VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration

Jia Rao, Xiangping Bu,
Cheng-Zhong Xu, Leyi Wang
Dept. of Electrical and Computer Engineering
Wayne State University
Detroit, Michigan, USA
{jrao, xpbu, czxu, lywang}@wayne.edu

George Yin
Dept. of Mathematics
Wayne State University
Detroit, Michigan, USA
gyin@math.wayne.edu

ABSTRACT

Virtual machine (VM) technology enables multiple VMs to share resources on the same host. Resources allocated to the VMs should be re-configured dynamically in response to the change of application demands or resource supply. Because VM execution involves privileged domain and VM monitor, this causes uncertainties in VMs' resource to performance mapping and poses challenges in online determination of appropriate VM configurations. In this paper, we propose a reinforcement learning (RL) based approach, namely VCONF, to automate the VM configuration process. VCONF employs model-based RL algorithms to address the scalability and adaptability issues in applying RL in systems management. Experimental results on both controlled environments and a testbed of clouds with Xen VMs and representative server workloads demonstrate the effectiveness of VCONF. The approach is able to find optimal (near optimal) configurations in small scale systems and shows good adaptability and scalability.

Categories and Subject Descriptors

D.4.8 [**Performance**]: Measurements, Modeling and prediction

General Terms

Management, Measurement, Performance

Keywords

Virtual machines, Reinforcement learning, Autonomic computing, Cloud computing

1. INTRODUCTION

Virtual machine (VM) technology enables multiple VMs, each running a traditional operating system (OS) and hosting one or more services, to share the resources on the same physical machine. It relies on a VM monitor, residing in between underlying hardware and guest OSes, for resource allocation between the VMs. The monitor facilitates on-demand resource reconfiguration in a timely manner.

There are reasons for online VM resource reconfiguration. When created from a template or arrival at a new host through live migration [7], VMs may need to be reconfigured so as to be incorporated to the new machine for better performance. Due to the time-varying resource demand of typical network applications, it is usually necessary to re-allocate resources to each hosted VM for overall performance. In service consolidation with heterogeneous applications, it is not easy to figure out the best settings for VMs with distinct resource demands. Modern VM monitors provide a rich set of configurable resource parameters for fine-grained run-time control.

Server virtualization has a key request for performance isolation. In practice, applications sharing the physical server still have chance to interfere with each other. In [15, 9], the authors showed that bad behaviors of one application in a VM could adversely affect the others' in Xen [3] VMs. The problem is not specific to Xen; it can also be observed on other virtualization platforms due to the presence of centralized virtual machine scheduling. The involvement of the virtualization layer in the execution of VMs causes uncertainties in VMs' performance. This makes the VM configuration problem even harder.

In Xen, data transfer between the hosted VMs and actual hardware devices is also handled by the VM monitor and a privileged domain. The VM data can be cached in hypervisor and the privileged domain. The effect of resource re-allocations, especially the memory reconfigurations, may not be immediately reflected in VMs' performance. Thus, it is necessary to consider delayed effects in fine-grained runtime VM control.

Recent work in [18, 32] used domain knowledge guided regression analysis to map VM configuration to performance. However, domain knowledge is not always available to administrators. Padala et al. employed a proportional controller to allocate CPU shares to VM-based multi-tier web applications [16]. This approach can not be easily extended to the VM configuration problem with multiple control knobs in terms of more than one configurable resources. The identification of system models that capture the relationship between multiple control and system outputs limits the effectiveness of traditional control theory in the VM configuration problem. The difficulties in determining proper VM configurations motivated us to develop a machine learning approach, reinforcement learning in particular, to automatic VM configuration.

ICAC'09, June 15–19, 2009, Barcelona, Spain.

Reinforcement learning (RL) is a process of learning by interactions with dynamic environment, which generates the optimal control policy for a given set of states. It requires no domain knowledge of the controlled system and is able to generate policies optimizing a long-term goal. RL approaches to the design of computer systems involve several important issues. The application of RL methods is non-trivial due to the exponentially increased state space when systems scale up. In online system management, interaction-based RL policy generation suffers from slow adaptation to new policies. Previous studies showed the feasibility of RL in optimizing server allocation [22, 24], power management [23] and designing self-optimizing memory scheduler [11]. There are no reports yet, to the best of our knowledge, about the application of RL in VM-level resource management. Designing a RL-enabled controller to automate VM configuration process poses unique challenges. In server consolidation, physical machines usually host a large number of heterogeneous VMs concurrently. The resource demands from individual VMs vary over time. The applicability of the RL approaches to large scale VM online auto-configuration problem deserves investigation.

In this paper, we propose a RL-based virtual machine auto-configuration agent, namely VCONF. The central design of VCONF is the use of model-based RL algorithms for scalability and adaptability. We define the reward signal based on summarized performance of each VM. By maximizing the long run reward, VCONF automatically directs each virtual machine configuration to a good (if not optimal) one. Experiments were conducted both in a controlled environment and a cloud computing testbed. Both systems running Xen virtual machines, E-commerce, OLTP and web server workloads demonstrate the effectiveness of VCONF. In the controlled environment, our approach was able to find the best (optimal) configuration for single and multiple VMs running homogeneous workloads. In the cloud computing testbed, VCONF showed good adaptation of policies in online auto-configuration with heterogeneous VMs. Although, there is no optimality guarantee for the derived configurations, VCONF was able to direct arbitrary initial configuration to a better one without performance penalties in any of the VMs.

The rest of this paper is organized as follows. Section 2 reviews the challenges in VM configuration. Section 3 introduces RL methods and their applicability in VM auto-configuration. Section 4 elaborates the design and implementation of VCONF. Section 5 gives the information on testbed settings and experimental results. Related work is presented in Section 6. Section 7 concludes this paper.

2. CHALLENGES IN ONLINE VM RECONFIGURATION

In this section, we briefly review the Xen virtual machine monitor (VMM) and the challenges in determining good configurations of VMs in shared environment.

