Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol II,
IMECS 2010, March 17 - 19, 2010, Hong Kong

Dynamic Session Management Based on Reinforcement Learning in Virtual Server Environment

Kimihiro Mizutani, Izumi Koyanagi, Takuji Tachibana *

Abstract— **In a virtualized server environment, machine resources such as CPU and memory are shared by multiple services. In such an environment, as the number of sessions for each service increases, the amount of resources that are utilized by the services increases. If thrashing occurs due to a lack of resources, the performance of the server is degraded. It is effective to estimate the amount of used resources; however, it is hard to estimate the amount of resources that are used dynamically by multiple services. In this paper, we propose a dynamic session management based on reinforcement learning in order to utilize the resources effectively and avoid the thrashing. In the proposed method, a learning agent estimates the amount of used resources from the response time for a service request. Then, the agent decides the acceptance or rejection of an arriving session request with Q-learning. Because this method can be implemented easily in a physical machine, it is expected that our proposed method is used in a real environment. We evaluate the performance of our proposed method with a simulation. From the simulation results, we show that the proposed method can allocate the resources to multiple services effectively while avoiding the thrashing and can perform the priority control for multiple services.**

Keywords: Virtual server environments, Session management, Reinforcement learning, Thrashing, Resource allocation

1 Introduction

These days, it has become possible to build a large-scale server at a reasonably low cost. On the other hand, almost all of the machine resources such as CPU and memory tend to be utilized when the server is idle. In order to utilize such machine resources effectively, [1] has proposed a method that can create a virtual server environment. In the virtual server environment, multiple virtualized servers are built on one physical server and a service can be provided by each virtualized server. Therefore, multiple services can be provided by a physical server in the virtual server environment.

*Graduate School of Information Science, Nara Institute of Science and Technology 8916-5 Takayama, Ikoma, Nara 630-0192, Japan. E-mail: {kimihiro-m,izumi-ko,takuji-t}@is.naist.jp

Techniques for creating a virtualized server environment are classified into two groups; application-level virtualization and operating system (OS)-level virtualization. In the application-level virtualization, a web server application such as Apache [2] and Tomcat [3] is utilized to manage multiple virtualized servers on one physical server. The web server application supports the virtual host, and the virtual host provides dedicated ports and domains for every virtualized server. Hence, multiple services can be provided independently on the physical server.

On the other hand, the OS-level virtualization is also classified into two groups, hosted virtualization and hypervisor virtualizations. In the hosted virtualization, multiple virtualized servers are built on an OS of the physical server [4, 5]. Here, an OS of each virtualized server is emulated on the OS of the physical server, and hence some functions such as system call for each virtualized server are performed via the OS of the physical server. This increases overhead for executing operations on virtualized servers, resulting in processing delay. In the hypervisor virtualization, on the other hand, multiple virtualized servers are built on BIOS of the physical server [6]. Because the OS of the physical server is never used for executing operations on virtualized servers, the overhead is smaller than that for the hosted virtualization. As a result, hypervisor virtualization is mainly utilized for a large-scale server.

Here, in order for virtualized servers to provide services without problems, a physical server should allocate its own resources to each virtualized server effectively [7, 8, 9]. If resources are allocated to each virtualized server at random, the servers could compete for getting resources in order to provide their own services. This competition results in thrashing and the performance of the physical server is degraded [10]. Therefore, in order to manage multiple virtualized servers effectively, it is indispensable to estimate the amount of resources that are used in each virtualized server [11].

In this paper, we propose a dynamic session management based on reinforcement learning in order to allocate resources to virtualized servers effectively and avoid the

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

thrashing. In our proposed method, a learning agent of reinforcement learning estimates the state of each virtualized server by measuring response time of the corresponding virtualized server for a new arriving session. Based on this estimation, the agent decides whether the arriving session is accepted or not. Here, the acceptance or rejection of a session is defined as action in reinforcement learning, and our proposed method derives the optimal action for each state according to the reward function. Because the agent can obtain a large reward when a large amount of resources are utilized but obtain a small reward when the thrashing occurs, our proposed method can allocate resources to multiple virtualized servers so as to avoid the thrashing. In addition, the method can provide service differentiation for virtualized servers by considering the priority of each service.

