2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

Coordinated VM Resizing and Server Tuning: Throughput, Power Efficiency and Scalability

Yanfei Guo and Xiaobo Zhou
Department of Computer Science
University of Colorado, Colorado Springs, USA
Email addresses: {yguo, xzhou}@uccs.edu

***Abstract*—Performance control and power management in virtualized machines (VM) are two major research issues in modern data centers. They are challenging due to complexities of hosted Internet applications, high dynamics in workloads and the shared virtualized infrastructure. Obtaining a model among VM capacity, server configuration, performance and power consumption is a very hard problem even for just one application. In this paper, we propose and develop GARL, a genetic algorithm with multi-agent reinforcement learning approach for coordinated VM resizing and server tuning. In GARL, model-independent reinforcement learning agents generate VM capacity and server configuration options and the genetic algorithm evaluates different combinations of those options for maximizing a global utilization function of system throughput and power efficiency. The multi-agent design makes GARL a scalable approach, which is important as more and more applications are hosted in data centers using cloud services. We build a testbed in a prototype data center and deploy multiple RUBiS benchmark applications. We apply a power budget in the testbed and observe superior system throughput and power efficiency of GARL. Experimental results also find that GARL significantly outperforms a representative reinforcement learning based approach in performance control. GARL shows better scalability when compared to a centralized approach.**

I. Introduction

Infrastructure-as-a-Service (IaaS) is one important form of cloud computing. In IaaS, users are provided with on-demand virtual machines (VMs). Besides the benefit to users, a key benefit of IaaS to the cloud provider is the high resource utilization in data centers. Due to the high flexibility in adjusting VM capacity for resource utilization efficiency, cloud providers can consolidate Internet applications into a fewer number of physical servers by multiplexing among the workload patterns of multiple VMs [14].

Performance control and power management in VMs are two major research issues in modern data centers provisioning cloud services. From the perspective of cloud providers, the objective is to meet the service level agreement (SLA) of individual applications, to improve the overall system throughput and to optimize resource utilization in the data center [17]. There are significant challenges. First, the user perceived performance of hosted applications relies on effective management of VM capacity configurations. It is non-trivial because of the complexity of applications, shared underlying hardware infrastructure, and the resulted performance correlation and interference among applications [10], [15], [16].

Second, Internet server applications have many configurable parameters that related to server concurrency level, worker process generating and network link alive time. They are very important to the performance of applications and to the resource utilization of the underlying computer system. An improper configuration can harm the application performance and resource utilization. However, server parameter tuning is a challenging task because of the complexities in workloads, parameter dependencies and variances in VM capacities. In order to improve the overall system throughput and to optimize resource utilization in the data center, those parameters must be tuned to match the workloads and VM capacities.

Third, power budget has been widely adopted for power and thermal management in data centers. It helps cloud providers to reduce the running cost. But enforcing power budget and improving system performance can be in conflict. Finding a good trade-off between power consumption and system performance is crucial for power management in data centers. Power efficiency is a popular metric for evaluation of different trade-offs [4], [6], [8]. But the power consumption of VMs is highly related to their resource utilizations, workload volumes and application characteristics [18], [26].

Moreover, as cloud services become popular, more and more applications are hosted in data centers. This imposes the scalability challenge to the performance control and power management. Obtaining an accurate model among VM capacity, server configuration, performance and power consumption is a very hard problem even for just one application due to the complexity of application characteristics, workload dynamics and burstiness.

In this paper, we propose and develop GARL, a model-independent and scalable approach for coordinated VM resizing and server parameter tuning. GARL targets a common scenario that multiple Internet applications are hosted in a shared resource pool with a certain power budget. It aims to improve the throughput and power efficiency of applications while meeting the resource constraints and the power budget in an IaaS environment.

GARL integrates the strengths of genetic algorithms and multi-agent reinforcement learning. Due to the model independency, reinforcement learning has been used in performance and power control in data centers [2], [20], [21], [23]. It is effective in a complex environment lacking an accurate performance model. Recently, Rao *et al.* proposed iBalloon [21],

1526-7539/12 $26.00 © 2012 IEEE
DOI 10.1109/MASCOTS.2012.41

a multi-agent reinforcement learning approach for improving performance of multiple applications via VM resizing. iBalloon improved application performance and achieved near-optimal resource allocation within an acceptable amount of time. However, iBalloon did not consider the coordination between VM resizing and server parameter tuning. Its multi-agent reinforcement learning approach is lacking of explicit coordination among multiple applications. Power efficiency is also not considered in iBalloon. These issues have significantly restricted the ability in improving both application performance and power efficiency in an IaaS environment.

In GARL, each application has two corresponding reinforcement learning agents, one performance agent and one power agent. The performance agent generates VM capacity and server configuration options. The power agent generates the prediction on power consumption. GARL's genetic algorithm based coordinator evaluates different combinations of those options for maximizing a global utilization function of system throughput and power efficiency. Because the state space size will increase exponentially when the number and complexity of applications increase, the traditional reinforcement learning based on Q-Table is poor in scalability. We employ function approximation to represent the Q-Table as a linear function to improve the scalability.

We implement GARL in a testbed of virtualized server cluster hosting multiple RUBiS benchmark applications. The testbed consists of 5 Dell PowerEdge R610 servers and 2 Dell PowerEdge R810 servers using VMware vSphere 5.0. Experimental results demonstrate that GARL outperforms a representative multi-agent reinforcement learning based approach (MRL) used in iBalloon [21] in system throughput and power efficiency. GARL achieves $5\% - 16\%$ higher effective system throughput than MRL for different combinations of RUBiS applications. It significantly outperforms MRL on power efficiency. Results also show that GARL has good scalability when the number of applications increases.