2.1 The Xen Virtual Machine Monitor

A VMM is the lowest level software abstraction running on the actual hardware. It provides isolation between guest OSes and manages access to hardware resources. Xen [3] is a high performance resource-managed VMM. It consists 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. The hypervisor performs 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 VMs and executes resource allocation policies. Dom0 hosts unmodified device drivers for VMs; it also has direct access to actual hardware. Xen provides a control interface in the driver domain to manage the available resources to each VM. The following configurable parameters have salient impacts on VM performance.

1. **CPU time.** The Xen VMM uses a credit scheduler to schedule CPU on domains. Each VM is assigned a credit number which statistically determines the portion of processor time allocated to each VM.

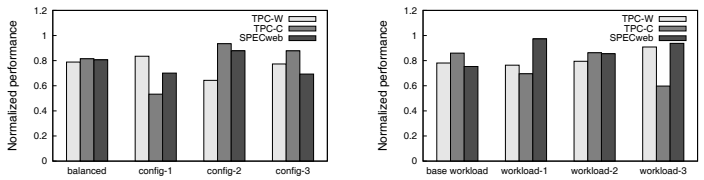
2. **Virtual CPUs.** This parameter determines how many physical CPUs can be used by a VM. The number of virtual CPUs together with the scheduler credit determine the total CPU resource allocated to a VM.

3. **Physical memory.** This parameter controls the amount of memory can be used by a VM. If not set appropriately, the application within the VM may need to communicate with disk frequently, which degrades user-level performance considerably.

2.2 Performance Interference between VMs

In Xen's implementation, privileged instructions and memory writes are trapped and validated by the hypervisor; I/O interrupts are 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 [28] and Hyper-V [10]. Thus, any bad behavior of one VM may adversely affect the performance of other VMs by depriving the hypervisor and driver domain resources. In [9], 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 VMs 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 performance degradation due to the other VMs' impediment caused by resource deallocation.

For example, on a host machine with two quad-core Intel Xeon CPUs and 8 GB memory, we created three VMs running representative server applications: E-Commerce (TPC-W [26]), online transaction processing (TPC-C [27]) and web server benchmark (SPECweb [25]). Details of the testbed and benchmark settings can be found in Section 5. Figure 1(a) shows the impact of resource configuration on application performance. The throughput for each application is normalized to a reference value resulted from running the application on the host exclusively. The balanced configuration in the form of (time, vcpu, mem) were set to (256, 2, 512M) in TPC-W, (256, 1, 1.5G) in TPC-C, and (512, 2, 512M) in SPECweb. The settings optimized the overall performance for all the applications. 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 moves 256 credits from SPECweb to TPC-C. Figure 1(a) suggests that VM performance is sensitive to the process of resource allocation. In certain times, unexpected degraded performance is observed in a VM with even more resources. For example,



(a) Due to configuration (b) Due to workload dynamics

Figure 1: Uncertainties in VM performance.

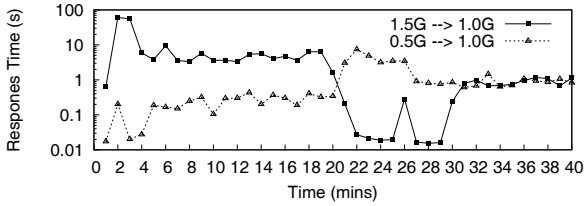


Figure 2: Delayed effect in memory reconfiguration.

in Config-1, with more memory the SPECweb VM had an unexpected worse performance. The reason is due to the increased competition for I/O bandwidth from the TPC-C VM which was de-allocated 1GB memory.

Figure 1(b) plots the performance with fixed VM configurations while changing the load level in each application. The workload selections are defined in Table 1. By reducing the incoming workload to TPC-W, TPC-C and SPECweb, we got three workloads ordered from left to right in the figure. Intuitively, reduced traffic should result in better performance due to alleviated resource contention. However, the assertion does not hold in Figure 1(b). In workload-1, reduced workload in TPC-W alleviated the CPU competition with SPECweb VM. However, with more chance to get scheduled, the SPECweb VM reduced the I/O bandwidth available to TPC-C which ended up with a performance loss. Although overall resource demand decrements, unbalanced VM configurations can still possibly lead to significant performance loss.

2.3 Sequence Dependent VM Performance

Furthermore, I/O data is transferred to and from each domain via Xen and the driver domain, using shared-memory. The hypervisor and driver domain may cache data to expedite VM I/O accesses. VMs with fewer memory may have more data cached by the VMM and driver domain. Thus the way of configuring a VM to its target memory size can potentially affect VMs' performance. Increasing or decreasing to the target memory size can have distinct effects. Due to the caching effects, the influence of a previous configuration may last several configuration steps. Figure 2 shows the delayed effect in memory allocation. The target memory size for TPC-C benchmark was set to 1GB, but from initial settings of 1.5GB and 0.5GB. The memory size was adjusted (at the 20th minute) after the initial configuration produced stable response times. The effect of the adjustment lasted for several minutes before the response time stabilized again.

The complicated resource to performance relationship and possible delayed consequences of previous allocation decisions pose challenges to on-the-fly VM resource management. In server consolidation, the resource allocation may need to be changed due to workload dynamics. The system-wide performance (performance summarized over all hosting applications) should be optimized. This motivated us to design a VM configuration agent to automate the management process. The self-optimizing agent should be able to dynamically alter the VM settings to a better one in consideration of performance interference between VMs and the delayed effects. Reinforcement learning gives a possible solution to the problem.

3. REINFORCEMENT LEARNING FOR VM AUTO-CONFIGURATION

3.1 Reinforcement Learning and Its Applicability to VM Auto-configuration

Reinforcement learning is concerned with how an agent ought to take actions in a dynamic environment so as to maximize long term rewards defined on a high level goal [21]. The RL offers two advantages [24]. First, it does not require a model of either the system in consideration or the environment dynamics. Second, RL is able to capture the delayed effect in a decision-making task.

The outcome of RL is a policy that maps the current state of the agent to the best action it could take. The "goodness" of an action in a state is measured by a value function which estimates the future accumulated rewards by taking this action. The RL-enabled agent performs trial-and-error interactions with the environment, each of which returns an instantaneous reward. The reward information is propagated backward temporally in repeated interactions, eventually leading to an approximation of the value function. The optimal policy is essentially choosing the action that maximizes the value function in each state. The interactions consist of exploitations and explorations. Exploitation is to follow the optimal policy; in contrast exploration is the selection of random actions to capture the change of the environment so as to enable the refinement of existing policy.

