VM, including the following three performance-critical configurable parameters: number of virtual CPUs (vcpu), schedule credit(time), and memory size (mem).

In Xen's implementation, privileged instructions and memory writes are trapped and validated by the hypervisor; I/O interrupt is handled by the VMM and data is transfered to VMs in cooperation with dom0. The involvement of the centralized virtualization layer in guest program execution can also be found in other platforms, such as VMware and Hyper-V. Thus, bad behavior of one VM may adversely affect the performance of other VMs by depriving the hypervisor and driver domain resources. In [10], the authors showed that for I/O intensive applications, by setting a fixed CPU share, the credit scheduler does not account for the work done for individual VM in the driver domain. Taking memory and virtual CPU into consideration, the involvement of dom0 and hypervisor in VM execution aggravates the uncertainties in resource to performance mapping. For example, allocating more resource to one VM may result in a performance degradation due to the other VMs' impediment caused by resource deallocation.

We created three VMs on a 2-socket quad-core Xeon server, running TPC-W (e-Commerce), TPC-C (online transaction processing, `www.tpc.org`), and SPECweb (`www.spec.org/web2005`) applications, respectively. Their initial configurations in the form of (vcpu, time, mem) were (2, 256, 512MB) in TPC-W, (1, 256, 1.5GB) in TPC-C, and (2, 512, 512MB) in SPECweb. We defined four workload scenarios for the three applications, as shown in Table 1 of Section 6. Figure 2(a) shows the applications performance under different workload scenarios under fixed VM configurations. We observed that the workload change in TPC-W from browsing to ordering mix (workload-l) boosted the performance of

SPECweb at the cost of TPC-C; the workload reduction in SPECweb (workload-3) led to significant performance degradation of TPC-C. Uncertainties in other workload change scenarios can also be observed. Figure 2(b) shows the normalized application performance due to VM configuration changes. Config-1 moves 1GB memory from TPC-C to SPECweb; Config-2 reduces the virtual CPU of TPC-W from 2 to 1; Config-3 moved 256 schedule credits from SPECweb to TPC-C. There are uncertainties of performance changes. In particular, in the case of config-2, the configuration change of TPC-W from 2 virtual CPUs to 1 unexpectedly causes a big drop of the TPC-C performance. The third objective of this study is to develop approaches for coordinated autoconfiguration of both VMs and appliances to adapt them to both cloud dynamics and workload uncertainty.

3. URL Framework for Autoconfiguration

In general, reinforcement learning is concerned with how an agent ought to take actions in a dynamic environment so as to maximize a long term reward defined on a high level goal [27]. Its outcome is a policy that maps the current environment state observed by the agent to the best action the agent should take. Each action would yield an immediate reward. The "goodness" of an action in a state is measured by a value function which estimates the future cummulative rewards by taking this action. The agent behavior is often formulated as a continuing discounted *Markov Decision Process* (MDP) with unknown transition probabilities. Formally, for a set of environment states $\mathcal{S}$ and a set of actions $\mathcal{A}$, at each time step t, the agent perceives its current state $s_t \in \mathcal{S}$ and the available action set $\mathcal{A}(s_t) \in \mathcal{A}$. By taking an action $a_t \in \mathcal{A}(s_t)$, the agent transits to the next state s_{t+1} and receives an immediate reward r_{t+1} from the environment. The value function of state-action pair (a, s) can be defined as:

$$Q(s,a) = E\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\}, \quad (1)$$

where $0 \leq \gamma < 1$ is a discount factor helping the $Q(s,a)$'s convergence. (1) is often referred to as Q-value function.

The objective of finding an optimal policy is to choose the action that maximizes the Q-value function in each state. It is equivalent to finding an estimation of $Q(s,a)$ which approximates $Q(s,a)$'s actual value. In theory, it is known the average of the sample $Q(s,a)$ values, collected in the past interactions, approximates the actual value of $Q(s,a)$, given sufficiently large number of samples. Q-learning is a temporal-difference (TD) method, which updates $Q(s,a)$ each time when a sample is collected:

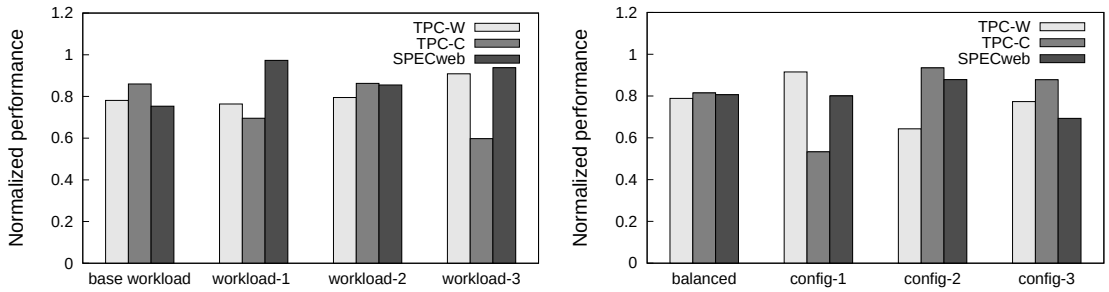
$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha * [r_{t+1} + \gamma * Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)], \quad (2)$$

where α is the learning rate and γ is the discount factor.

The VM configuration task fits within the agent-environment framework. Consider multiple VMs to be configured on one or more physical machines. The environment comprises the VMs and the agent is a VM controller, namely `VM-Agent`. The agent keeps monitoring the performance of each VM and adapt their configurations to the dynamics of the environment online. Each time when the agent changes a VM configuration, it receives performance feedback, either reward or penalty. After sufficient interactions, the agent would obtain good estimations of the Q-value of each state-action pair. Starting from any initial configuration, the agent is able to drive the VMs to optimal configurations in terms of system's throughput, utilization, or any other application-level utility functions. In the case that the VMs are running different components of a single application (e.g. multi-tier website), the objective can also be a service level objective (SLO), defined in the application's SLA.

