An SLA-aware Load Balancing Scheme for Cloud Datacenters

Chung-Cheng Li and Kuochen Wang
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan 300
shinji10343@hotmail.com, kwang@cs.nctu.edu.tw

***Abstract*—One of the most important issues about cloud computing is how to achieve load balancing among thousands of virtual machines (VMs) in a large datacenter. In this paper, we propose a novel decentralized load balancing architecture, called tldlb (*two-level decentralized load balancer*). This distributed load balancer takes advantage of the decentralized architecture for providing scalability and high availability capabilities to service more cloud users. We also propose a neural network-based dynamic load balancing algorithm, called nn-dwrr (*neural network-based dynamic weighted round-robin*), to dispatch a large number of requests to different VMs, which are actually providing services. In nn-dwrr, we combine VM load metrics (CPU, memory, network bandwidth, and disk I/O utilizations) monitoring and neural network-based load prediction to adjust the weight of each VM. Experimental results support that our proposed load balancing algorithm, nn-dwrr, can be applied to a large cloud datacenter, and it is 1.86 times faster than the wrr, 1.49 times faster than the Capacity-based, and 1.21 times faster than the ANN-based load balancing algorithms in terms of average response time. In addition, tldlb can reduce the SLA (service-level agreement) violation rate via in-time activating VMs from a spare VM pool.**

***Keywords*—cloud computing; decentralized architecture; load balancing; neural network; service level agreement**

I. INTRODUCTION

Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility (like the electricity grid) over a network (typically the Internet) [1]. Cloud Computing has been envisioned as the next-generation architecture of IT enterprises. Therefore, it rapidly grows in recent years. We can clearly find that the number of users which use cloud computing grows very fast, as shown in Figure 1. We can see the growth of average daily instance launch counts in Amazon EC2 is very fast.

The load of a cloud computing system is highly dynamic. Different users may require different services, and it may lead to load unbalance between the servers (virtual machines, VMs) in a cloud datacenter. To conquer this problem, user requests are sent to a load balancer and the load balancer then forwards them to the appropriate VMs for processing in cloud datacenters. The function of load balancing aims to realize a high ratio of user satisfaction and facilitate high resource utilization in the cloud [3]. Improper allocation rules might cause the inefficiency of the cloud system [4]. Therefore, we need a load balancer in a cloud computing system to receive user requests and forward them to appropriate servers (or VMs) to service the user requests [5][6].

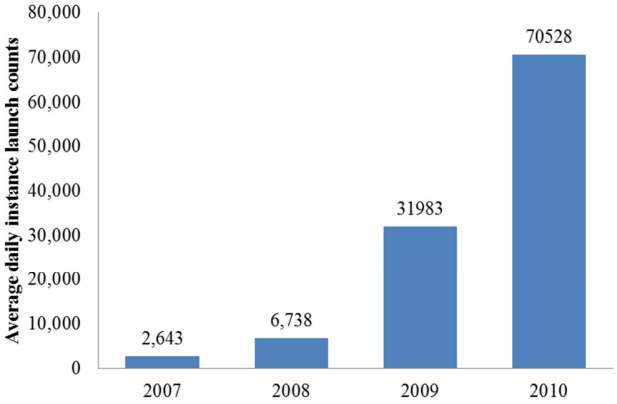

Figure 1. The cloud scales: Amazon EC2 growth [2].

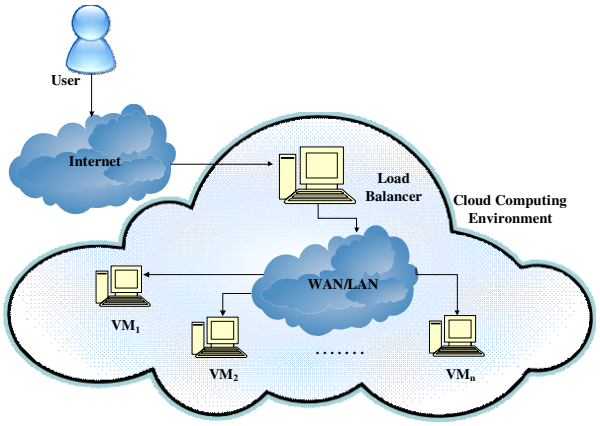

Figure 2. A classic load balancer architecture in a cloud datacenter.

Figure 2 illustrates a classic load balancer architecture in a cloud datacenter. All user requests will be connected to a load balancer. Obviously, we cannot expect one load balancer to deal with the burden of the entire cloud datacenter. We can use a technique which is similar to Amazon's Auto Scaling. When one load balancer is overloaded, it will start another load balancer to share the load of user requests.

A service-level agreement (SLA) is a part of a service contract where the level of service is formally defined [7]. The SLA will typically have a technical definition in terms of response time, throughput, or similar measurable details [7]. In this paper, we aim to reduce the SLA violation rate while designing a load balancing architecture and algorithm.