We also consider how to implement our proposed method into a physical server. We evaluate the performance of this method with simulation, and then we compare its performance with the conventional method without reinforcement learning.

The rest of the paper is organized as follows. Section 2 introduces the existing methods of dynamic session management as related works. In Section 3, we present our proposed dynamic session management based on reinforcement learning. Section 4 explains how to implement this method in Apache and Xen. We show some numerical examples in Section 5 and present conclusions in Section 6.

2 Related Work

In general, a Web server has to manage sessions independently of both the number of established sessions and service time for each session [12, 13]. On the other hand, it is important for the server to manage sessions so that processes for sessions do not exceed its own processing capacity. Here, the processing capacity is calculated from the server's information such as response time for each session, throughput, which is the number of established sessions per unit of time, and CPU utilization. Several methods of dynamic session management have been proposed to manage sessions by using this information.

[14] has proposed a dynamic session management in which the server's information about response time is utilized. This method does not accept arriving session requests when the response time is over a certain threshold. Here, the threshold is determined dynamically from the weighted sum of response times that are measured for a period of time. When the weight of the latest response time is large, sessions can be managed by considering the instantaneous change of CPU load. If the weights of past response times are large, sessions can be managed more stably. Once a session is established, the server keeps establishing the session until the process of this session is completed. Thus, the performance of this method is affected by the threshold.

Moreover, [15] has proposed a dynamic session management where both CPU utilization and service utilization are utilized. In this method, CPU utilization and the number of sessions are measured for a period of time. From the measurement information, this method calculates the average CPU utilization when a new session is established during the period. Then, based on the average CPU utilization, the server manages sessions during the next period. Here, as the measurement period decreases (increases), the number of sessions that are established during the period becomes small (large). Therefore, when the period is small, the average CPU utilization is close to the actual CPU utilization that is measured when a session is established. For example, if only a session is established during the period, the average CPU utilization is the same as the actual CPU utilization. However, the average CPU utilization may be changed drastically by some factors other than the establishment of session. In this case, the impact of the session establishment cannot be estimated. On the other hand, when the period is large, the impact of the factors becomes small. However, a larger number of sessions are established for the period, and hence the difference between the average CPU utilization and the actual CPU utilization may become large. Hence, it is indispensable to determine the measurement period so as to estimate the impact of session establishment on the CPU utilization.

The above methods are utilized only when a physical server provides only one service. Therefore, in a virtual server environment, these methods cannot manage sessions for allocating machine resources to multiple virtualized servers [7, 15].

3 Dynamic Session Management-Based on Reinforcement Learning

In this paper, we propose a dynamic session management based on reinforcement learning in order to manage sessions in a virtualized server environment. In the following, at first, we explain Q-learning, which is used in our proposed method. Then we describe the proposed session management in detail.

3.1 Q-learning

In reinforcement learning, a learning agent moves from state to state by performing action a. The learning agent receives a reward or a punishment at each state, and it tries to perform an action so as to maximize the total reward. From these experiences, the agent can learn which action is the optimal for each state.

Q-learning is one of the reinforcement learning tech-

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

niques, and in Q-learning, an action-value function Q is used to determine the optimal action for each state. Now, let s_t denote a state and a_t denote an action at time t, respectively. In addition, $R^{a_t}_{s_t,s_{t+1}}$ denotes a reward that the agent can obtain when it moves from s_t to s_{t+1} with a_t. When the agent moves from state to state s_{t+1} by performing action a_t, $Q(s_t, a_t)$ is updated as follows:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[R^{a_t}_{s_t,s_{t+1}} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right], \quad (1)$$

where α $(0 < \alpha \leq 1)$ is the learning rate and γ $(0 < \gamma \leq 1)$ is the discount factor. The learning rate determines to what extent the latest information will override the old information. The agent does not learn anything when α is 0, while the agent considers only the newest information when α is 1. The discount factor γ determines the influence of future rewards. The agent considers only the current reward when γ is 0, while the agent tries to find a long-term high reward when γ is 1.