Figure 3 shows the architecture of URL framework. `VM-Agent` is designed as a standalone daemon residing in the driver domain [23]. It takes advantage of the control interface provided by dom0 to control the configuration of individual VMs. `VM-Agent` manages the VM configurations by monitoring performance feedback from each VM. Re-configuration actions take place periodically based on a predefined time interval. `VM-Agent` queries the driver domain for current state and available actions. Following the policy generated by the RL algorithm, `VM-Agent` selects a re-configuration action and sends it to dom0 for VMs re-configuration. At the end of each re-configuration step, `VM-Agent` collects the performance feedbacks in each VM and calculates the immediate reward. The new sample of the immediate reward is processed by the RL algorithm and `VM-Agent` updates the configuration policy if necessary.

Appliances running on different VMs have their own objectives. Recall that an appliance usually contains a number of performance-critical parameters to be configured and the parameter settings (configuration) should be adaptive to the workload uncertainty and VM environment dynamics. The appliance configuration can be controlled by a RL-based agent, as well. We refer to this agent



(a) Under different workloads

(b) Under different VMs configurations

Figure 2: Uncertainties in VM performance.

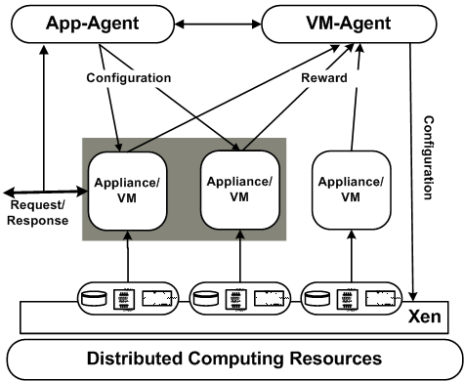


Figure 3: Architecture of URL in a virtualization platform.

as `App-Agent`. It monitors the performance of appliances belonging to the same application and refine their configurations through interactions with the appliances in a way similar to `VM-Agent` to meet the application's SLO requirement [5]. Unlike `VM-Agent`, `App-Agent` is run in a separate VM, requiring no OS or hardware level information.

The `VM-Agent` and `App-Agent` interplay with each other. Because they are targeted at objectives of system-wide utilization and application-specific performance, respectively. Coordinated configuration decisions by `VM-Agent` and `App-Agent` in the URL framework offers an opportunity of tradeoff between the two objectives.

4. RL algorithms for Autoconfiguration

4.1. Autoconfiguration of Virtual Machines

Consider a group of VMs to be configured or reconfigured. The VMs can be running appliances of the same application or different applications, on a single or multiple physical servers in a cloud. A VM contains a number of configurable parameters, which together form a configuration. The configuration is observable in the driver domain. For a group of k VM machines, we define their collective configuration as a state, represented by a vector $s = (s^1, s^2, \ldots, s^k)$. Each element s^i, $1 \leq i \leq k$, is a vector, representing the parameter settings of the i^{th} VM. Without loss of generality, we consider three key VM performance parameters: number of virtual cpus (vcpu), scheduler credit (cpu time), and memory size (mem), assuming all other parameters are configured with default values, or remaining unchanged in reconfiguration. Then, we represent a VM state s^i as $(vcpu, time, mem)$. For example, state (2, 256, 512MB) represents a VM configured with 512MB memory, 256 scheduling credit, and 2 virtual CPUs. Because the total amount of computing resources available to the group of VMs is limited, the parameter settings in different VMs have constraints. For example, the memory size should not exceed the total size allocated to the group of VMs, and the number of virtual CPUs should not be larger than the number of physical cores.

For each configurable parameter, we define three operations: $increase$, $decrease$, and nop, which change the parameter setting in a predefined step size or keep it unchanged. A VM reconfiguration operation is a combination of the operations on the parameters. For the group of VMs under consideration, `VM-Agent` defines each action as a combination of the VMs reconfiguration operations.

The objective of `VM-Agent` is to find a configuration for each VM, which together lead to an optimal system-wide performance, defined as a long-time cumulative reward. The immediate reward of an action is defined according to the overall goal of the `VM-Agent`. In the scenario that the VMs running appliances of the same multi-tier web application, for example, the performance objective is to meet the SLO requirement in terms of response time. Accordingly, the im-

mediate reward r of an action can be as simple as the deviation of measured response time (RT) from the SLO setpoint, i.e. $r = SLO - RT$. For multiple VMs running different applications in the same pool of physical servers, we define a utility function to measure the immediate reward. It is known that each application in cloud computing has its own target SLO in terms of response time and throughput. We define the utility function as a geometric mean of their normalized SLOs, i.e. measuredSLO/targetSLO. This is in agreement with the performance metric defined in VMmark virtualization benchmark [25].

Based on the RL algorithm, the `VM-Agent` constructs a Q-value table for the state-action pairs and issues reconfiguration actions following the maximum Q-value policy. Besides this type of exploitation, the agent will perform exploration actions, following an ϵ-greedy policy. With a small probability ϵ, the agent randomly selects an action possible in the state so as to to capture any changes of the environment for policy refinement.

4.2. Model-based Reinforcement Learning

The basic RL method can learn by itself for autoconfiguration. Starting from any initial policy, the `VM-Agent` would gradually drive the policy to converge to a best one through exploitation and exploration of the environment. Two remaining issues for online autoconfiguration are adaptability and scalability.

Adaptability is the ability of RL algorithms to revise the existing policy in response to environment changes. To adapt current policy to a new one, the `VM-Agent` needs to perform a small percentage of exploration actions to collect new environment information. In production systems, the exploration actions can be prohibitively expensive due to bad client experiences. The RL algorithm usually requires a long time to collect enough samples or accumulate enough experience to derive a new policy. This is not acceptable for online policy generation tasks like VM auto-configuration.