In section II, we review architecture and algorithms of existing load balancer designs in cloud computing environments and depict the differences between centralized and distributed load balancer designs. In section III, we propose a new architecture, called an SLA-aware two-level decentralized load balancer (tldlb), to support dynamic load balancing in cloud datacenters and also proposes a novel load balancing algorithm, called neural network-based dynamic weighted round-robin (nn-dwrr), to dispatch requests to appropriate VMs. Experimental results and the comparison of different load balancing algorithms are discussed in section IV. Finally, section V gives concluding remarks.

II. RELATED WORK

There are several load balancing architectures in cloud computing environments [3][8][9][10]. All these architectures broadly implement load balancing algorithms, which can be static or dynamic, and also use centralized or decentralized control [8]. Therefore we roughly divide load balancing architectures into two categories, centralized and decentralized. The centralized load balancer architecture has a single load balancer which receives an incoming request and then selects a proper VM to serve the request by a scheduling algorithm. In this architecture, the load balancer may become a bottleneck in cloud environments if the request rate grows to exceed the capacity of the load balancer. That is, this architecture lacks scalability in cloud environments. The decentralized load balancer architecture has several load balancers in cloud environments. Incoming requests will be dispatched to load balancers randomly or adjacent load balancers. Although the decentralized load balancer architecture has more scalability than the centralized load balancer architecture, it needs more communication cost to share load information among load balancers.

We review two existing load balancing algorithms and then propose a neural network-based dynamic weighted round-robin (nn-dwrr) scheduling algorithm. The first existing scheduling algorithm is called the weighted round-robin scheduling algorithm (wrr) [10]. It assigns a fixed weight to each VM depending on the VM's processing capacity at the startup. The second capacity-based scheduling algorithm (Capacity-based) monitors the resources of each VM and distributes more requests to the VM which has more remaining resources [12]. The main concept is distributing requests to a VM which has the most remaining capacity. In contrast, the proposed neural network-based dynamic weighted round-robin algorithm (nn-dwrr) adjusts weights based on neural network-based load prediction, and it will be detailed in the next section.

III. PROPOSED SLA-AWARE LOAD BALANCING SCHEME FOR CLOUD DATACENTERS

We propose an SLA-aware two-level decentralized load balancer (tldlb) architecture and a neural network-based dynamic weighted round-robin scheduling algorithm (nn-dwrr) to support dynamic load balancing in cloud data centers. The decentralized load balancer architecture in our design, as shown in Figure 3, is divided into two levels: global load balancer and local load balancer. Each global load balancer is connected to an SLA-aware local load balancer that forms a virtual zone. The distributed load balancer architecture is described as follows:

1) Local load balancer

A local load balancer has two main tasks. The first task is monitoring the load of VMs which are in the same virtual zone. The local load balancer will obtain four load metrics (CPU, memory, network bandwidth, and disk I/O utilizations) from each VM and the response time of each request for VMs. The local load balancer will provide the above information to the global load balancer. If the current working VMs (VM_1 through VM_n) can't handle the load, the local load balancer will activate some spare VMs from a spare VM pool (VM_{s1} through VM_{sm}) to provide the service. The second task is choosing an appropriate VM using a neural network-based load balancing algorithm and then redirects the request to the VM. Our local load balancer is SLA-aware, which assigns requests to appropriate VMs for servicing so as to meet the SLA requirement.

2) Global load balancer

Global load balancers are connected to one another via P2P connections. The global load balancers exchange the load information of each virtual zone using the load information from each local load balancer. If there is no VM available in the spare VM pool to serve an overloaded virtual zone to meet the SLA requirement, the corresponding global load balancer will direct the requests to another light-loaded virtual zone to service the requests.

Figure 4 shows the architecture of an SLA-aware local load balancer along with a spare VM pool. The following is a brief description of each module.

- *Request Handler*

This module receives requests and forwards them to the Request Scheduler module. When the workload of a virtual zone reaches the upper limit, this module will redirect the requests to another Request Handler which belongs to another virtual zone.

- *Request Scheduler*

This module assigns the requests from the Request Handler module to the selected VMs based on the weights from the Weight Adjustment module. We give each VM a weight and then the Request Scheduler module distributes the requests to appropriate VMs by these weights.

- *Load Monitor*

It monitors four utilization metrics (CPU, memory, network bandwidth, and disk I/O utilizations) of each VM in this local load balancer. These utilization data allow the local load balancer to dynamically adjust the capacity index (CI_i) for VM_i.

- *History Storage*

The load history information collected by the Load Monitor

module and the weights history data from the Weight Adjustment module will be stored in this module. The weights history data can support the Load Prediction module to predict the load at the next time slot.