From the learned Q-function, the learning agent can determine the optimal action a^*_t by using ϵ-greedy policy. That is, the agent selects an action that can maximize $Q(s_t, a_t)$ with probability $1 - \epsilon$ $(0 \leq \epsilon \leq 1)$ and selects one of the other actions with probability ϵ.

3.2 Server model

In this paper, we focus on a virtualized server environment that satisfies the following assumptions.

- The number of virtualized servers on a physical server is M and virtualized server i provides web service i $(1 \leq i \leq M)$.
- Service i $(1 \leq i < j \leq M)$ has higher priority over class j if $i < j$.
- A higher-priority class requires a smaller session blocking probability than a lower-priority session.
- The maximum number of sessions for service i is N_i.
- The number of established sessions for service i at time t is $L_i(t)$ $(L_i(t) \leq N_i)$.

Note that we do not make any assumptions about the arrival process of the session and the distribution of the session holding time.

3.3 Definition of state and action for our proposed method

To utilize reinforcement learning, state s_t and action a_t must be defined. In our method, let state $s_t \in S$ denote the number of sessions for each service and we define s_t as follows.

$$s_t = (L_1(t), L_2(t), \cdots, L_M(t), U(t)). \quad (2)$$

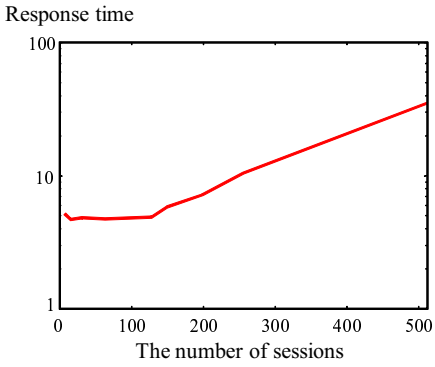

Figure 1: Experimental result of resource utilization.

In (2), $U(t)$ is a variable that indicates the resource utilization for the physical server at time t. Here, $U(t)$ is set to $busy = 3$, $normal = 2$, or $idle = 1$ depending on the resource utilization. We explain how to set $U(t)$ in the next subsection.

We also define action $a_t \in A$ of the learning agent at time t as follows.

- Only a session of service i $(1 \leq i_1 \leq M)$ can be accepted.
- A session of classes i or j $(1 \leq i \neq j \leq M)$ can be accepted.
- A session among m $(2 < m < M)$ classes can be accepted.
- A session among all M class can be accepted.

The total number of actions is given by 2^{M-1}.

3.4 Estimation of resource utilization

In terms of (2), the learning agent can observe the number $L_i(t)$ of sessions for service i at time t but cannot observe resource utilization $U(t)$. Therefore, the agent estimates the resource utilization by measuring response time of each virtualized server.

It is well known that response time for a service increases exponentially when the number of established sessions increases and thrashing occurs[16]. Figure 1 shows our experimental result in a web server with Apache. Note that the vertical axis is logarithmic. This figure shows the response time of the server against the number of established sessions. From this figure, we find that the response time is almost constant when the number of sessions is smaller than about 120. However, once the number of sessions becomes larger than about 120, the response time increases exponentially. This result also proves that the server's response time increases exponentially when the thrashing occurs. From these results, we

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

measure the response time of each virtualized server in order to avoid the thrashing.

Here, in our proposed method, $U(t)$ is set by estimating the resource utilization. Now, we define $N_i(t)$, ST_i, $RT_i(t)$, and $C(t)$ as follows.

$T_i(t)$ **:** This denotes response time of server i $(1 \leq i \leq M)$ when a new session is established for server i at time t. If a new session is not established, this is equal to zero.

ST_i**:** This denotes the smallest response time of server i $(1 \leq i \leq M)$ from time 0 to t.

$RT_i(t)$**:** This denotes the ratio between the measured response time $T_i(t)$ of server i $(1 \leq i \leq M)$ and the smallest response time ST_i.

$C(t)$**:** This denotes the rate of change of $RT_i(t)$ from time $t-n$ to t.