Scalability issue refers to the problem that the number of Q values grows exponentially with the state variables. Recall that the optimal action policy is stored in the Q-value table. Convergence to the optimal policy depends critically on the assumption that each table (state, action) entry is visited at least once. In practice, even if the storage and computational cost for a large Q table is not a concern, the time required to collect sample rewards to populate the Q table would be prohibitively long. For example, in the case of three VMs, each with three configurable parameters, assuming three different settings of each parameter and six actions for each state, the minimum number of interactions required to collect the reward samples for all state-action pairs will be as high as $3^9 * 6 = 118,908$ times!

In the basic RL method, `VM-Agent` updates the estimation of each $Q(s, a)$ value directly from the recently collected immediate reward. In this paper, we deploy an environmental model to generate simulated experiences for Q-value function estimation so as to enhance the agent's scalability and adaptivity. The environment model should be able to capture the relationship between current configuration, action and the observed performance feedback. The model can be trained offline from previously collected samples using supervised learning. Once trained, the model is able to predict the reward values r for un-visited state-action pairs.

Model-based RL with an environmental model for reward prediction offers two advantages: First, model-based RL is more data efficient [2]. With limited samples, the model is able to shed insight on unobserved rewards. Especially in online policy adaptation, the model is updated every time with new collected samples. Then the modified model generates simulated experiences to update the value function, which expedites policy adaptation. Second, the immediate reward models can be reused in a similar environment as the one in which the models were learned. The environmental dynamics in VM configuration task are the time-varying resource demands in each VM. Different models can be learned for different combination of demands in VMs through offline learning.

In model-based RL, the scalability problem is alleviated by the model's ability in coping with relatively scarcity of data in large scales. The conventional table-based Q values can be updated using the batch of experiences generated by the environmental model. However, the table-based Q representation requires a full population using the rewards simulated by the model. This is problematic when the RL problem scales up. In this study, we use another layer of approximation for the Q-value function. It is expected to help greatly reduce the time in updating the value function in each configuration step.

In implementation, we constructed a standard multilayer feed-forward back-propagation neural network (NN) with sigmoid activations and linear output to approximate the immediate reward of actions over certain configurations. The NN selection was due to NN's ability to generalize from linear to non-linear relationship between the en-

vironment and the real-valued immediate reward. More importantly, it is easy to control the structure and complexity of the network by changing the number of hidden layers and the number of neurons in each layer. This flexibility facilitates the integration of the supervised learning algorithm with RL for better convergence. We implemented another NN-based function approximator to replace the tabular form. The NN function approximator takes the state-action pairs as input and output the approximated Q value. It directs the issue of re-configuration actions based on the ϵ-greedy policy.

Algorithm 1 shows the pseudo-code of the `VM-Agent` algorithm. It is run in the monitor domain, keeping watching over the status of each resident VM and recommending reconfiguration to the VM monitor periodically. At each interval, `VM-Agent` records previous state and observes the actual immediate reward obtained after taking the re-configuration action. Next action is selected by ϵ-greedy policy according to output of function approximator Q. `VM-Agent` identifies the workload by examining system-level metrics during last interval using OS-level and hardware-level performance metrics; see [24] for details. The function $select\_workload$ is implemented in supervised learning. The new sample (s_t, a_t, r_{t+1}) then updates the selected environmental model. The Q function approximator is batch-updated as in Algorithm 2.

Algorithm 1 The `VM-Agent` online algorithm

Initialize Q_{appx} to trained function approximator.
Initialize $t \leftarrow 0, a_t \leftarrow nop$.
repeat
  $s_t \leftarrow get\_current\_state()$
  $re\_configure(a_t)$
  $r_{t+1} \leftarrow observe\_reward()$
  $a_{t+1} \leftarrow get\_next\_action(s_t, Q_{appx})$
  $workload \leftarrow identify\_workload()$
  $R_{model} \leftarrow select\_model(workload)$
  $update\_R_{model}(s_t, a_t, r_{t+1}, R_{model})$
  $update\_Q_{appx}(R_{model}, Q_{appx})$
  $t \leftarrow t + 1$
until `VM-Agent` is terminated

Algorithm 2 Update the Q approximator

Initialize Q_{appx} to the current function approximator.
repeat
  $sse \leftarrow 0$
  for n iterations **do**
    $(s_t, a_t, r_t) \leftarrow generate\_sample(R_{model})$
    $target \leftarrow r_t + \gamma * Q_{appx}(s_{t+1}, a_{t+1})$
    $error \leftarrow target - Q_{appx}(s_t, a_t)$
    $sse \leftarrow 0.9 * sse + 0.1 * error * error$
    train $Q_{appx}(s_t, a_t)$ towards $target$
  end for
until $converge(sse)$

4.3. Autoconfiguration for Virtual Appliances.

The principles of model-based RL is applicable to the design of `App-Agent` for autoconfiguration of virtual appliances. Today's web systems contain hundreds of configurable parameters. Although not all of them are performance related, even a small number of performance-critical parameters would lead to a state-action space too large to deal with online by the basic RL method.

Like `VM-Agent`, we design a model-enhanced reinforcement learning approach for `App-Agent`. It assumes an external policy initialization strategy to accelerate the learning process. Briefly, it first samples the performance of a small portion of typical configurations and uses these sample data to predict the performance of other similar configurations. Based on these information, the agent runs another reinforcement learning process to generate an initial policy for the online learning procedure. Such policy initialization is expected to increase the learning adaptability and help the `App-Agent` to drive appliances into a good configuration quickly.