The main contributions of our work are:

- We propose to coordinate VM auto-configuration with server parameter tuning to maximize the system throughput and power efficiency in an IaaS environment.
- We develop a model-independent approach that integrates the strengths of genetic algorithms and reinforcement learning. It is effective in a complex environment lacking an accurate model among VM capacity, server configuration, performance and power consumption.
- We enhance the approach with a multi-agent design for the scalability in an IaaS environment.
- We build a testbed in a prototype data center and evaluate the new approach with benchmark applications.

In the following, Section II discusses related work. Section III describes the system design of GARL. Section IV presents the reinforcement learning agents of GARL. Section V describes the genetic algorithm for the coordination of VM resizing and server tuning in GARL. Section VI gives the testbed implementation. Section VII presents the experimental results and analysis. Section VIII concludes the paper.

II. Related Work

VM capacity planning is crucial to application performance assurance. Recent studies focused on improving user-perceived performance through automated VM resizing [19], [20], [21]. VCONF is a reinforcement learning based approach for VM auto-configuration [20]. It identifies the performance interference between different VMs and the sequence dependent of VM performance as the major challenges. It achieved good performance for TPC-W, TPC-C, and SPECjbb benchmark applications. VCONF is based on a centralized design of reinforcement learning. It has the scalability problem when being applied to multiple complex applications.

Recently, iBalloon is proposed to improve the average response time and throughput for multiple applications using VM resizing [21]. It is a multi-agent reinforcement learning approach, in which each agent is associated with one application. The major benefit of the decentralized design is the scalability. iBalloon generalized the reinforcement learning by function approximation. It reduced the complexity of reinforcement learning agent. But iBalloon is lacking of explicit coordination among multiple applications. This restricts its capability of improving application performance. There is no coordination between VM resizing and server parameter tuning. It does not address the important power efficiency issue.

A few recent studies focused on automated server parameter tuning for multi-tier server systems [2], [7]. In our recent study [7], we proposed to use the effective system throughput as the primary performance metric for multi-tier Internet applications. We employed a neural fuzzy control based approach to achieve automated and agile server parameter tuning and used it to improve the application performance. Experimental results showed that server parameter tuning can improve application performance without increasing the VM capacity. However, the tuning approach has a limitation of fixed VM capacity. Due to the correlation between VM capacity and server configuration, it is necessary to perform server parameter tuning and VM resizing in a coordinated manner.

In an IaaS platform, multiple complex applications are hosted in a virtualized and shared server infrastructure. Power control is a complex issue due to the correlation between the power consumption and the resulted performance. Performance-oriented approaches aim to guarantee a performance target and do not have explicit control over power consumption [3], [9], [29]. Power-oriented approaches aim to enforce power budget and disregard the SLA of hosted applications [13], [18], [27], [28]. Power efficiency is used to achieve a trade-off between performance gain and power consumption [4], [6], [8].

Recent studies focused on coordinated power and performance control. pMapper [25] tackles power-cost trade-offs under a fixed performance constraints. vManage [8] performs VM placement to save power without degrading performance. Co-Con [28] is a two-level control architecture for power

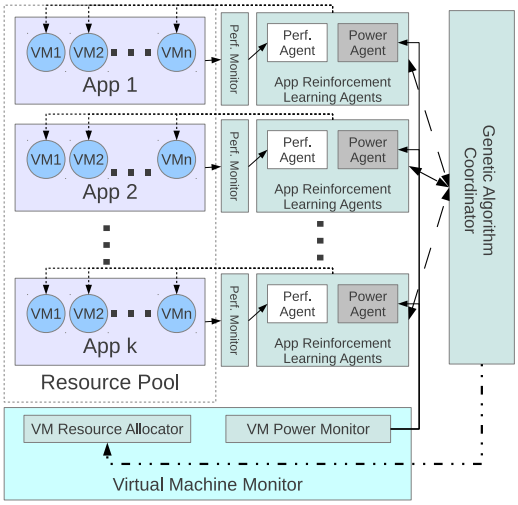


Fig. 1. The architecture of GARL.

and performance coordination in virtualized server clusters. It gives a higher priority to enforcing power budget and set performance as a secondary goal. vPnP [5] coordinates power and performance in virtualized servers using utility function optimization. It provides the flexibility to choose various tradeoffs between power and performance. PERFUME [12] employed a fuzzy MIMO control technique to providing accuracy and stability for joint power and performance control under highly dynamic workloads.

In GARL, we integrate the strengths of genetic algorithm and multi-agent reinforcement learning to achieve coordination between VM resizing and server parameter tuning. The resulted approach can significantly improve the effective system throughput, power efficiency, and scalability.

III. GARL: System Design

The joint performance and power management imposes three major requirements to our new approach. First, the approach must perform VM resizing and server parameter tuning coordinately to achieve best performance in an IaaS platform. Second, there should be coordination among applications to find the best tradeoff for improving the overall system throughput and power efficiency. Third, the approach has to be scalable so that it can support many complex applications.

A. Architecture of GARL

Figure 1 illustrates the architecture of GARL. The system under control is a virtualized server cluster hosting multiple multi-tier Internet applications. Each application consists of multiple VMs. All applications share the same resource pool. Each application is associated with one performance monitor, one reinforcement learning based performance agent and one reinforcement learning based power agent.

By using the VMware's Intelligent Power Management Interface, the VM power monitor periodically measures the average power consumption of the server system at the VM level and feeds the power consumption data into the power agent. The performance agent obtains the performance data via the performance monitor of each application. The reinforcement learning agents generate VM capacity and server configuration options and the multi-agent coordinator uses genetic algorithm to evaluate different combinations of those options for maximizing a global utilization function of system performance and power efficiency.

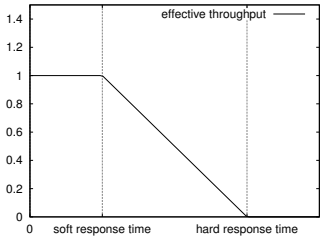


Fig. 2. Linear decaying function for effective system throughput definition.