The VM configuration task fits within the agent environment framework. Consider the agent as a controller residing in `dom0`. The states are VM resource allocations; possible changes to the allocations form the set of actions. The environment comprises the dynamics underlying the virtualized platform. Each time the controller adjusts the VM configurations, it receives performance feedback from individual VMs. After sufficient interactions, the controller obtains good estimations of the "goodness" of the allocation decisions given current VM configurations. Starting from an arbitrary initial setting, the controller is able to lead the VMs to optimal configurations by following the optimal policy. Through explorations, the controller can modify its resource allocation policy according to the dynamics of VM traffics.

A RL problem is usually modeled as a *Markov Decision Process* (MDP). Formally, for a set of environment states $\mathcal{S}$ and a set of actions $\mathcal{A}$, the MDP is defined by the transition probability $P_a(s,s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ and an immediate reward function $R = E[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s']$. At each step t, the agent perceives its current state $s_t \in \mathcal{S}$ and the available action set $\mathcal{A}(s_t)$. By taking 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 taking action a in state s can be defined as:

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

where $0 \leq \gamma < 1$ is a discount factor helping $Q(s,a)$'s convergence.

3.2 Formulation of VM Configuration as a RL Task

The VM configuration task is naturally formulated as a continuing discounted MDP. The goal is to optimize the overall VM(s) performance. We define the reward function based on individual VM's application level performance. The state spaces are the hardware configuration of VMs which are fully observable in the driver domain. Actions are the combination of the change to the configurable parameters. The configuration task is formulated as following:

The reward function. The long-term cumulative reward is the optimization target of RL. In the VM configuration task, the desired configurations are the ones which optimize system-wide performance. The immediate rewards are the summarized VM(s) performance feedbacks on the resulted new configuration. The performance of individual VM is measured by a score which is the ratio of current throughput ($thrpt$) to a reference throughput plus possible penalties when response time ($resp$) based SLAs (Service Level Agreement) are violated:

$$score = \frac{thrpt}{ref\_thrpt} - penalty.$$

$$penalty = \begin{cases} 0 & \text{if } resp \leq SLA; \\ \frac{resp}{SLA} & \text{if } resp > SLA. \end{cases}$$

The reference throughput (ref_thrpt) values are the maximum achievable application performance under SLA constraints in current hardware settings. We obtained the reference for one application by dedicating the physical host and giving more than enough resources to the corresponding VM. A low score indicates either lack of resource or interference between VMs, both of which should be avoided in making allocation decisions. Then, the immediate reward is the summarized scores over all VMs. As suggested by virtualization benchmarks [6, 29] for summarized performance, we define the reward as:

$$reward = \begin{cases} \sqrt[n]{\prod_{i=1}^{n} w_i * score_i} & \text{if for all } score_i > 0; \\ -1 & \text{otherwise,} \end{cases}$$

where w_i is the weight for the i^{th} VM, $1 \leq i \leq n$. We strictly refuse the configurations of negative scores (i.e. violation of SLA) by assigning a reward of -1. In the case of soft SLA thresholds, the reward function can be revised correspondingly to tolerate transient SLA violations.

The state space. In the VM configuration task, the state space is defined to be the set of possible VM configurations. In the driver domain, VM configurations are fully observable. States defined on the configurations are deterministic in that $P_a(s,s') = 1$, which simplifies the RL problem. We define the RL state as the global resource allocations:

$$(mem_1, time_1, vcpu_1, \cdots, mem_n, time_n, vcpu_n).$$

where mem_i, $time_i$ and $vcpu_i$ are the i^{th} VM's memory size, scheduler credit and virtual CPU number, respectively. Since the hardware resources available to VMs are limited, constraints exist. The value of mem should not exceed the total size of memory that can be allocated to VMs. In addition, $vcpu$ should be a positive integer, not exceeding the number of physical CPUs and the scheduler credit be positive, too.

The actions. For each of the three configurable parameters, possible operations can be either *increase*, *decrease* or *nop*. The actions for the RL task are defined as the combinations of the operations on each parameter. For parameters like *time* and *mem* that are continuous, we quantify them by defining change steps. Memory is reconfigured in unit of 256MB; scheduler credit changes in a step of 256 credits and virtual CPU number is incremented or decremented by one at a time.

An action is invalid if by taking the action, the target state violates state constraints. Another restriction for taking action is that only one parameter is considered at a time and only one-step reconfiguration is allowed. It follows the natural trail-and-error method that searches the configuration state space exhaustively. More importantly, resource adjustment in small steps smooths the configuration process.

3.3 Solutions to the RL Task

The solution to a RL task is an optimal policy that maximizes the cumulative rewards at each state. It is equivalent to finding an estimation of $Q(s,a)$ which approximates its actual value. The experience-based solution is based on the theory that the average of the sample $Q(s,a)$ values collected approximates the actual value of $Q(s,a)$ given sufficiently large number of samples. A sample is in the form of (s_t, a_t, r_{t+1}). The basic RL algorithms in experience-based solution are called *temporal-difference* (TD) methods, which update $Q(s,a)$ at each time a sample is collected:

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

where α is the learning rate and γ is the discount factor. The Q values are usually stored in a look-up table and updated by writing new values to the corresponding entries in the table. In the VM configuration task, the RL-based agent issues reconfiguration actions following an ϵ-greedy policy. With a small probability ϵ, the agent picks a random action, and follows the best policy it has found for most of the time. Starting from any initial policy, the agent gradually refines the policy based on the feedback perceived at each step.

4. THE DESIGN AND IMPLEMENTATION OF VCONF

In this section, we introduce VCONF, a RL-based VM auto-configuration agent. Including multiple VMs in the RL problem poses challenges to the adaptability and scalability of VCONF. We address the challenges by employing model-based RL methods with two layers of approximation.