The scalability issue is dealt with by choosing representative states for approximation. A tradeoff exists between the number of states to be considered and their coverage of the states representing the dynamics of the system. We propose a *parameter grouping* technique to aggregate parameters with similar characteristics together so as to reduce the state space. For example, in Apache appliance, both parameters `MaxClients` and `MaxThreads` are limited by the system capacity and often set to the same value in practice; they could be put in the same group. Likewise, both parameters `KeepAlive timeout` and `session timeout` are limited by the number of multiple connection transactions and they could be put in another group. Based on the Q-value of representative configurations, we use a polynomial regression algorithm to predict the appliance performance due to different settings of the parameter and hence the performance of other configurations.

5. Evaluation Methodology and Settings

We developed prototypes of `VM-Agent` and `App-Agent` and evaluated their effectiveness for autoconfiguration of three web appliances: TPC-W, TPC-C, and SPECweb, on Xen-based virtual machines. The physical platform was a cluster of Dell servers connected by a gigabit Ethernet. Each server was configured with 2 quad-core Xeon CPUs and 8GB memory, and virtualized through Xen Version 3.1. Both the driver domain and the VMs were running CentOS 5.0 with Linux kernel 2.6.18. The VMs mounted their file-based disk images through a NFS server on the same cluster.

The three web appliances are target applications of today's server virtualization with different characteristics; their mix is also recommended in VMmark application benchmark for the evaluation of virtualization technologies [25]. TPC-W is a transactional web E-Commerce benchmark. It defines 14 different types of requests for an online bookstore service. TPC-W defines three traffic mixes of different types of requests: browsing, ordering and shopping. In general, ordering mix has more requirements for CPU resource in application server and browsing and shopping mixes impose more burden on database server.

TPC-C is an online transaction processing (OLTP) workload that represents a wholesale parts supplier operating out of a number of warehouses and their associate sales districts. It simulates the activities of terminal operators in each warehouse in the execution of transactions against a database. Each transaction involves a large amount of small disk and network I/O operations. The workload size is determined by the number of warehouses and the population of operators in each warehouse.

SPECweb is a benchmark for evaluating the performance of web servers. It has support for three types of workloads: banking, e-commerce, and support. Each benchmark workload measures the maximum number of simultaneous user sessions that a web server is able to support while still meeting specific throughput and error rate requirements.

We ran the appliances and their VMs in four settings, as illustrated in Figure 4: (a) Each tier of TPC-W application is run as an appliance on different physical servers; (b) The appliances of the same TPC-W service are run in the same physical servers; (c) Multiple homogeneous appliances are run concurrently in the same server; (d) Multiple heterogeneous appliances are consolidated in the server. The performance in (a) and (b) provides a reference to the other cases for comparison or capacity planning.

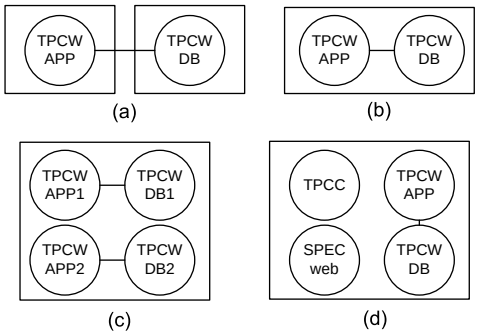


Figure 4: VM and Appliances Settings for Evaluation.

6. Experimental Results

Autoconfiguration has a two-fold objective. One is to provide the assurance of SLA from the perspective of individual applications. The other is to maximize the system utilization without compromising the applications' SLAs.

6.1. Appliance Performance Optimization

The first experiment was designed to show the effectiveness of the URL approach to adapting the configuration of virtualized resources and web appliances. We tested the performance of the `App-Agent` in a three-tier TPC-W implementation: Apache, Tomcat, and MySQL each running in a VM. We dynamically changed workload and VM resource budgets to test the adaptivity of the `App-Agent`.

In the terminology of machine learning, the input traffic mix and the VM resource budget level together form the agent environment. The environment would keep changing, from one scenario to another. Recall in Section 2 we defined three levels of VM configuration: Level-1 (4 virtual CPUs and 4GB memory), Level-2 (3 virtual CPUs and 3GB memory), and Level-3 (2 virtual CPUs and 2GB memory) and showed each VM configuration had its own preferred `MaxClients` and other parameters settings. Likewise, we showed there was no single parameter setting that would work for all kinds of input mixes in TPC-W applications. In this experiment, we turned on the `App-Agent` and let it to configure Apache and Tomcat servers automatically. Figure 5 shows the response time change in a time window of 90 iterations, adapting to the environment changes from scenario (shopping, VM Level-1 resource) to (ordering, VM Level-2) at iteration 30, and to the scenario (browsing, VM

Level-3) at iteration 60. Results from the default configuration and a heuristic trial-and-error method are also included. In the trial-and-error method, the agent tuned the parameters one by one, starting from their default settings, until all performance-critical parameters had been tried. It mimics the way an administrator may use to tune the system manually.

The figure shows that the App-Agent can adapt appliances' configuration to the environment change in about 5 iterations. The trial-and-error method enumerated configurations started from the initial parameter value. It could be trapped with local optimal settings. Although it could find fairly good configurations in a short time, its performance was about 30% worse than the RL-based agent. Although the experiment scale was small, the results show a good sign of potential of the App-Agent.