Here, $T_i(t)$ is measured by virtualized server i or the physical server. ST_i is also given by $ST_i = \min_t T_i(t)$. Because $T_i(t)$ is equal to zero when a new session is not established for server i at time t, $RT_i(t)$ is given by $RT_i(t) = \sum_{i=1}^{M} T_i(t)/ST_i$. Therefore, with least-square approach, $C(t)$ is given by

$$C(t) = \frac{n\sum_{i=t-n}^{t} i\log(RT_i(t)) - \sum_{i=t-n}^{t} i \sum_{i=t-n}^{t} \log(RT_i(t))}{n\sum_{i=t-n}^{t} i^2 - \left(\sum_{i=t-n}^{t} i\right)^2}. \quad (3)$$

From (3), the resource utilization increases as $C(t)$ becomes large. Therefore, based on $C(t)$, we set $U(t)$ as follows:

$$U(t) = \begin{cases} busy = 3, & \text{if } busy \leq C(t), \\ normal = 2, & \text{if } normal \leq C(t) < busy, \\ idle = 1, & \text{if } idle \leq C(t) < normal. \end{cases}$$

3.5 Reward function

When virtualized server i establishes a new session for service i, the learning agent obtains reward r_i $(0 \leq r_i)$. In our proposed method, in order to provide service differentiation for M services, r_i satisfies the following inequalities.

$$0 \leq r_M \leq \cdots \leq r_1. \quad (4)$$

According to (4), the learning agent can obtain a high reward when the agent accepts a high priority session. Let $R^{a_t}_{s_t,s_{t+1}}$ denote a reward function when the learning agent transits from s_t to s_{t+1} by performing action a_t. $R^{a_t}_{s_t,s_{t+1}}$ is given by

$$R^{a_t}_{s_t,s_{t+1}} = \sum_{i=1}^{M} \left(\frac{ST_i L_i(t) r_i}{T_i(t)} - L_i(t)U(t) \right). \quad (5)$$

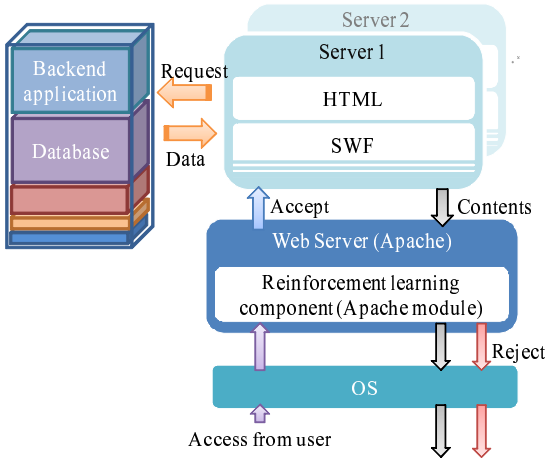

Figure 2: An implementation of our proposed method with Apache module.

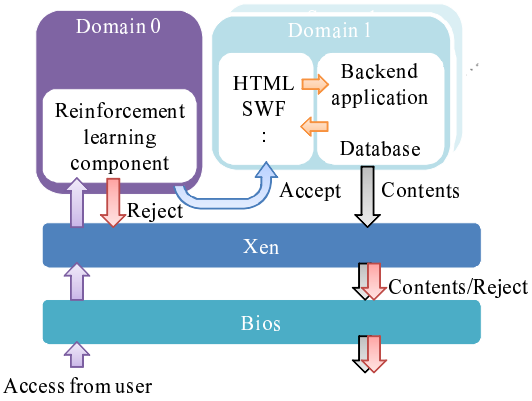

Figure 3: An implementation of our proposed method with Xen.

In the first term of (5), $R^{a_t}_{s_t s_{t+1}}$ becomes large when a session of high-priority service is established, given that response time increases against the number of sessions at the same rate regardless of services. Moreover, in the second term of (5), $R^{a_t}_{s_t s_{t+1}}$ becomes large when response time is large due to some reason such as thrashing, that is, when $U(t)$ is large.