4.1 Overview

VCONF is designed as a standalone daemon residing in the driver domain. It takes advantage of the control interface provided by dom0 to control the configuration of individual VMs. Figure 3 illustrates the organization of VCONF and the Xen virtualization environment. VCONF manages the VM configurations by monitoring performance feedbacks from each VM. Reconfiguration actions take place periodically based on a predefined time interval. VCONF queries the driver domain for current state and computes valid actions. Following the policy generated by the RL algorithm, VCONF selects an action and sends it to dom0 for VMs reconfiguration. At the end of each step, VCONF collects the performance feedbacks in each VM and calculates the immediate reward. The new sample of the immediate reward

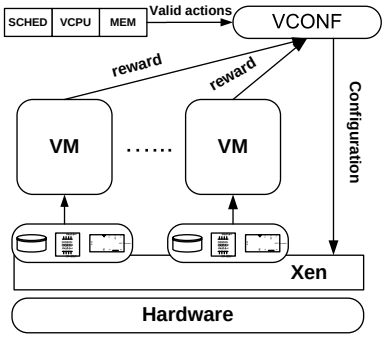


Figure 3: The organization of VCONF.

is processed by the RL algorithm and VCONF updates the configuration policy accordingly. VCONF implements a basic look-up table based Q function for small systems. In a larger system, VCONF employs model-based RL algorithm for adaptability and scalability.

4.2 Adaptability and Scalability

Adaptability is the ability of RL algorithms to revise the existing policy in response to the change of the environment. To adapt current policy to a new one, the RL agent needs to perform a certain amount of exploration actions, which are believed to be suboptimal actions leading to bad rewards. In production systems, the explorations can be prohibitively expensive due to bad client experiences. The RL algorithm usually requires a long time for new samples collection before a new policy can be derived. This is not acceptable for online policy generation tasks like VM auto-configuration.

Scalability issues refer to the problem that the number of Q values grows exponentially with the state variables. In a look-up table-based Q implementation, the values are stored separately without interactions. The convergence of the optimal policy depends critically on the assumption that each table entry be visited at least once. In practice, even if the storage and computation complexity for a large Q table are not a concern, the time required to collect sample rewards to populate the Q table is prohibitively long.

Instead of updating each $Q(s, a)$ value directly from the immediate reward recently collected, VCONF employs environment models to generate simulated experiences for value function estimation. The environment models are essentially data structures that capture the relationship between current configuration, action and the observed reward. The model can be trained from previous collected samples in the form of (s_t, a_t, r_{t+1}) using supervised learning. Once trained, a model is able to predict the r values for unseen state-action pairs.

The use of environment models offers two advantages for RL tasks: 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. The modified model generates simulated experiences to update the value function, and hence expedites policy adaptation. Second, the immediate reward models can be reused in a similar environment. 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. We call such a combination a workload. In online adaptation, once VCONF identifies the resource demand is similar to a previous workload, the corresponding model is re-used. Instead of starting from scratch, the reuse of previous models is equivalent to starting from guided domain knowledge, which again improves online performance.

Algorithm 1 The VCONF online algorithm

1: **Initialize** Q_{appx} to trained function approximator.
2: **Initialize** $t \leftarrow 0, a_t \leftarrow nop$.
3: **repeat**
4: $s_t \leftarrow get\_current\_state()$
5: $re\_configure(a_t)$
6: $r_{t+1} \leftarrow observe\_reward()$
7: $a_{t+1} \leftarrow get\_next\_action(s_t, Q_{appx})$
8: $workload \leftarrow identify\_workload()$
9: $R_{model} \leftarrow select\_model(workload)$
10: $update\_R_{model}(s_t, a_t, r_{t+1}, R_{model})$
11: $update\_Q_{appx}(R_{model}, Q_{appx})$
12: $t \leftarrow t + 1$
13: **until** VCONF is terminated

In model-based RL, the scalability problem is alleviated by the model's ability in coping with relatively scarcity of data in large scale problems. The conventional table-based Q values can be updated using the batch of experiences generated by the environment 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 VCONF, we use another layer of approximation for the value function, which helps to reduce the time in updating the value function in each configuration step.

4.3 Model Initialization and Adaptation

We selected standard multi-layer feed-forward back propagation neural network (NN) with sigmoid activations and linear output to represent the environment model. The selection was due to NN's ability to generalise from linear to non-linear relationship between the environment 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 supervised learning algorithms with RL for better convergence. The performance of model-based RL algorithms depends on the accuracy of the environment model in generating simulated samples. Thus, the training samples used to train the model should be representative. We generated the training samples for the model by enumerating important configurations. In the implementation of Q function, an NN-based function approximator replaces the tabular form. The NN function approximator takes the state-action pairs as input and outputs the approximated Q value. It directs the selection of reconfiguration actions based on the ϵ-greedy policy.

Algorithm 1 shows the VCONF online algorithm. VCONF is designed to run forever until being stopped. At each configuration interval, VCONF records the previous state and observes the actual immediate reward obtained. Next action is selected by ϵ-greedy policy according to output of function approximator Q. VCONF identifies the workload by examining system-level metrics during last interval. The function *select_workload* is implemented in a way similar to the one in [17] using supervised learning except that the output is the predicted workload type. 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 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 $converge(sse)$

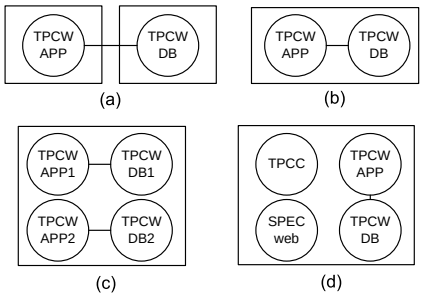


Figure 4: The design of experiments.