In the second experiment, we investigated the adaptivity of VM-Agent by setting default values to the parameters of Apache/Tomcat/MySQL servers. Each VM has a configuration of resources. We refer to the combination of the VM configurations of different applications as a resource distribution. From Figure 2, we know that there is no single distribution that would lead to maximal system performance under different workloads. In this experiment, we assumed two homogeneous TPC-W appliances running concurrently in the way of Figure 4(c). The TPCW-1 appliance had input change from initial shopping to browsing at time 30; the TPCW-2 appliance had input change from initial ordering to shopping at time 60. Figure 6(a) shows the throughput change of these two appliances, under the control of VM-Agent for autoconfiguration. From the figure, we can see both VMs suffered performance degradation when the input workload mix changed. This suggests the presence of mismatched VM configuration due to traffic dynamics. The figure also shows that the VM-Agent was able to correct the configuration mismatch in a few steps and maintain the performance at a high level. An examination of the reconfiguration logs revealed that the VM-Agent suggested the *nop* action during this period. As a result, both TPC-W appliances were able to deliver much higher throughput than the ones with only limited configuration control. Because the VM-Agent has limited interactions with the environment, the recommended policies are not necessarily optimal. There is no guarantee the throughput of both appliances are maximized.

From the 60th iteration on, the two VMs started to run browsing and shopping mixes, respectively. Their resource contention and performance interference are more pronounced under this scenario. We examined the effectiveness of VM-Agent by comparing it with a general trial-and-error method. With little system knowledge, an administrator is likely to try different resource configuration by tuning parameters one by one, whenever application performance degradation is detected. Figure 6(b) plots the performance due to this method. The figure suggests that, on average the VMs running browsing and shopping mixes can achieve a maximum throughput of 4500 and 6500 concurrent requests. In comparison, the VM-Agent agent could bring the throughput of both appliances to about 5000 and 7000, respectively with an improvement of about 10%. More importantly, the VM-Agent automatically directed the resource allocation towards target configurations without any human intervention.

6.2. System Performance Optimization

The third experiment was designed to show the effectiveness of the URL approach from the perspective of a system administrator. Consider three applications: TPC-W, TPC-C, and SPECweb, running on their own VMs as in Figure 4(d). The application mix is a consolidation representative defined in VMmark benchmark [25]. Recall that TPC-W is primarily CPU-intensive while TPC-C requires a large amount of disk I/Os and that processing of the requests in SPECweb involves processor and network I/O for dynamic content generation and static image serving. The objective of VM-Agent is to maximize the overall system throughput. In this experiment, we assumed the same initial configuration for each VM: 1.5G memory, 4 virtual CPUs, and 256 credits (for CPU scheduling). We define four workload scenarios for the three appliances in Table 1. The NN models were trained initially from offline collected experience traces which consisted of representative resource allocations uniformly scattered in the state space. With a learning rate of 0.0001 and a momentum of 0.1, the initial training requires approximately 10 minutes. To fit the training within the resource configuration interval (i.e. 60 second in our setting) and reduce the computation requirement in the driver domain, we performed incremental updates of the NN models based on newly collected samples. The incremental computation was limited to 50 iterations and 100 sweeps resulting in a 50-second compute time. We assume that there were always sufficient resources reserved for the driver domain. The reservation of dedicated resources for the control domain is widely employed in practice.

In this experiment, we changed the workload from

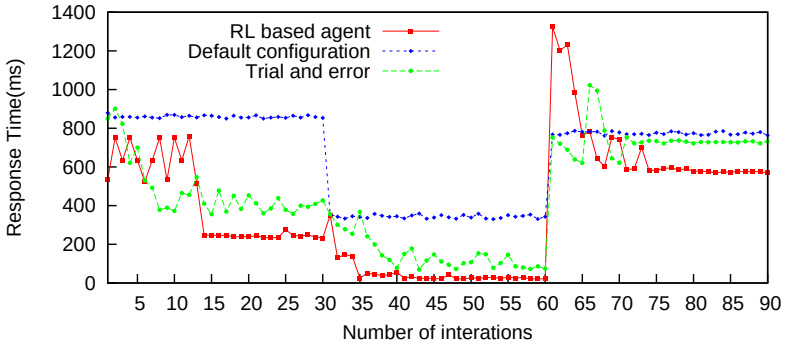
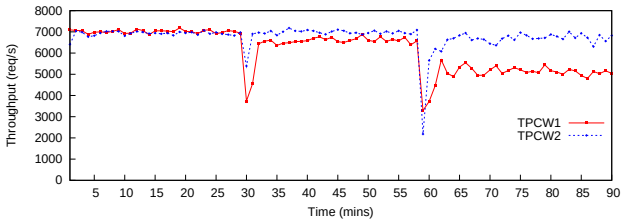
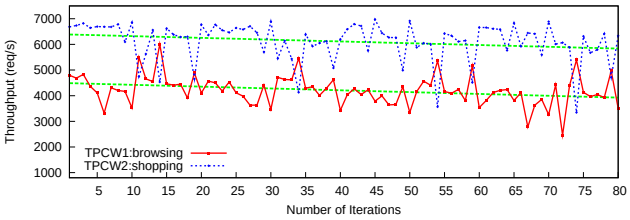


Figure 5: Performance changes with workload and VM resources due to different auto-configuration policies.



(a) Under autoconfiguration of `VM-Agent`



(b) Reconfiguration in trial-and-error method

Figure 6: Throughput change with workload in two homogeneous TPC-W appliances.

Workload-1 to Workload-3 at time 25, to Workload-0 at time 50, to Workload-2 at time 75. In general, change of ordering to browsing mix would increase the CPU utilization; reduction of the operator number tends to reduce the requirement for memory; reduction of banking clients in SPECweb reduces the requirements for both CPU and memory. Figure 7 shows the response time and throughput of each appliance due to model-based `VM-Agent`, in comparison with a model-free approach. The `Max` plots in Figure 7(b) are reference throughput of each appliance, obtained when the appliance was run on a physical server exclusively (with sufficient resources). Due to VM interferences and possible inappropriate configuration, it is expected that the throughput of each appliance would be lower than the reference value.