Finally, the learning agent derives the optimal action a^*_t for each state with (1) and (5). As the learning time of the agent becomes large, the performance of the proposed method can be improved.

4 Implementation of Our Proposed Method

Currently, Apache and Xen are widely used to build multiple virtualized servers on a physical server. Apache [2] creates a virtualized server environment according to application-level virtualization. On the other hand, Xen

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

creates a virtualized server environment according to hypervisor virtualization. We explain how our proposed method can be implemented into Apache and Xen.

Figure 2 shows the implementation of our proposed method into a physical server with Apache. In the server, some functions that are utilized for establishing sessions can be changed by utilizing the Apache module, managing sessions flexibly. Therefore, we implement the Q-learning component into a session management component of the server. When the server receives a request for a new session, it changes a function of session acceptance according to the optimal action that is derived with Q-learning.

In Xen, on the other hand, machine resources such as CPU and memory can be controlled by executing some commands. For example, tc command can be utilized to determine the priority of each virtualized server. Therefore, we implement the Q-learning component into Domain 0, which is for session management (see Fig. 3). When a session request arrives at the physical server, it executes a command according to the optimal action that is derived with Q-learning. Thus, our proposed method can be implemented into a virtualized server environment easily.

5 Numerical Examples

In this section, we evaluate the performance of our proposed method with simulation. Here, we assume that our proposed method is implemented into a physical server with Apache as shown in Fig. 2.

For the physical server, the number of virtualized servers is $M = 3$ and virtualized server i ($i = 1, 2, 3$) provides only service i. We assume that session requests of service i arrive at virtualized server i according to a Poisson process with $\lambda_i = 1.0$. The holding time of the established session is exponentially distributed with mean $1/\mu_i = 0.05$ regardless of service i. Here, we calculate resource utilization of the physical server by $\sum_{i=1}^{M} T_i(t)$. Thrashing occurs when $\sum_{i=1}^{M} T_i(t)$ is larger than 100. Once the thrashing occurs, response time increases exponentially.

For our proposed method, we set α and γ to 0.9, and ϵ is equal to 0.9. In addition, rewards of session establishment are $r_1 = 20$, $r_2 = 10$, and $r_3 = 5$. The maximum number of sessions for service i is $N_i = 100$. For simplicity, we set $ST_1 = 2$, $ST_2 = 3$, and $ST_3 = 4$ regardless of time t.

5.1 Impact of session management on service differentiation and resource utilization

In this section, we investigate the impact of our proposed method on the effectiveness of service differentiation and resource utilization. Figure 4 shows the session blocking probability for each service against the learning time. Here, the session blocking probability for each service is calculated per 10,000 arriving sessions. From this figure, we can find that the differences among blocking probabilities are small when the learning time is small. However, as the learning time becomes large, the differences become large so that service differentiation can be provided for M services.

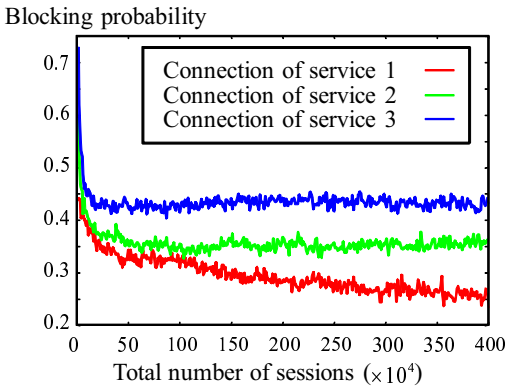

Figure 4: Blocking probability of each service vs. number of arriving session requests.

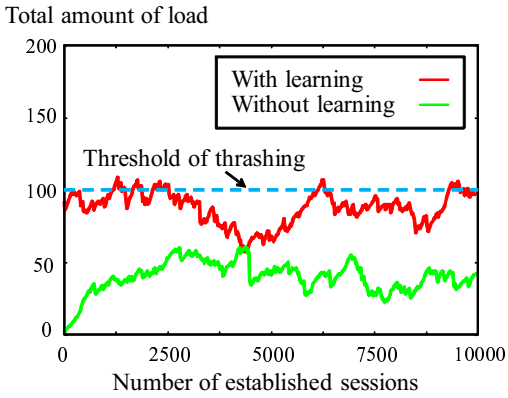

Figure 5: Total amount of CPU load vs. number of established sessions.