B. Effective System Throughput

From the perspective of cloud providers, the objective is to meet the SLA of individual applications but also to improve the overall system throughput and to optimize resource utilization in the data center. Effective system throughput is defined as the number of requests that meet the SLA requirement on the application response time [7].

We use a SLA with two response time bounds, hard response time and soft response time. The effective throughput is the number of requests that are processed within the SLA time bounds. If a request is processed between the hard and soft response time bounds, its effective throughput is measured according to a utility decaying function. Figure 2 illustrates the example of a linear function based effective throughput. We define the normalized effective throughput as the ratio of the effective throughput to the total number of incoming requests.

IV. GARL: Reinforcement Learning Agents

Reinforcement learning is a process of learning through interactions with an external environment. The VM resizing and server parameter tuning problem can be formulated as a finite Markov decision process. It consists of a set of states and several actions for each state. During the transition of each state, the learning agent perceives the reward defined by a reward function $r(s, a)$. The goal of the reinforcement learning agent is to develop the policy $\pi : S \rightarrow A$, which can maximize the cumulative rewards through iterative trial-and-error interactions.

For each application, the state space S is defined as the set of possible resource allocations and server parameter values for each VM. For an application that has n VMs, the state space (S) is represented as a collection of state vectors (s):

$$s = [R_{11}, R_{12}, \cdots, R_{ni}, P_{11}, P_{12}, \cdots, P_{nj}].$$

The elements in the state vector are resource allocations and server parameters, in which i and j are the number

of the resource types and the number of server parameters, respectively. In this work, we focus on CPU and memory resources. Note that the hard disk size is online configurable but the I/O bandwidth is not controllable.

The action for each state element is represented as a vector. We define three actions for each state element: keep, increase, and decrease. Hence, an action vector can be $(1,0,0)$, or $(0,1,0)$, or $(0,0,1)$. For example, $(1,0,0)$ means to keep the current value of one state element. The action set (A) is represented as a collection of action vectors (a):

$$a = [a_{R_{11}}, a_{R_{12}}, \cdots, a_{R_{ni}}, a_{P_{11}}, a_{P_{12}}, \cdots, a_{P_{nj}}].$$

A. Performance Agent

We use a Q-Learning agent to control the effective system throughput. The Q-Learning agent uses a Q-Table to determine the action choice on each state. The Q-Table stores the Q-Value for each state-action pair. The learning process will continuously update the Q-Values based on the reward it receives. The reward function of an action a on state s in k_{th} time slot is defined using the effective throughput (ET) and the normalized effective throughput (NET):

$$r(s_k, a_k) = \beta |ET_{s_k,a_k} - ET_{s_{k-1},a_{k-1}}| \tag{1}$$

where $\beta = NET_{s_k,a_k} - NET_{s_{k-1},a_{k-1}}$. It uses the change in normalized effective throughput β as the correction factor. For example, if the effective throughput has increased significantly and the normalized effective throughput has almost no change, it implies that the change in the effective throughput could be just due to the variance of the number of incoming requests. This phenomenon indicates that the current server system configuration is favorable for the present workload and no significant update should be applied to the reinforcement learning. With the reward function, the Q-Value of an action a on state vector s in k_{th} time slot is updated as $Q(s_k, a_k)$. It is refined by

$$\begin{aligned} Q(s_k, a_k) = Q(s_k, a_k) + \varepsilon [r(s_k, a_k) - Q(s_k, a_k) \\ + \gamma Q(s_{k+1}, a_{k+1})] \end{aligned} \tag{2}$$

where ε is the learning rate and γ is the discount rate to guarantee that the accumulated reward converges in continuing tasks. We apply a variable learning rate for fast convergence. The learning rate for different state-action pair is defined as

$$\varepsilon = \frac{\mu}{N(s,a)} \tag{3}$$

where μ is a given constant and $N(s,a)$ is the number of times that state-action pair (s,a) has been visited. The future action a_{k+1} is determined by the multi-agent coordinator.

B. Power Agent

For the power consumption of each application, we use a temporal-difference agent to learn the model between resource allocation and power consumption. The power agent observes the power consumptions for all state in the state space (S). The updating function of power agent is defined as

$$U(s_{k-1}) = U(s_{k-1}) + \varepsilon [p(s_{k-1}) + \gamma U(s_k) - U(s_{k-1})] \tag{4}$$

where ε is the learning rate, γ is the discount rate, and $p(s_{k-1})$ is the change in power consumption. It will update the resource-power model using the measure power consumption. Through reinforcement learning, the resource-power model converges and reflects the relations between resource allocation and power consumption.

C. Generalization of Reinforcement Learning

The state space consists of multiple discrete states. Therefore, the reachable configurations are limited due to the sparsity of state space and the predefined adjustment value for each tuning action. Using a small adjustment value improves the performance of reinforcement learning and increases the number of states in the state space.

However, this approach only works in small-scale problems that do not have a lot of states. For the coordinated VM resizing and server parameter tuning problem in complex multi-tier applications, the size of state space increases exponentially with the increasing number of applications. This will significantly reduce the converging speed and make reinforcement learning poor in scalability or even infeasible.