In the traffic change from Workload-1 to Workload-3 at time 25, SPECweb's workload was reduced from 800 clients to 200 clients. Both of its reference and sustained throughput dropped significantly, as shown in Figure 7(b). From Figure 7(a), we can see there was little change with its response time during the time, as expected. Similarly, in the traffic change from Workload-0 to Workload-2 at time 75, TPC-C's load was reduced from 10 operators to 1 operator. This led to the sharp decrease of its sustained throughput and reference throughput. Its response time was also reduced.

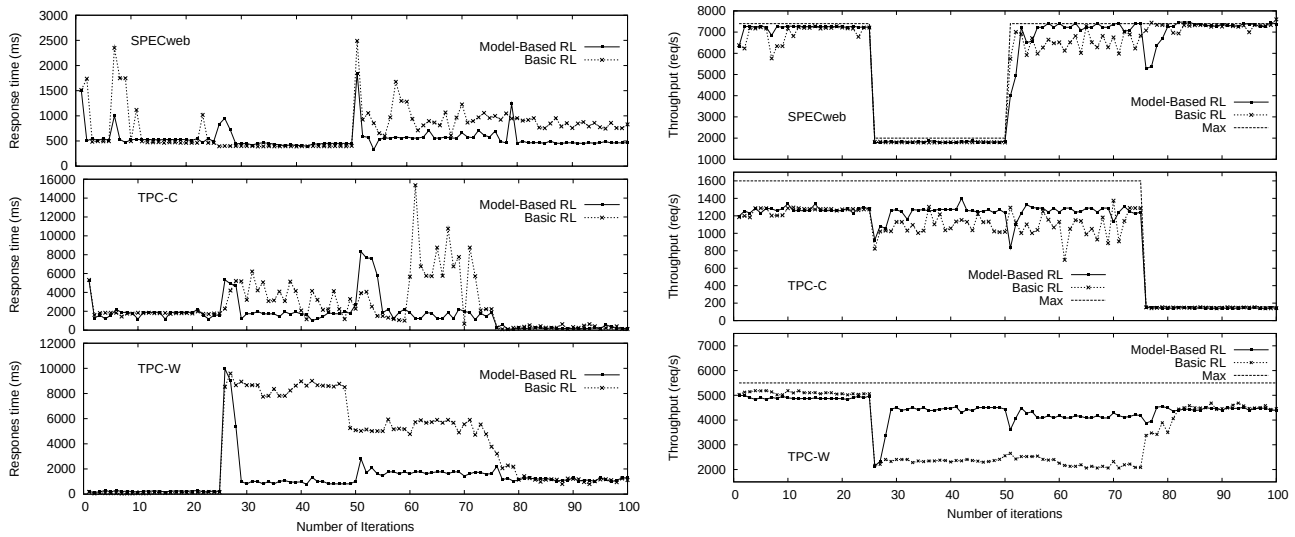
At time 25, TPC-W traffic changed from ordering to browsing mix. A sharp increase of response time and a decrease of throughput were observed. The model-based `VM-Agent` brought the performance back to normal quickly in less than 4 iterations through online resource reconfiguration. At time 50 when traffic changed from Workload-3 to Workload-0, SPECweb load increased from 200 clients to 800, causing its response time to jump from 0.5sec up to 2.5sec. The model-based `VM-Agent` brought it back to 0.5sec almost right away in 2 iterations.

Both the model-based and basic model-free agents were started with the same VM initial configuration. In comparison with the basic RL approach, `VM-Agent` achieves a much higher throughput and lower response time for each benchmark during the online learning. In addition, `VM-Agent` is stable in the sense that its configuration policy is able to be maintained for the same workload input. In contrast, basic RL agent would wagger between several configurations, some causing significant performance penalties.

The advantage of model-based RL approach is due to the model's ability to generalize the environmental changes. In another word, the model-based approach is more data efficient: any change in the environment can be applied to other

Table 1: Workload scenarios in heterogeneous appliances.

	TPC-W	TPC-C	SPECweb
Workload-0	600 clients in ordering mix	50 warehouses & 10 operators	800 banking clients
Workload-1	600 clients in browsing mix	50 warehouses & 10 operators	800 banking clients
Workload-2	600 clients in browsing mix	50 warehouses & 1 operator	800 banking clients
Workload-3	600 clients in browsing mix	50 warehouses & 10 operators	200 banking clients



(a) Response Time

(b) Throughput

Figure 7: Performance change of heterogeneous appliances with the workload change.

state-action pairs. Because the basic RL approach stores Q values separately without interactions, an environmental change can only influence the agents decision when the specific entry in the Q-value table is visited next time.

7. Related Work

In their visionary article, Kephart and Chess defined autonomic computing as a general methodology in which computing systems can manage themselves given high-level objectives from administrators [16]. Its essence is self-management that attempts to "free system administrators from the details of system operation and maintenance and to provide users with a machine that runs at peak performance 24/7." Since then, many research have been devoted to the methodology; see [11] for a recent comprehensive survey on autonomic computing.

Autoconfiguration aims to automate the system configuration process and have the system reconfigured at run-time to improve performance. In general, there are two classes of approaches for system configuration: *multivariate optimization* and *feedback control*. In the multivariate optimization approaches, system configuration is formulated as a combinatorial optimization problem with respect to an application-specific objective (performance) function of configurable parameters. Automated configuration is to find a combination of parameter settings that maximize the performance function. Such approaches were applied to configuration of software systems like Apache server [33], application servers [34, 37], database server [17, 26], and online transaction services [7, 38]. For example, Xi et al. [34] and Zhang et al. [37] applied hill-climbing and Simplex algorithms to search optimal server configurations by adjusting a small group of selective parameters. They treated the system as a black-box and assumed that the application tier configurations were independent of other tiers.