5. EXPERIMENTAL RESULTS

5.1 Methodology

We designed a set of experiments to show the effectiveness of the RL-enabled VCONF in VM auto-configuration. Figure 4 lists four different VM settings. The experiments are divided into two parts. In the first part, VCONF was evaluated in controlled environments in which the number of applications and resources were limited to a small set. The multi-tier TPC-W benchmark was selected as the application. As in Figure 4(a), its reference performance was obtained by running TPC-W application server and database server on two separate physical servers exclusively. Figure 4(b) shows a single instance of TPC-W with two tiers. Since TPC-W is primary CPU-intensive, the memory parameter was fixed in this controlled environment. By adjusting CPU resources allocated to each tier, VCONF is to maximize TPC-W's throughput. The experiment in Figure 4(c) augmented the single application problem by adding another instance of TPC-W. VCONF needs to optimize system-wide performance finding balanced CPU allocation schemes for competing applications. In the second part, restrictions on the number of applications and resources in consideration were relaxed. As in Figure 4(d), three applications with heterogeneous resource demands were consolidated in the host. The memory parameter needs to be considered. In the scaled-up problem, the state-action space is considerably larger than the controlled experiments. VCONF's implementation of model-based RL algorithm was evaluated and compared with basic RL methods.

5.2 Experiment Settings

The machines used in the experiments consist of virtual servers, client and compute machines. The physical machines for virtual hosting are Dell PowerEdge1950 with two quad-core Intel Xeon CPU and 8GB memory. In the controlled experiment, all VMs were pinned to the first four cores. We separated the RL related computation to a compute node in order to avoid possible VM performance interference. All the client and compute nodes were the same model Dell machines and were connected by Gigabyte Ethernet network.

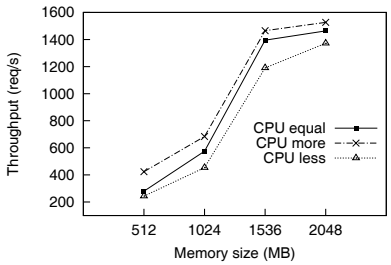


Figure 5: TPC-C performance in different settings.

We used Xen version 3.1 as our virtualization environment. Both dom0 and the guest VMs were running CentOS Linux 5.0 with kernel 2.6.18. The VMs mounted their file-based disk images through a NFS server. For the benchmark applications, MySQL, Tomcat and Apache were used for database, application and web servers. The VM configuration actions were issued through dom0's privileged control interface `xm`.

We selected the TPC-W [26], TPC-C [27] and SPECweb [25] benchmarks as the workloads running within the VMs. They are typical server applications in today's data centers which are the targets of virtualization technology.

5.3 Applicability of RL-based VM Autoconfiguration

First, we studied the applicability of RL algorithms in the VM configuration task. In [18], the authors assumed independence of configuration parameters. With this assumption, VM configuration task can be easily solved by greedy search in each resource dimension. They showed database query costs drop linearly with more CPU shares. The cost is independent with the memory size allocated to the VM. Thus, greedy search together with linear regression are sufficient to find the optimal configuration without visiting every possible configurations. However, the independence assumption does not always hold. Due to the involvement of dom0 in VM execution, applications hungry for memory can be affected by CPU-intensive applications. Figure 5 plots the performance of TPC-C under different CPU settings: equal, more and less. The VM competing for resource is an instance of TPC-W. "equal", "more" and "less" indicate 50%, 80% and 20% CPU allocations for TPC-C, respectively. The figure suggests a strong correlation between memory and CPU in determining application performance. That is, regression based greedy search approach needs to search the entire configuration space.

The RL algorithm does not assume any model of the system in consideration. It derives policies from interactions and continues to refine the policy with newly collected experiences. We validated the effectiveness of RL methods in VM auto-configuration starting from a simple problem. As showed in Figure 4(b), a two-tier TPC-W application was hosted by the virtual server. We assume the application throughput as the optimization target. Requests execution in TPC-W involves processing on both tiers. Thus, the resulted performance is affected by the processing capacity on both tiers. TPC-W defines three different traffic mixes: shopping, browsing and ordering mix. Different traf-

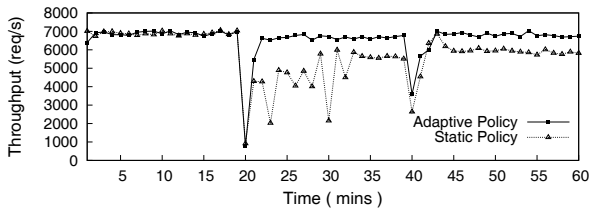


Figure 6: VCONF performance with TPC-W application.

fic mixes put processing pressure on distinct tiers. Thus, it is not easy to determine the CPU assignment to each tier for balanced configuration. Moreover, due to dynamic CPU demands from different traffic mixes, existing CPU allocation needs to be frequently revised.

To limit the problem size, we restricted each tier to have up to 3 virtual CPUs. Only three scheduler credit assignments were selected: equal share, tomcat tier with 80% share and Database tier with 80% share. The resulted state space contains 27 configurations. VCONF was deployed with a table-based Q function which was initialized to all zeros. We used the $Sarsa(0)$ algorithm [20] with $\alpha = 0.1, \gamma = 0.9, \epsilon = 0.1$ to drive the configuration agent, the configuration interval was set to 60 seconds. If otherwise specified, the same RL parameters and interval were used in the remaining experiments. The agent exits until the Q function converges. An optimal configuration policy can then be derived from the Q table. The RL learning process was repeated for above three workloads resulting in three policies.

Figure 6 shows the online performance of VCONF with adaptive and static policies. The plots are the achieved throughput in TPC-W. During the testing, workloads were dynamically switched in the order of ordering, shopping and browsing mix every 20 minutes. VCONF with adaptive policies continuous monitored the system level performance metrics and identified workload changes. The policies were switched accordingly as recommended by VCONF. Configured with a static initialization policy, VCONF revised the initial policy only based on online interactions. The figure suggests that both RL agents were able to automatically drive an inappropriate configuration to a better setting in a small number of steps. The TPC-W throughput was brought up and maintained at a high level. The adaptive agent achieved the optimal performance, which is the best possible result for the RL approach. The one with the static policy also showed the effectiveness of RL, but with limited adaptability to a new policy when traffic changed.