One way to handling such a problem is function approximation [22]. The function approximator use a Q-Function to calculate the Q-Value. It breaks the limitation of the predefined adjustment value and allows the learning agent to calculate the Q-Value for any state-action pair. The Q-Function can be defined as

$$\begin{aligned} Q_\theta(s,a) = \theta_0 + \theta_1 R_{11} + \theta_2 R_{12} + \cdots + \theta_{in} R_{ni} \\ + \theta_{in+1} P_{11} + \theta_{in+2} P_{12} + \cdots + \theta_{(i+j)n} P_{nj} \\ + \theta_{(i+j)n+1} a_{R_{11}} + \theta_{(i+j)n+2} a_{R_{12}} + \cdots + \theta_{(2i+j)n} a_{R_{ni}} \\ + \theta_{(2i+j)n+1} a_{P_{11}} + \theta_{(2i+j)n+2} a_{P_{12}} + \cdots + \theta_{2(i+j)n} a_{P_{nj}} \end{aligned} \tag{5}$$

where the values for the parameters $\theta_0, \theta_1, \theta_2, \cdots, \theta_{2(i+j)n}$ are learned through the reinforcement learning algorithm.

We use an online learning algorithm to update the parameters in each time slot. The most common approach is using the Widrow-Hoff rule [22]. The Widrow-Hoff rule calculates the error as the squared difference between successive states. To move the parameter in the direction of decreasing the error, the parameter should be updated using

$$\begin{aligned} \theta_p &= \theta_p - \varepsilon \frac{\partial E(s,a)}{\partial \theta_i} \\ &= \theta_p + \varepsilon [r(s_t, a_t) - Q_\theta(s_t, a_t) \\ &\quad + \gamma Q_\theta(s_{t+1}, a_{t+1})] \frac{\partial Q_\theta(s_t, a_t)}{\partial \theta_i}. \end{aligned} \tag{6}$$

Similarly, the utility function of temporal-difference agent is represented as

$$\begin{aligned} U_\varphi(s) = \varphi_0 + \varphi_1 R_{11} + \varphi_2 R_{12} + \cdots + \varphi_{in} R_{ni} \\ + \varphi_{in+1} P_{11} + \varphi_{in+2} P_{12} + \cdots + \varphi_{(i+j)n} P_{nj}. \end{aligned} \tag{7}$$

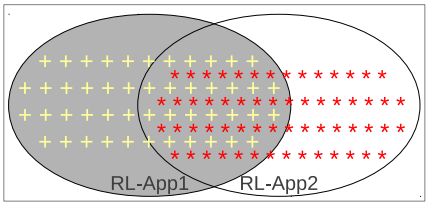


Fig. 3. Solution spaces of multiple reinforcement learning agents.

By using the Widrow-Hoff rule, the update function of parameters $\varphi_0, \varphi_1, \varphi_2, \cdots, \varphi_{(i+j)n}$ is defined by

$$\varphi_q = \varphi_q + \varepsilon[p(s_{t-1} + \gamma U_\varphi(s_t) - U_\varphi(s_{t-1})]\frac{\partial U_\varphi(s_{t-1})}{\partial \varphi_i}. \tag{8}$$

V. GARL: Genetic Algorithms for Coordination of VM Resizing and Server Tuning

A. The Coordination Problem

The objective of GARL is to generate VM configurations and server parameter values for each application that can maximize the effective system throughput and power efficiency while meeting the resource and power constraints. Reinforcement learning is able to find the optimal configuration for each application separately. However, the local optimal configuration for each application is not necessarily able to yield global optimal performance and power efficiency for all application in an IaaS platform.

For example, Figure 3 shows the solution spaces of reinforcement learning agents of two applications. The '+' sign and '*' sign represent the state-action pair candidates of each application, respectively. The overlap of state-action pair spaces is determined by the resource constraint and power budget. Therefore, only the combining configuration options in the overlap area of the two state-action pair spaces can satisfy the global constraints. As the number of application increases, the complexity of searching and finding such a good operation set increases exponentially. Therefore, we need to find an effective way to solving the coordination problem of multiple reinforcement learning agents.

B. The Genetic Algorithm in GARL for Coordination

The genetic algorithm is a search heuristic that is used to generate solutions for optimization and search problem. In the genetic algorithm, we represent one operation set as a chromosome. The structure of the chromosome for m applications is represented as

$$c = \{(s,a)_1, (s,a)_2, \cdots, (s,a)_m\} \tag{9}$$

where each segment of the chromosome represents a configuration option for one application. GARL generates a population with different chromosomes. The population evolves by forming a child population with chromosomes in the parent population. This is motivated by a hope that the new population will be better than the old one. Chromosomes are selected to form new solutions by their fitness - the higher fitness they are the higher chance they will be selected. During each evolution, the genetic algorithm has two major steps: crossover and mutation.

In crossover, the algorithm will pick parent chromosomes form the current population by their fitness, generate child chromosomes by swapping the codes at random locus (point in chromosome). After that, those low fitness chromosomes will be replaced by the new chromosomes. In mutation, the algorithm will select a chromosome and randomly change the code at one locus. Then, it is put back into the population. During each evolution, the crossover and mutation are controlled by the crossover rate and mutation rate respectively.

GARL aims to improve the effective system throughput and power efficiency. Therefore, the fitness function is defined as

$$F(c) = C_1(R)C_2(p) \sum_{i=VM_j \in c} \omega_i \frac{Q_\theta(s,a)_i}{U_\theta(s)_i} \tag{10}$$

where ω_i is the weight of i_{th} application, $C_1(R)$ and $C_2(R)$ are the resource pool constraint and power budget, respectively. The definitions of $C_1(R)$ and $C_2(R)$ are

$$C_1(R) = \begin{cases} R_{max} - \sum_{allVMs} R_i & \text{if } \sum R_i \le R_{max} \\ 0 & \text{if } \sum R_i > R_{max}. \end{cases}$$

$$C_2(p) = \begin{cases} p_{max} - \sum_{allVMs} p_i & \text{if } \sum p_i \le p_{max} \\ 0 & \text{if } \sum p_i > p_{max}. \end{cases}$$

If either the resource allocation constraints or the power budget has been violated, the fitness function will result in zero. The summation part of the fitness function is the weighted effective throughput per power consumption. The heuristic for the fitness is to find the combination of configuration options that achieve the highest fitness function value.