In a multi-component system, configurations of different components interfere with each other. In [7], Chung et al. demonstrated that the performance improvement cannot be easily achieved by tuning individual parameters of each component of a 3-tier website. In [38], Zheng et al. developed a method to generate a parameter dependency graph between the components to reduce the configuration search complexity. Server virtualization adds an additional

dimension for autoconfiguration. In [26], Soror et al. defined a performance function with respect to VM resources in database applications and proposed a multi-resource regression method to search good VM configurations offline.

We note that in the VM-based configuration problem, the parameters of a VM, such as CPU time, memory size, and network bandwidth, the parameters of an application in different components, are not necessarily independent. These complicate the multivariate optimization problem. The high time complexity of classical optimization approaches renders them impractical for online autoconfiguration.

There were other recent approaches based on feedback control for online configuration optimization and QoS-aware resource allocation; see [35] and references therein. For example, Liu et al. proposed a fuzzy control based algorithm to adjust an Apache `MaxClients` parameter online in response to the changing workload [18]; Lu et al. augmented a feedback control design to request scheduling for request delay guarantees [19]; Xu et al. proposed adaptive control control approaches to modulate CPU share between different classes of traffic for page-view response time guarantees [31, 35]; Abdelzaher et al.[1] and Kamra et al. [15] suggested to control admission rate to keep the system utilization bounded. For controllability, most of them restricted themselves to single control knob or actuator of target systems.

Conceptually, feedback control can be applied to individual components of a multi-tier system. Diao et al.[9] demonstrated the potential of such distributed controllers in a two-tier website with an actuator in each tier. But even in such simple designs, setting control granularity and tuning of the controllers became an artful work, because of the cross-tier dependence. Padala et al. attempted to apply adaptive control to automate the configuration of virtual machines of multi-tier web applications [21]. Their approach was based on an assumption that the VMs on the same physical node run in a non-work-conserving mode so that each controller can be tuned *independently*. The sharing mode is too conservative because the underlying computing resources tend to be under-utilized. By the adaptive control approach, each VM was reconfigured online and automatically. However, only results from an integral (I) controller with a single control knob: CPU share, were reported. The controller has very limited stability. Any extension to more practical controllers like Proportional-Integral (PI) and PID controllers with consideration of more performance-critical parameters like memory size and virtual CPU number would immediately increase the complexity of the configuration problem to an untractable level.

In this paper, we present a reinforcement learning approach for online autoconfiguration of VMs and multi-tier web applications. It is a technique of learning by interactions with dynamic systems in clouds. Unlike adaptive control which relies on a system identification procedure to form an explicit model of the system, reinforcement learning executes control over the system *directly*, without imposing model structure: if an action on a system state yields a reward, the tendency to produce that action is strengthened (or reinforced) [27]. It is a technique of learning from delayed "costs," or a method for adaptive optimization of a controlled discrete-time Markov chain with a finite or countable state space. A comprehensive discussion of the method and research progress in this area towards a general framework can be found in our recent work in [36]. A survey of reinforcement learning technology from a computer science perspective can be seen in [14].

Due to its salient features, most recently the RL method and Q-learning in particular, has found its application in various aspects of computer systems. Examples include job and parallel task scheduling [36, 3], server allocation in a server farm [28, 30], power management [29], and self-optimizing memory controller [13]. Designing a RL-based controller to automate the configuration process of VMs and appliances poses unique challenges as we discussed in Section 2.

8. Concluding Remarks

In this paper, we have presented a unified reinforcement learning methodology (URL) for autoconfiguration of VMs and appliances in cloud computing. The URL method relies on an RL-based `App-Agent` to tune the application parameter settings and an RL-based `VM-Agent` to adjust VM configurations online towards maximizing long-term, delayed performance reward. To accelerate the learning process in large scale systems and improve data efficiency in learning, we have developed various models to approximate the immediate reward of actions on VM configurations. We have evaluated the model-based RL approaches in typical web applications on Xen-based virtualization platforms. Experimental results have demonstrated their adaptivity and gain in appliance performance and system throughput.

Current implementation of the URL framework was limited to web appliances, focusing on the resources of CPU and memory. There were no considerations about network-I/O and disk-I/O bandwidth. Another important resource is

L2 cache space. In many-core CPUs, L2 cache tends to be the first-class resource in virtualization. Extension of the URL framework to take into account these resources and in other enterprise applications deserves further study.

The URL framework work opens a new path for autoconfiguration of virtualized resources and appliances in clouds. There are limitations with our work and reinforcement learning approaches in general. RL approaches are able to drive any initial configuration to an optimal one. But the learning process itself takes time. Model-based RL approaches assume a system model constructed offline to approximate immediate rewards of actions and accelerate the learning process. Although the model accuracy has little impact on the quality of final configuration, it affects the quality of intermediate configurations in the learning process. This would hinder the RL approaches from being applied for short-lived appliances and virtual machines. In this paper, we only empirically studied the steady state property of the URL approach. That is the autoconfiguration agent can always finish the reconfiguration process before the appliance workload changes again. The transient property of the learning agent in response to traffic perturbations deserves further investigations in our future work.

The URL framework creates an opportunity for the `App-Agent` and `VM-Agent` to coordinate their actions for trading-off between system-wide utilization and application-specific performance objectives. This opportunity was explored manually in current implementation and the `App-Agent` and `VM-Agent` components were evaluated separately. It is expected their interplay would give us one more knob to tune for cloud performance and service availability on-the-fly. On the other hand, search space explosion due to coordinated configuration of VMs and their resident applications makes the problem of online autoconfiguration even more challenging. We are developing a hybrid approach to complementing the model-free RL method with Simplex space reduction in the beginning of coordination configuration [6].

We also note that current implementation assumes a centralized `VM-Agent` for configuration of VMs in the same physical machine. For configuration of VM clusters across multiple physical machines, a distributed RL approach is under development [22].

Acknowledgments

The authors are grateful to the anonymous reviewers for their constructive comments. This research was supported in part by U.S. NSF grants CNS-0702488, CNS-0914330, and CCF-1016966.