In this figure, the proposed method is effective for service differentiation when the number of arriving sessions is larger than 1.5 million. This means that it takes a long time for the agent to learn the optimal action. Therefore, it is important to find the optimal action as soon as possible.

5.2 Impact of Q-learning on avoidance of thrashing

Moreover, we investigate how our proposed method affects the load on each virtualized server. Figure 5 shows

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the load against the total number of sessions for the physical server. In this figure, a result of our proposed method is denoted as "With learning" but the result of the existing method is denoted as "Without learning."

From this figure, we can see that thrashing never occurs when the existing method is utilized. However, the resource utilization is much smaller (about 50%) in this method.

On the other hand, the proposed method can utilize a larger amount of resources (almost 100%) effectively. Although thrashing sometimes occurs in our method, the interval between two successive episodes of thrashing becomes large as the agent learns the optimal action.

6 Conclusions

In this paper, we proposed a dynamic session management based on reinforcement learning for utilizing machine resources effectively and avoiding thrashing. In addition, we showed an implementation method for two types of virtualized servers. We evaluated the performance of the proposed method with a simulation, and we found that our proposed method can provide service differentiation for each class. We also observed that our proposed method can be effective for reducing the number of thrashing episodes and for assigning the resources of each virtualized server. On the other hand, our proposed method requires a lot of learning time. In order to use our proposed method in a real environment, we will extend the method so that the agent can learn the optimal action as soon as possible.

Acknowledgment

This paper is supported in part by the Creative and International Competitiveness Project (CICP).

References

[1] C.A. Waldspurger, "Memory Resource Management in VMware ESX Server," in *Proc. the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, ACM Operating Systems Review, Winter 2002 Special Issue, USA, Dec. 2002, pp. 181-194.

[2] Apache Software Foundation. http://www.apache.org/.

[3] Apache Tomcat Software Foundation. http://tomcat.apache.org/.

[4] VMware Server/Workstation/Player. http://www.vmware.com/.

[5] Qemu. http://www.nongnu.org/qemu/.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proc. the Nineteenth ACM Symposium on Operating Systems Principles*, USA, Oct, 2003.

[7] G. Tesauro, N.K. Jong, R. Das, and M.N. Bennani, "A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation," in *Proc. the 5th IEEE International Conference on Autonomic Computing (ICAC 2006)*, June 2006, pp. 65-73.

[8] G.W. Dunlap, S.T. King, S. Cinar, M. Basrai, and P.M. Chen, "ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay," in *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, USA, Dec. 2002, pp. 211-224.

[9] T. Garnkel, M. Rosenblum, and D. Boneh, "Flexible OS Support and Applications for Trusted Computing," in *Proc. the 9th Workshop on Hot Topics in Operating Systems*, Hawaii, May 2003.

[10] Y. Diao, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury, "Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics with Application to the Apache Web Server," in *Proc. IEEE/IFIP Network Operations and Management Symposium*, Apr. 2002.

[11] M.R. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835-846, Dec. 1997.

[12] L. Cherkasova and P. Phaal, "Session Based Admission Control: A Mechanism for Improving Performance of Commercial Web Sites," in *Proc. Seventh Int'l Workshop Quality of Service*, May 1999.

[13] M. Arlitt, "Characterizing Web User Sessions," *HP Labs Report*, 2000.

[14] L. Cherkasova and P. Phaal, "Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites," *IEEE Transactions on Computers*, Vol. 51, no. 6, pp. 669-685, June 2002.

[15] Q. Zhang, L. Cherkasova, and E. Smirni, "A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications," in *Proc. the 4th IEEE International Conference on Autonomic Computing (ICAC 2007)*, June 2007.

[16] M.F. Arlitt and C.L. Williamson, "Web Server Workload Characterization: The Search for Invariants," in *Proc. ACM SIGMETRICS*, Philadelphia, Apr. 1996.

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)