VI. System Implementation

We build our testbed in a university prototype data center, which consists of five Dell PowerEdge R610 servers and two Dell PowerEdge R810 servers. Totally, they have 10 Intel hexa-core Xeon X5650 CPUs, 8 Intel hexa-core E7540 CPUs, and 704 GB memory. Each server is equipped with 2-way Intel quad-core Xeon E5530 CPUs and 32GB memory. The servers are connected with 10 Gbps Ethernet. VMware vSphere 5.0 is used for server virtualization.

As many others in [2], [11], [12], [20], [24], [30], we use RUBiS [1] as the benchmark application in conducting the experiments. RUBiS is an open source multi-tier Internet benchmark application. RUBiS emulates three different categories of workload at different concurrent level. We implement a bursty workload generator for RUBiS benchmark using the approach proposed by *Mi et al.* [16], which changes the think time of each user.

By mixing different architectures, workload mixes and burstiness, we create twelve RUBiS application templates as shown in Table I. Each application template represents different resource demands in CPU and memory. For example, a multi-tier application needs more resources than a single-tier application due to the operating system overhead and

communication overhead of each VM. Different workload mixes result in different resource demand on each tier. A bursty workload uses more resource usage than a stationary workload when processing the same volume of workload.

TABLE I
RUBiS APPLICATION TESTING TEMPLATES.

Name	Architecture	Workload
RUBiS-A1	Multi-tier	Stationary Browsing
RUBiS-A2	Multi-tier	Stationary Selling
RUBiS-A3	Multi-tier	Stationary Bidding
RUBiS-B1	Single-tier	Stationary Browsing
RUBiS-B2	Single-tier	Stationary Selling
RUBiS-B3	Single-tier	Stationary Bidding
RUBiS-C1	Multi-tier	Bursty Browsing
RUBiS-C2	Multi-tier	Bursty Selling
RUBiS-C3	Multi-tier	Bursty Bidding
RUBiS-D1	Single-tier	Bursty Browsing
RUBiS-D2	Single-tier	Bursty Selling
RUBiS-D3	Single-tier	Bursty Bidding

For multi-tier applications, we allocate three VMs for each application, Apache web server in the first, PHP application server in the second, and MYSQL database server in the third. The maximum capacity of all three VMs is the same, which is 1 VCPU (up to 2.66 GHz) and 1 GB memory. For single-tier applications, the web server, the application server, and the database server reside in the same VM. The maximum VM capacity is 2 VCPU (up to 5.32 GHz) and 2 GB memory. All VMs use Ubuntu server 10.04 with Linux kernel 2.6.35. We consider to tune the server parameter *MaxClients* as our previous study [7] found that it dominates the performance impact on server performance.

VII. EVALUATIONS

We first focus on the achieved effective system throughput and power efficiency of multiple applications due to GARL. Then we explore the trade-offs between applications. Finally, we study the scalability of GARL using different number of applications and compare the achieved system throughput.

A. Effective Throughput and Resource Allocation

We use four different applications, RUBiS-A1, RUBiS-A3, RUBiS-D2, and RUBiS-C3. Initially their VMs are set to half of the maximum capacity. It satisfies the demand of the initial workload of each application. The resource pool is configured as 20 GHz in CPU and 8 GB in memory. The power budget of the resource pool is set at 150 Watt. The control interval is 30 seconds and the experiment lasts for 2 hours. The experiment is separated into four stages. At the 60_{th}, 120_{th}, and 180_{th} control intervals, workload of RUBiS-A1, RUBiS-A3, and RUBiS-D2 is increased from 1000 to 2000 concurrent users, respectively. We assign the same weight to each application's throughput. For comparison, we also implement a multi-agent reinforcement learning (MRL) approach that was used in iBalloon [21]. We measure the reference effective system throughput, power efficiency and resource allocations of the chosen applications without a resource constraint and power budget. It represents the best effective system throughput and the resource demands of those applications.

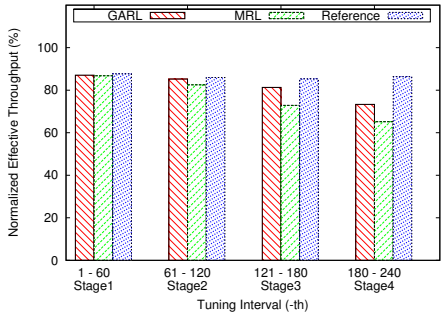


Fig. 4. Normalized effective system throughput comparison.