References

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: a control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[2] C. G. Atkeson and J. C. Santamar'ia. A comparison of direct and model-based reinforcement learning. In *In International Conference on Robotics and Automation*, 1997.

[3] A. Bar-Hillel, A. Di-Nur, L. Ein-Dor, R. Gilad-Bachrach, and Y. Ittach. Workstation capacity tuning using reinforcement learning. In *SC*, 2007.

[4] T. Bittman. The future of infrastructure and operations: the engine of cloud computing. In *Gartner 27th Annual Data Center Conference*, December 2008.

[5] X. Bu, J. Rao, and C.-Z. Xu. A reinforcement learning approach to online web system autoconfiguration. In *Proc. of Int. Conf. on Distributed Computing Systems (ICDCS)*, 2009.

[6] X. Bu, J. Rao, and C.-Z. Xu. A model-free learning approach for coordinated configuration of virtual machines and appliances. In *Proc. of MASCOTS*, 2011.

[7] I.-H. Chung and J. K. Hollingsworth. Automated cluster-based web service performance tuning. In *HPDC*, pages 36–44, 2004.

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI*, 2005.

[9] Y. Diao, J. Hellerstein, S. Parekh, H. Shaikh, and M. Surendra. Controlling quality of service in multi-tier web applications. In *Proc. of ICDCS*, 2006.

[10] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in xen. In *Middleware*, 2006.

[11] M. C. Huebscher and J. A. Mccann. A survey of autonomic computing: degrees, models, and applications. *ACM Computing Surveys*, 40(3), August 2008.

[12] IDC. Virtualization and multicore innovations disrupt the worldwide server market. document number: 206035, 2007.

[13] E. Ipek, O. Mutlu, J. F. Martinez, and R. Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *ISCA*, 2008.

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

[15] A. Kamra, V. Misra, and E. Nahum. Yaksha: A self-tuning controller for managing the performance of 3-tiered websites. In *Proc. of IWQoS'04*, pages 47–56, 2004.

[16] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, January 2003.

[17] E. Kwan, S. Lightstone, A. Storm, and A. Wu. Automatic configuratoin of ibm db2 universal database, 2002.

[18] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. S. Parekh. Online response time optimization of apache web server. In *IWQoS*, pages 461–478, 2003.

[19] C. Lu, Y. Lu, T. Abdelzaher, J. Stankovic, and S. Son. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE Trans. on Parallel and Distributed Systems*, 17(9):1014–1027, 2006.

[20] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling i/o in virtual machine monitors. In *Poc. of ACM/Usenix Int. Conf. on Virtual Execution Environment (VEE)*, 2008.

[21] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *EuroSys*, 2007.

[22] J. Rao, X. Bu, C.-Z. Xu, and K. Wang. A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In *Proc. of MASCOTS*, 2011.

[23] J. Rao, X. Bu, C.-Z. Xu, L. Y. Wang, and G. Yin. Vconf: a reinforcement learning approach for virtual machine autoconfiguration. In *Proc. of Int. Conf. on Autonomic Computing (ICAC)*, 2009.

[24] J. Rao and C.-Z. Xu. Online measurement of the capacity of multi-tier websites using hardware counters. In *Proc. of ICDCS*, 2008. (An extended version is to appear in IEEE Trans. on Parallel and Distributed Systems).

[25] L. Roderick, E. Zamost, and J. Anderson. Vmmark: a scalable benchmark for virtualized systems. Technical Report VMware-TR-2006-002, VMware, Inc., 2006. `www.vmware.com/products/vmmark`.

[26] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In *SIGMOD Conference*, 2008.

[27] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[28] G. Tesauro. Online resource allocation using decompositional reinforcement learning. In *AAAI*, 2005.

[29] G. Tesauro, R. Das, H. Chan, J. Kephart, D. Levine, F. Rawson, and C. Lefurgy. Managing power consumption and performance of computing systems using reinforcement learning. In *Advances in Neural Information Processing Systems 20*. 2007.

[30] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 2007.

[31] J. Wei and C.-Z. Xu. eQoS: Provisioning of client-perceived end-to-end QoS guarantees in Web servers. *IEEE Trans. on Computers*, 55(12):1543–1556, Dec. 2006.

[32] J. Wei, X. Zhou, and C.-Z. Xu. Robust processing rate allocation for proportional slowdown differentiation on Internet servers. *IEEE Transactions on Computers*, 54(8):964–977, Aug. 2005.

[33] A. Whitaker, R. S. Cox, and S. D. Gribble. Configuration debugging as search: Finding the needle in the haystack. In *OSDI*, 2004.

[34] B. Xi, Z. Liu, M. Raghavachari, C. H. Xia, and L. Zhang. A smart hill-climbing algorithm for application server configuration. In *WWW*, pages 287–296, 2004.

[35] C.-Z. Xu, J. Wei, and F. Liu. Model predictive feedback control for end-to-end qos guarantees in web servers. *IEEE Computer*, 41(3):66–72, 2008.

[36] G. Yin, C.-Z. Xu, and L. Wang. Q-learning algorithms with random truncation bounds and applications to effective parallel computing. *J. of Optimization and Applications*, 137(2):435–452, March 2008.

[37] Y. Zhang, W. Qu, and A. Liu. Automatic performance tuning for j2ee application server systems. In *WISE*, pages 520–527, 2005.

[38] W. Zheng, R. Bianchini, and T. D. Nguyen. Automatic configuration of internet services. In *EuroSys*, pages 219–229, 2007.

[39] X. Zhou, J. Wei, and C.-Z. Xu. Resource allocation for session-based 2d service differentiation on e-commerce servers. *IEEE Trans. on Parallel and Distributed Systems*, 17(8):838–850, 2006.