5.4 RL-based System Wide Performance Optimization

In this experiment, we add one more TPC-W application to the problem. The goal of the RL agent is to maximize the cumulative reward which is defined as the summarized performance scores over both TPC-W instances. Adding more applications complicates the VM configuration problem. As the state space grows, the time required for the RL agent to obtain an optimal policy in online interaction becomes prohibitively long. For example, in the case of two TPC-W instances with two application server VMs and two database VMs, the state space increases to around 400 states if the state is defined similarly as in the first experiment. It would take the agent more than 400 minutes to visit every state. The convergence of the Q function usually requires multiple visits to each entry and the RL agent following the $\epsilon-greedy$ policy may not update different entries each time. Thus, the resulted time required for online RL learning is unacceptable. One possible solution is to pre-define a policy that guides the RL agent in online learning. Upon an acceptable good policy is derived from online guided interactions, the RL agent is handed over to the generated policy.

We designed the initial policy to be as simple as visiting different configurations at each step. As more states are visited, the RL agent performs sweeps of batch updates to the Q table using the collected rewards. In this experiment, the pre-defined policy terminates when all configurations have been visited. Due to the presence of delayed effects, different sequences of visiting may receive different rewards. Theoretically, the Q function approximates its actual value only if the agent perceives the effect of all the state-action pairs. Thus, the generated policy still needs online refinement before the optimal policy is achieved. In practice, near optimal policy often satisfies users' requirement.

The RL-based VM resource management is to optimize both applications in operation. Because VMs with identical resource demands can be configured to have the same resource allocation. To test VCONF, the hosted TPC-W applications ran different traffic mixes. Randomly selecting two traffic mixes as the input traffic to the VMs forms three different resource demands for the whole system. The optimization goal for the RL agent is to maximize system wide throughput for both applications. Figure 7 shows the change of their throughput during RL online learning. The incoming workload changes every 30 minutes. We randomly selected a time period with three different workloads and evaluated VCONF's ability in system wide performance optimization. For TPC-W1, the traffic mix changes were: ordering, ordering and shopping. To form different resource demands, TPC-W2 ran shopping, browsing and browsing mixes correspondingly. From the figure, we can see that both applications suffered performance degradation when the workload changed at the $30th$ and $60th$ time points. This is partially due to unbalanced VM configuration caused by traffic dynamics. On the other hand, the RL agent was able to correct unbalanced configurations within a few steps. For example, TPC-W1's throughput dropped to 2000 during the second workload change. The RL agent brought the performance back and maintained the throughput around 7000 within 7 steps. Note that the policy employed by the RL agent is not guaranteed to be an optimal policy because of the agent's limited interactions with the environment. There is no guarantee that the throughputs for both applications were maximized.

From the $60th$ time point, the two VMs ran browsing and shopping mixes respectively. The resource contention and performance interference between the two VMs are more pronounced under this workload. We examined the effectiveness of RL-based approach by comparing the performance of the derived RL policy with a general trial-and-error method. The method fixes the value of one parameter and tries different settings for another parameter. Figure 8 plots performance of the trial-and-error method. The trend line in the figure is the linear regression of the performance in both VMs. The figure suggests that, on average the VMs running browsing and shopping mix can achieve a maximum throughput of 4500 and 6500 concurrent requests. Compared with the trial-and-error, the RL-based approach brought the throughput of both applications to around 5000

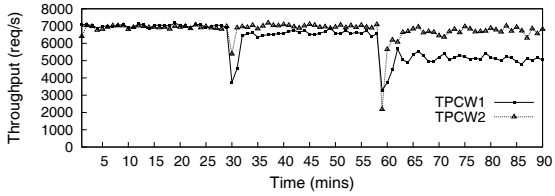


Figure 7: VCONF performance with two TPC-W instances.

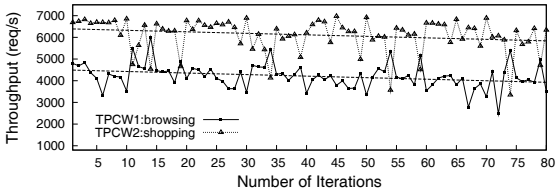


Figure 8: Performance of trial-and-error.

and 7000 respectively. More importantly, the RL approach automatically directed the resource allocation towards target configurations without any human intervention.

We define the difference between the system throughput under current configuration and the throughput achieved in the target configuration as the performance deviation. Figure 9 plots the performance deviation in each configuration step with a 95% confidence interval. The figure suggests that starting from arbitrary configurations, the RL agent should be able to continuously improve the system throughput at each configuration step. On average, the system wide throughput would stabilize within 7 configuration steps.

5.5 Model-based RL in VM Auto-configuration

In previous experiments, we showed the effectiveness of RL in small scale problems. VCONF was able to find the optimal configuration for a single application. In the multiple-application problem, a policy generated by RL using previous collected traces achieved good results in optimizing system wide performance. Statistical results showed that the RL approach would continuously improve the configuration step by step and reach the target configuration within a small number of iterations. However, as the VM configuration problem scales up, the state space grows dramatically.

Standard RL approaches depends critically on the experiences with the environment to generate policies. Unfortunately, the number of experiences needed for an optimal RL policy grows with the state space. The pre-defined policy used to collect experiences is likely to converge to sub-optimal policies due to the relatively data scarcity in the huge state space. Model-based RL provides a solution to the problem by providing a generalization over the collected experiences. By training a model that captures the relationship between state-action pairs and the rewards collected, the RL agent is able to simulate experiences for unseen state-action pairs. Then, the simulated experiences are used to update the Q values. The performance of the model-based RL approach relies on the accuracy of the trained model. Policies for experience collection should be carefully designed in order to record representative sample data.

In the last experiment, we scaled the previous controlled VM configuration problem in two dimensions. We consolidated three benchmarks, TPC-W, TPC-C and SPECweb,

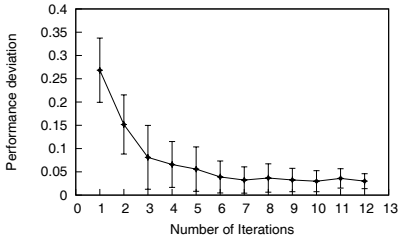


Figure 9: Performance deviation during reconfiguration.