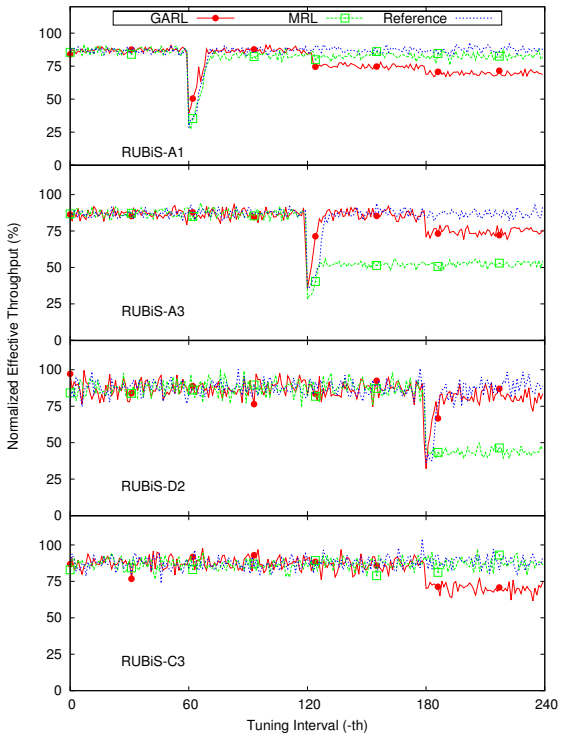
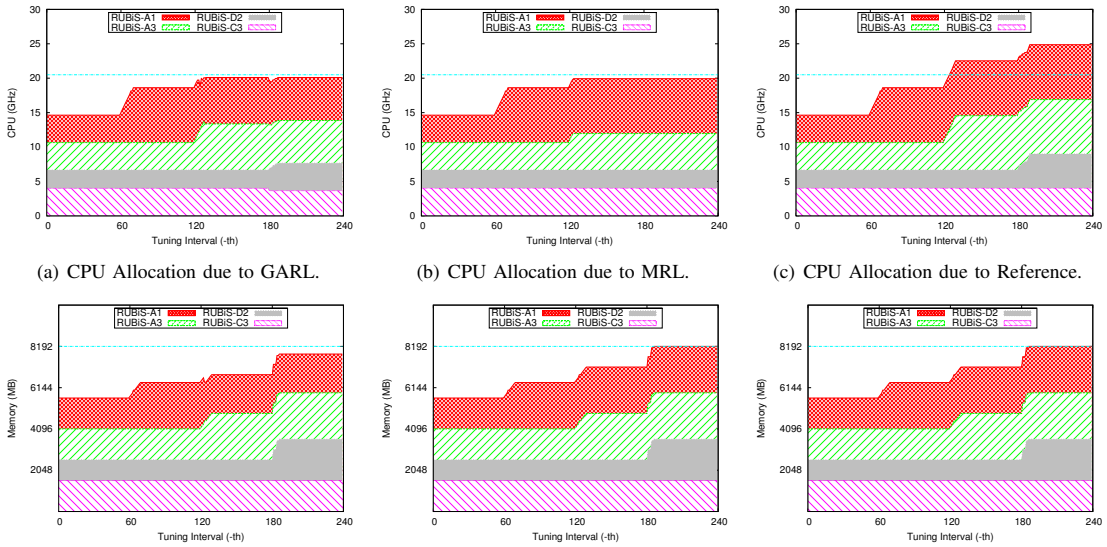


Fig. 5. Performance trace of different approaches on four RUBiS applications.

Figure 4 depicts the normalized effective system throughput of all four applications. In stage 1 and stage 2, the resource pool has sufficient resources to satisfy the increased demand. Therefore, GARL and MRL achieve similar effective system throughput. In stage 3 and stage 4, there is not enough resource left in the resource pool to satisfy the increased demand of all applications. GARL outperforms MRL in achieving the normalized effective system throughput by 12% and 13% more respectively in the two stages by coordinating the VM resizing and server parameter tuning of multiple applications.

Figure 5 illustrates more detailed performance trace of



(a) CPU Allocation due to GARL. (b) CPU Allocation due to MRL. (c) CPU Allocation due to Reference.

(d) Memory Allocation due to GARL. (e) Memory Allocation due to MRL. (f) Memory Allocation due to Reference.

Fig. 6. Traces of CPU and memory allocations using GARL and MRL.

GARL, MRL and the reference performance of four applications. The performance degradation occurs at a stage when each workload volume increases. Both GARL and MRL show the adaptiveness to the workload changes, but GARL significantly outperforms MRL. In stage 3 and stage 4, GARL achieves 49% and 82% more effective system throughput of applications RUBiS-A3 and RUBiS-D2 than MRL does. Due to the coordination among applications, GARL improves the effective throughput of RUBiS-A3 and RUBiS-D2 significantly with slightly performance degradation of RUBiS-A1 and RUBiS-C3. It significantly improves the overall system throughput. On the contrast, MRL is lacking of such a coordinated VM resizing and server parameter tuning. Therefore, the effective system throughput due to MRL is lower.

Figure 6 shows the CPU and memory allocations due to GARL and MRL. Figures 6(a) and 6(b) show that in stage 1 and stage 2, GARL and MRL have very similar CPU allocations. In stage 3, the increased demand of CPU resource for RUBiS-A3 is higher than the available CPU capacity of the resource pool. Both GARL and MRL allocate the available CPU resource to RUBiS-A3. After reaching CPU constraint of the resource pool, GARL reduces the CPU allocation of RUBiS-A1 and reallocates it to RUBiS-A3. Doing so, it improves the effective throughput of RUBiS-A3 by 49% by only sacrificing less than 10% throughput of RUBiS-A1. In stage 4, GARL reduces the CPU allocations RUBiS-A1, RUBiS-A3, and RUBiS-C3 to increase the CPU allocation of RUBiS-D2. It improves the effective throughput of RUBiS-D2 by 82%, and importantly it improves the overall system throughput by 13%. Through the coordination among applications, GARL is able to make favorable trade-off between applications for maximizing the effective system throughput. MRL is not able to coordinate the resource allocation of the applications in such a self-optimizing manner, which results in lower overall effective system throughput.

Figure 6(d) and 6(e) show the memory allocations due to GARL and MRL. In stage 3, GARL reduces the memory allocation of RUBiS-A1 when reducing the CPU allocation. This is because the memory demand decreases when the CPU allocation is reduced. By doing this, GARL avoids resource wastage and improves power efficiency. However, MRL does not have the capability of self-managing the resource allocation for power efficiency.

Figures 6(c) and 6(f) show the reference demands on CPU and memory of those four application. Results show that GARL is able to achieve near to the optimal performance with respect to the achieved effective system throughput, but also meet the resource and power constraints. Note that for the reference approach, the CPU demands in stage 3 and stage 4 are much higher than the resource pool's total CPU capacity.