- *Load Prediction*

This module uses load history data, weights history data, and the specified response time from the SLA Engine module to predict a neural network index (NI_i) for VM_i. The NI_i's are sent to the Weight Adjustment module. In addition, we use an artificial neural network (ANN) with the delta leaning rule in our design (see Figures 5 and 6).

- *SLA Engine*

This module records the response time of each request and checks if the response time satisfies the SLA requirement.

- *Weight Adjustment*

This module adjusts the weight of each VM which belongs to this local load balancer according to the remaining capacity information (CI_i) of each VM_i from the Load Monitor module and load prediction information (NI_i) from the Load Prediction module.

- *Active VMs and a Spare VM pool*

There are active VMs and some suspended VMs in the spare VM pool. When active VMs can't handle incoming requests to meet the SLA requirement, the Request Handler module will wake up some spare VMs to service the requests.

In this paper, we focus on dynamically adjusting the weight of each VM. We propose a novel neural network-based load balancing algorithm, called *nn-dwrr* (neural network-based dynamic weighted round-robin), to dispatch requests to appropriate VMs based on their weights. A weight should be able to reflect the remaining capacity of a VM. We give each active VM a weight according to the capacity index (CI_i) from the Load Monitor module and the neural network index (NI_i) from the Load Prediction module. The Request Scheduler module distributes the requests to active VMs by their weights assigned by the Weight Adjustment module.

The first part of the information required by the Weight Adjustment module is remaining capacity information. Load balancing ought to be achieved using an inferred system state based on locally gathered data [11]. The Load Monitor module collects four load metrics, utilizations of CPU, memory, network bandwidth, and disk I/O. The Weight Adjustment module uses the following formula to calculate the capacity index (CI_i) for VM_i.

$$CI_i = 1 - MAX(CPU_i, Mem_i, Bandwidth_i, Disk\ I/O_i)$$

The larger capacity index means more remaining resources in this VM. We are not sure what kinds of services will be provided in datacenters. Different services require different critical resources. For example, the critical resource of a Web server is CPU and the critical resource of a FTP server is network bandwidth. The critical resource may become the bottleneck of a VM. Therefore we simply use a maximal to find the current bottleneck of a VM [13].

The second part is the load prediction information from a neural network-based load predictor. Remind that we used the delta learning rule in our ANN design because the neural network has the capability of optimization and prediction. Due to no certain mathematical approach for obtaining the optimum number of hidden layers and their neurons [14], we used a single hidden layer for less computation time in our design.

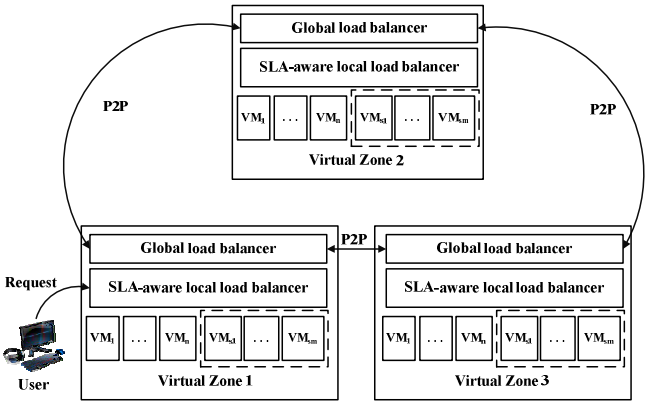

Figure 3. Proposed two-level decentralized load balancer (tldlb) architecture.

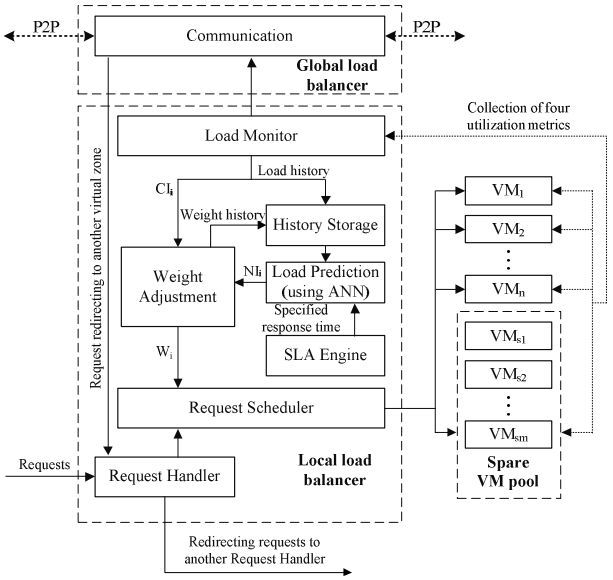

Figure 4. The architecture of an SLA-aware local load balancer along with a spare VM pool.