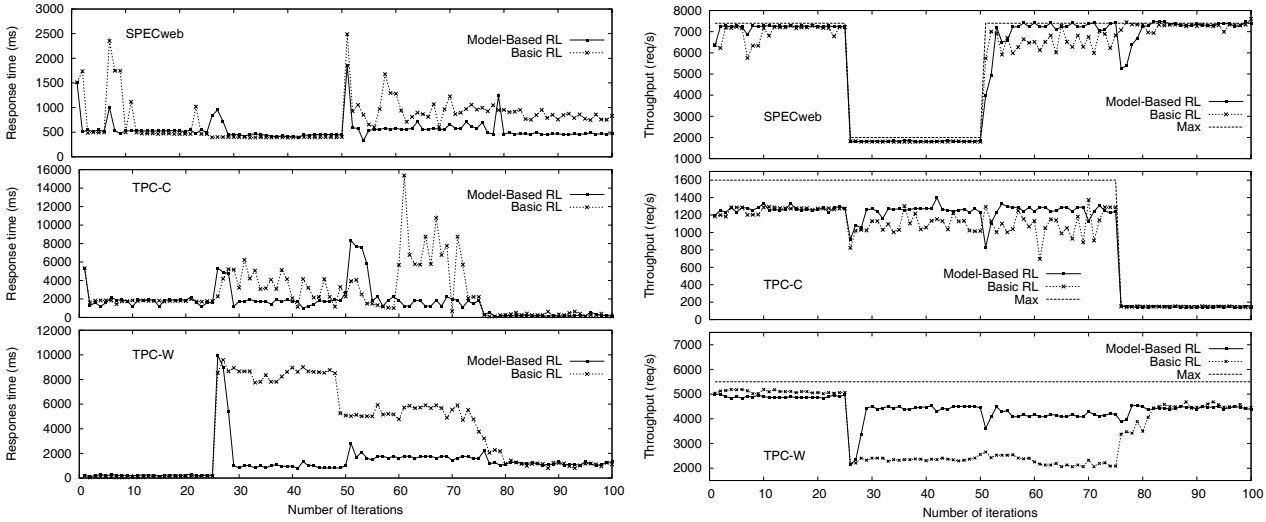
with heterogeneous resource profiles in the virtual server. TPC-W is primary CPU-intensive while TPC-C requires a large amount of disk I/Os. The execution of requests in SPECweb involves processor and network I/O for dynamic content generation and static image serving. The VM resources in consideration were the virtual CPU number, scheduler credit and memory size. We defined different workloads with varying resource demands and tabularized them in Table 1.

All VMs were initially set to an identical configuration: 1.5GB memory, 4 virtual CPUs and a credit of 256. We designed the policy for experience collection as a traversal in a pre-defined resource configuration set. The set contains representative combinations of the allocations uniformly scattered in the state space. The NN models were trained with a learning rate of 0.0001 and a momentum of 0.1. Four models were trained for different workloads. A second layer NN generalization was used as the Q function approximator and its learning process is listed in Algorithm 2. The time required to train a NN model from an arbitrary neural network is approximately 10 minutes. When updated incrementally, the training time reduces to around 1 minute. In order to fit the updates of the NN model and the Q approximator between each interval, we limited the update of the NN model and the Q approximator to 50 iterations and 100 sweeps respectively, which resulted in a 50-second compute time. We compared model-based RL approach with the basic table-base RL algorithm. To be fair comparison, the basic RL's Q tables for different workloads were initialized by the NN-based Q approximators. During online learning, VCONF identifies workload changes and recommends corresponding models and Q tables to model-based RL agent and basic RL agent. Both the model-based RL agent and the basic RL agent were started with the same VM initial configuration.

We randomly selected a time period with four different workloads. Figure 10 shows the performance of VCONF with respect to response time and throughput. The "Max" plot is the reference throughput for each application. The reference value was obtained when each application ran alone on the virtual server with sufficient resources. Due to VM interferences and possible inappropriate configurations, the throughput for each application is less than the reference value. Model-based RL approach outperforms basic RL in that it achieved a higher throughput and lower response time during online learning. The model-based RL was able to adapt to workload changes well. It improved the application throughput by 20%-100% over the basic RL approach in different applications. In addition, model-based approach was more stable sticking with the "best" configuration during the same workload. The basic RL agent wagered between several configurations some of which incurs considerable performance penalty.

Table 1: Workload settings.

	TPC-W	TPC-C	SPECweb
workload-0	600 browsing clients	50 warehouses, 10 terminals	800 banking clients
workload-1	600 ordering clients	50 warehouses, 10 terminals	800 banking clients
workload-2	600 browsing clients	50 warehouses, 1 terminal	800 banking clients
workload-3	600 browsing clients	50 warehouses, 10 terminals	200 banking clients



(a) Response Time (b) Throughput

Figure 10: Performance of VCONF with heterogeneous applications.

The advantage of model-based RL approach over basic RL is due to the model's ability generalizing the environmental changes. In another word, the model-based approach is more data efficient [2] that a change in the environment can spread to other state-action pairs because they are co-related within the model. The basic RL approach stores Q values separately without interactions, then an environmental change can only influence the agent's decision when the affected Q value entry is visited next time.

6. RELATED WORK

Early work in autonomic computing [1] aimed to develop computer systems of self-management to overcome the rapidly growing complexity of system management. Recent work often focuses on the design and implementation of self-healing, self-optimization and self-configuration systems.

Self-healing systems automatically discover and correct faults. In [8, 33], Cohen, et al. suggested to use a machine learning model to generate system signatures for the purpose of performance problem diagnosis. They correlated system low-level metrics to high-level performance states. By monitoring sensor readings, the statistical approach was able to narrow down possible faults. In [17], we defined a performance index to measure the system health based on hardware performance counters. A bayesian network model was assumed to automatically map hardware events to system overload state. Studies in [5] reduced downtime of J2EE applications by rapidly and automatically recovering from transient and intermittent software failures, without requiring application modifications.

Self-optimization systems automatically monitor and control resources to ensure optimal performance with respect to defined requirements. Control theory has recently been applied in computer systems for performance optimization. Similar self-tuning adaptive controller were designed in [12, 13] for multi-tier web sites and storage systems. There are other efforts towards automatically allocating resources in a fine grain to individual requests using fuzzy control [30, 14].