B. Power Efficiency and Resource Allocation

We evaluate the power efficiency of the GARL and MRL with respect to the effective system throughput per watt (TPW). Figure 7 plots the TPW of GARL, MRL, and Reference. In stage 1 and stage 2, the resource pool has sufficient available resources to satisfy the increased demand. Both GARL and MRL achieve similar TPWs. In stage 3 and 4, GARL achieves 15% and 20% higher TPW than MRL does. This is because RUBiS-A3 and RUBiS-D2 have significant performance degradation due to MRL. One interest phenomenon is that GARL can achieve higher TPW than the reference value. Although GARL achieves slightly lower effective system throughput to the reference, it uses much less resource than the reference. Therefore, we observe that the TPW of GARL is 3.2% higher than the reference in stage 3.

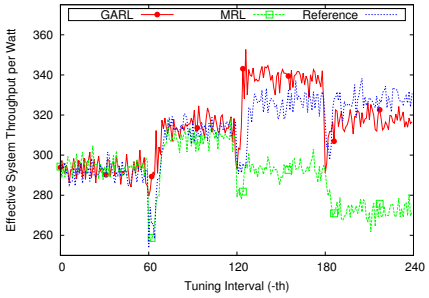


Fig. 7. Power efficiency comparison of of GARL and MRL.

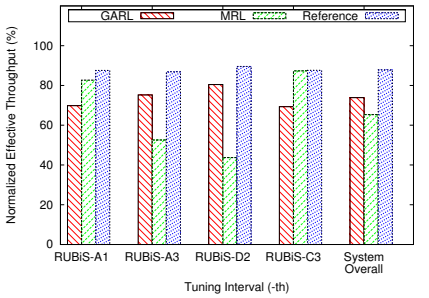


Fig. 8. Performance comparison of applications due to GARL and MRL.

As the performance degradation increases in stage 4, the TPW due to GARL is lower than the reference.

We conclude that the differences in effective system throughput and power efficiency between GARL and MRL are mainly because MRL is lacking of the coordination among multiple applications. Without the coordination, it is infeasible for MRL to optimize its actions for the maximized effective system throughput and the power efficiency.

C. Trade-offs among Multiple Applications

Figure 8 shows the normalized effective throughput of individual applications and the system overall due to GARL and MRL. It shows that the effective throughput of RUBiS-A3 and RUBiS-D2 due to GARL are 42% and 82% higher than those due to MRL, while the effective throughput of RUBiS-A1 and RUBiS-C3 due to GARL are 15.6% and 20% lower than those due to MRL. GARL improves the overall system throughput by 13%.

Furthermore, we apply different combinations of applications to evaluate the effective system throughput difference between GARL and MRL. Table II gives the comparison. As the resource requirement of the application combination increases, GARL shows more performance improvement than MRL. For example, application combination (*B1, B1, B2, B3*) is consisted of single-tier applications and application combination (*A1, A3, B1, B3*) is consisted of multi-tier and single-tier applications. Performance gain by GARL is more significant with the combination (*A1, A3, B1, B3*) than that of application combination (*B1, B1, B2, B3*).

TABLE II
OVERALL PERFORMANCE IMPROVEMENT OF GARL.

RUBiS Apps	Stage 2	Stage 3	Stage 4	Overall
B1, B1, B2, B3	2%	3%	5%	5%
A1, A3, B1, B3	5%	18%	27%	16%
A1, A1, A2, A3	10%	21%	17%	16%
C1, C1, C2, C3	21%	14%	6%	14%

TABLE III
PER-APPLICATION PERFORMANCE IMPROVEMENT OF GARL.

RUBiS Apps	App1	App2	App3	App4
B1, B1, B2, B3	8%	10%	7%	$< 1\%$
A1, A3, B1, B3	-5%	36%	24%	-2%
A1, A1, A2, A3	-12%	51%	67%	-19%
C1, C1, C2, C3	-16%	37%	40%	-21%

Table III shows per-application effective throughput improvement due to GARL. It shows the trade-offs between multiple applications. GARL sacrifices the effective throughput of some applications slightly to improve the effective throughput of others as the resource contention level increases. Such a trade-off results in the improvement of effective system throughput. However, when the resource contention is too severe, the gain due to the GARL's coordination among application is significantly reduced.

D. The Scalability of GARL

We evaluate GARL's scalability. We deploy multiple single tier applications. We run GARL with an increasing number of applications, from 2 to 20, to evaluate the normalized effective system throughput. Each application has a VM capacity of 2.66 GHz CPU and 1 GB memory. For comparison, we implement a centralized reinforcement learning (RL) approach. The centralized RL approach puts all resource and server parameters into one large Q-Table. It uses ϵ-greedy policy to generate resource allocations and server parameters for all applications.

Figure 9 shows when the number of applications is equal or less than 16, the centralized RL approach and GARL achieve similar performance. When the number of applications is greater than 16, the effective system throughput due to the centralized RL decreases significantly. This because in the centralized RL approach, the size of Q-Table increases exponentially when the number of applications increases. The complexity of learning agent makes it harder for the centralized RL approach to converge and to find the optimal solution. In GARL's multi-agent design, the Q-table size of each reinforcement learning agent is fixed. Increasing the number of applications will not significantly increase the complexity of reinforcement learning. Therefore, GARL has good scalability.

VIII. CONCLUSIONS

This paper tackles the important but challenging problem of improving performance and power efficiency for multiple complex applications co-hosted in a data center. We have

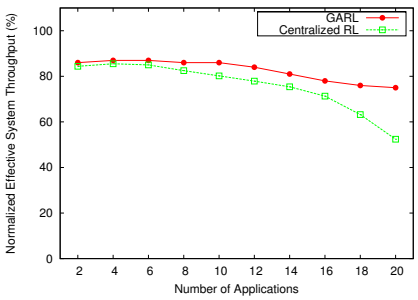


Fig. 9. Impact of scalability of different approaches on performance.

proposed GARL, an integration of genetic algorithm and multi-agent reinforcement learning approach, for coordinated VM resizing and server parameter tuning. GARL integrates the model-independency of reinforcement learning with the global optimization capability of genetic algorithm. It can significantly improve the effective system throughput and power efficiency of multiple complex applications while meeting the resource constraints and power budget of the shared resource pool. We have implemented GARL on a testbed in our university prototype data center. Experimental results using RUBiS benchmark applications have demonstrated that GARL significantly outperforms a multi-agent reinforcement learning approach in both performance improvement and power efficiency. The results have also demonstrated the good scalability of GARL. It outperforms a centralized reinforcement learning approach.

Our future work will be to explore the interaction among VM capacity planning and resizing, workload multiplexing, and server parameter tuning in Cloud-computing data centers.

Acknowledgement

This research was supported in part by U.S. National Science Foundation CAREER Award CNS-0844983.

REFERENCES

[1] C. Amza, A. Chanda, A. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite. Specification and implementation of dynamic web site benchmarks. In *Proc. IEEE Int'l Workshop on Workload Characterization (WWC)*, pages 3 – 13, 2002.

[2] X. Bu, J. Rao, and C.-Z. Xu. A reinforcement learning approach to online web system auto-configuration. In *Proc. IEEE Int'l Conference on Distributed Computing Systems (ICDCS)*, 2009.

[3] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *Proc. ACM SIGMETRICS*, pages 303–314, 2005.

[4] G. Dasgupta, A. Sharma, A. Verma, A. Neogi, and R. Kothari. Workload management for power efficiency in virtualized data centers. *Commun. ACM*, 54(7):131–141, July 2011.

[5] J. Gong and C.-Z. Xu. vpnp: Automated coordination of power and performance in virtualized datacenters. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2010.

[6] S. Govindan, J. Choi, B. Urgaonkar, A. Sivasubramaniam, and A. Baldini. Statistical profiling-based techniques for effective power provisioning in data centers. In *Proc. the EuroSys Conference*, 2009.

[7] Y. Guo, P. Lama, and X. Zhou. Automated and agile server parameter tuning with learning and control. In *Proc. of IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS)*, 2012.

[8] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan. vmanage: Loosely coupled platform and virtualization management in data centers. In *Proc. IEEE Int'l Conference on Autonomic Computing (ICAC)*, 2009.

[9] D. Kusic and J. O. Kephar. Power and performance management of virtualized computing environments via lookahead control. In *Proc. IEEE Int'l Conference on Autonomic computing (ICAC)*, 2008.

[10] P. Lama and X. Zhou. Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In *Proc. IEEE/ACM Int'l Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 151–160, 2010.

[11] P. Lama and X. Zhou. aMOSS: Automated multi-objective server provisioning with stress-strain curving. In *Proc. IEEE Int'l Conference on Parallel Processing (ICPP)*, 2011.

[12] P. Lama and X. Zhou. PERFUME: Power and performance guarantee with fuzzy mimo control in virtualized servers. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2011.

[13] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *Proc. IEEE Int'l Conference on Autonomic Computing (ICAC)*, 2007.

[14] X. Meng, C. Isci, J. Kephart, L. Zhang, and E. Bouillet. Efficient resource provisioning in compute clouds via vm multiplexing. In *Proc. Int'l Conference on Autonomic Computing (ICAC)*, 2010.

[15] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *Proc. ACM/IFIP/USENIX Int'l Middleware Conference*, 2008.

[16] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Injecting realistic burstiness to a traditional client-server benchmark. In *Proc. IEEE Int' Conference on Autonomic Computing (ICAC)*, 2009.

[17] J. Moses, R. Iyer, R. Illikkal, S. Srinivasan, and K. Aisopos. Shared resource monitoring and throughput optimization in cloud-computing datacenters. In *Proc. of IEEE Int'l Symposium on Parallel and Distributed Processing (IPDPS)*, 2011.

[18] R. Nathuji and K. Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, pages 265–278, 2007.

[19] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proc. of the EuroSys Conference (EuroSys)*, pages 13–26, 2009.

[20] J. Rao, X. Bu, C. Xu, L. Wang, and G. Yin. Vconf: A reinforcement learning approach to virtual machines auto-conguration. In *Proc. IEEE Int'l Conference on Autonomic Computing Systems (ICAC)*, 2009.

[21] J. Rao, X. Bu, C.-Z. Xu, and K. Wang. A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In *Proc. IEEE/ACM Int'l Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS)*, 2011.

[22] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[23] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proc. IEEE Int'l Conference on Autonomic Computing (ICAC)*, 2006.

[24] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier Internet applications. *ACM Trans. on Autonomous and Adaptive Systems*, 3(1):1–39, 2008.

[25] A. Verma, P. Ahuja, and A. Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Proc. ACM/IFIP/USENIX Int'l Middleware Conference*, 2008.

[26] A. Verma, P. De, V. Mann, T. Nayak, A. Purohit, G. Dasgupta, and R. Kothari. Brownmap: Enforcing power budget in shared data centers. In *Proc. ACM/IFIP/USENIX Int'l Middleware Conference*, 2010.

[27] X. Wang, M. Chen, and X. Fu. Mimo power control for high-density servers in an enclosure. *IEEE Trans. on Parallel and Distributed Systems*, 21(10):1412–1426, 2010.

[28] X. Wang and Y. Wang. Co-con: Coordinated control of power and application performance for virtualized server clusters. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2009.

[29] Y. Wang, X. Wang, M. Chen, , and X. Zhu. Partic: Power-aware response time control for virtualized web servers. *IEEE Trans. on Parallel and Distributed Systems*, 21(4), 2010.

[30] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang. Probabilistic performance modeling of virtualized resource allocation. In *Proc. IEEE Int'l Conference on Autonomic computing (ICAC)*, 2010.