Self-configuration systems automatically adapt software parameters, hardware resources for the purpose of correct function or better performance. In [19], *AutoBash* leveraged causal tracking support in Linux to automate tedious parts of fixing a mis-configuration. Chronus in [31] used checkpoint and rollback for configuration management to diagnose kernel bugs. Other work focused on automatically configuring a software of a physical system to a better configuration for performance. Our work addresses the problem in the granularity of VMs with actual hardware resources.

Different from the above approaches in designing self-managed system, RL offers tremendous potential benefits in autonomic computing. Although practical issues exist using RL in real-world applications, efforts has been made to apply RL in computer systems. In [22, 24], the authors used hybrid RL algorithms to optimize server allocation in a server farm. RL was also applied to balance power-performance of computing systems [23] and automate application parameter tuning [4]. Above works defined state spaces on the discretization of a single metric, which is not easily extended to a higher dimension. The state space in our work was defined on three dimensions. Ipek et al. designed a self-optimizing memory controller [11] and implemented the RL in a sim-

ulator. Our deployment of RL algorithms in web hosting VMs poses more challenges in efficient design.

The emergence of virtual machines provides an alternate approach to provision and managing resources . VMs become the target for resources allocation and configuration. In [18], an VM advisor automatically configured VMs for database workloads. The advisor required domain knowledge. Padala et al. applied classical control theory to adjust a single resource each VM [16]. This work is closely related to ours in that it employed a black-box approach which requires not no domain knowledge. However, the authors' approach was limited to one configurable resource and non-work-conserving sharing of the resource. This assumption does not hold for heterogeneous VMs competing for multiple resources. Without the assumption, their single-input single-output classical control theory is not applicable to a more complex domain. To the best of our knowledge, our work is the first to use RL-based methods in VM auto-configuration with multiple resources. Our approach has two major benefits: First, we do not assume any domain knowledge as in [18] and applies to a wider range of VM applications. Second, our model-based approach generates reasonable good policies with limited interactions even in a scaled up problem.

7. CONCLUSIONS AND DISCUSSIONS

In this work, we presented VCONF, a RL-based agent for virtual machine auto-configuration. VCONF automates the VM reconfiguration process by generating policies learned from iterations with the environment. Experiments on Xen VMs with typical server applications showed VCONF's optimality in controlled problems and good adaptability and scalability in a cloud computing testbed. In the presence of workload dynamics, VCONF was able to adapt to a good configuration within 7 steps and showed 20% to 100% throughput improvement over basic RL methods.

Nevertheless, there are several limitations this work. First, the quality of the samples used in model training affects the quality of the policies. We consider a uniform distribution of the samples over the state space as representative. In a system with different VM behaviors, the strategy for sample collection may need specific design for the environment in consideration. Second, in hosted environment, VMs usually share network interfaces and the access to centralized storage service. Network and disk bandwidth should also be considered. Another important resource is the shared L2 cache space in modern chip-multiprocessors. In a system with hundreds of CPU cores, L2 cache may be the first-class resource to be considered in VM configuration.

Acknowledgement

We would like to thank the anonymous reviewers for their constructive comments. This research was supported in part by U.S. NSF grants CCF-0611750, DMS-0624849, CNS-0702488, and CRI-0708232.

8. REFERENCES

[1] http://www.research.ibm.com/autonomic.
[2] C. G. Atkeson and J. C. Santamar'ia. A comparison of direct and model-based reinforcement learning. In *In ICRA*, 1997.
[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP*, 2003.
[4] X. Bu, J. Rao, C.-Z. Xu. A reinforcement learning approach to online web systems auto-configuration. In *ICDCS*, 2009.
[5] G. Candea, E. Kiciman, S. Kawamoto, and A. Fox. Autonomous recovery in componentized internet applications. *Cluster Computing*, 2006.
[6] J. P. Cassaza, M. Greenfield, and K. shi. Redefining server performance characterization for virtualization benchmarking. In *Intel technology Journal*, 2006.
[7] 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.
[8] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *SOSP*, 2005.
[9] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in xen. In *Middleware*, 2006.
[10] Hyper-V server. http://www.microsoft.com/servers/hyper-v-server.
[11] E. Ipek, O. Mutlu, J. F. Martinez, and R. Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *ISCA*, 2008.
[12] A. Kamra, V. Misra, and E. M. Nahum. Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites. In *IWQoS*, 2004.
[13] M. Karlsson, C. T. Karamanolis, and X. Zhu. Triage: performance isolation and differentiation for storage systems. In *IWQoS*, 2004.
[14] 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*, 2003.
[15] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling i/o in virtual machine monitors. In *VEE*, 2008.
[16] 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.
[17] J. Rao and C.-Z. Xu. Online measurement the capacity of multi-tier websites using hardware performance counters. In *ICDCS*, 2008.
[18] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In *SIGMOD*, 2008.
[19] Y.-Y. Su, M. Attariyan, and J. Flinn. Autobash: improving configuration management with operating system causality analysis. In *SOSP*, 2007.
[20] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*. 1996.
[21] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
[22] G. Tesauro. Online resource allocation using decompositional reinforcement learning. In *AAAI*, 2005.
[23] 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*. 2007.
[24] 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.
[25] The SPECweb benchmark. http://www.spec.org/web2005.
[26] http://www.tpc.org/tpcw.
[27] http://www.tpc.org/tpcc.
[28] VMware. http://www.vmware.com.
[29] VMware VMmark. http://www.vmware.com/products/vmmark.
[30] J. Wei and C.-Z. Xu. A self-tuning fuzzy control approach for end-to-end qos guarantees in web servers. In *IWQoS*, 2005.
[31] A. Whitaker, R. S. Cox, and S. D. Gribble. Configuration debugging as search: Finding the needle in the haystack. In *OSDI*, 2004.
[32] J. Wildstrom, P. Stone, and E. Witchel. Carve: A cognitive agent for resource value estimation. In *ICAC*, 2008.
[33] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox. Ensembles of models for automated diagnosis of system performance problems. In *DSN*, 2005.