In Figure 5, input x is a vector which contains recent ten history weights. To avoid SLA violations, such as the response time required (d_i), which is specified in the SLA, we consider the response time when training the neural network. The neural network will calculate a weight for each VM_i, which we call neural network index$_i$ (NI_i). The Request Scheduler module allocates requests according to NI_i, and then measure the average response time (o_i). When the current average response time is close to the certain proportion (called pre-reaction rate (p), e.g., 80%) of the response time specified in the SLA, the neural network will automatically adjust the hidden layer's weights before SLA being violated. If the learning rate (α) is set to a large value, the neural network can learn faster. However, if there is a large variation in input, then the neural network may not learn well. We use the following formulas to train the neural network:

$$NI_i = f\left(\sum f(net_j)\right)$$
$$r = (p \times d_i - o_i) \times f'(net_j)$$
$$\Delta\omega = \alpha \times r \times x$$
$$w_{j(t+1)} = w_{j(t)} + \Delta\omega_j$$

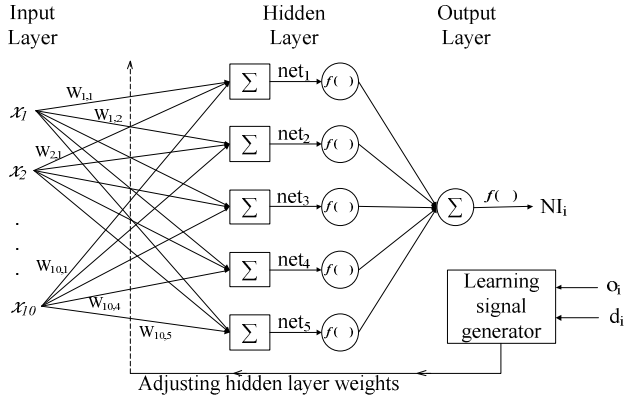

Figure 5. Schematic representation of an ANN model for VM_i.

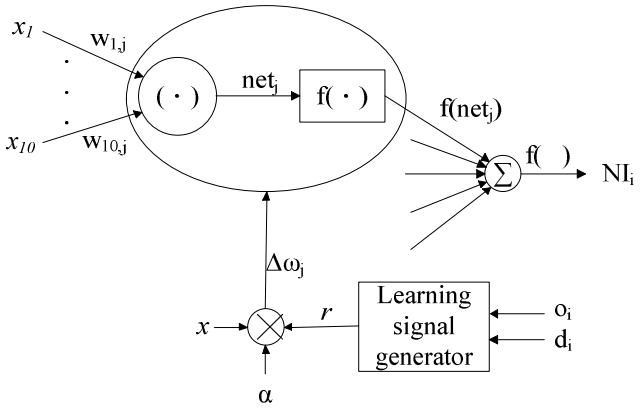

Figure 6. The process of delta learning rule for VM_i.

If there are n VMs in a local load balancer, the Weight Adjustment module will combine remaining capacity index CI_i and neural network index NI_i together to calculate a weight (W_i) for VM_i by the following formula:

$$W_i = \frac{CI_i \times NI_i}{\sum_{j=1}^{n}(CI_j \times NI_j)} * 100\%$$

W_i reflects the remaining resources proportion of VM_i in the entire n VMs. The Weight Adjustment module sends these weights to the Request Scheduler module.

IV. Evaluation and Discussion

We built a testbed that includes a local load balancer and five VMs. This testbed was for hosting a web service. There was three active VMs (VM_1, VM_2, and VM_3) with different capabilities and two spare VMs (VM_{s1} and VM_{s2}), which were running in an Apache web server in a virtual zone. We used the load balancer to link these VMs together to form a virtual zone. The load balancer would distribute requests to three VMs according the proposed scheduling algorithm nn-dwrr. The experimental environment setup and related parameters are shown in Table I and the configuration of the five VMs is shown in Table II.

Table I. Load balancing experiment parameters.

OS	CentOS 5.5
Virtual machine hypervisor	Xen
Number of VMs	3
Number of spare VMs	2
Application	Web service
Duration (time limit)	60 sec
Response time specified in SLA	2000, 1000, 432 ms
Pre-reaction rate (p)	80%
Transfer function (f) (for hidden and output layers)	Log-sigmoid
Learning rate (α)	0.5

Table II. The configuration of the five VMs.

Virtual Machine	VM_1	VM_2	VM_3	VM_{s1}	VM_{s2}
CPU (cores)	1	2	3	2	2
Memory (MB)	512	1024	2048	1024	1024
Virtual disk (GB)	10	10	10	10	10
Static weight (wrr)	1	2	4	-	-

We used this testbed to host the web service and evaluated average response time using an Apache benchmark (ab) to collect real web traffic for different load balancing algorithms. Requests were based on a real web service. We compare four different scheduling algorithms. How to utilize the advantage of cloud computing and make each task to obtain the required resources in the shortest time is an important topic [9]. Therefore, we used average response time as a metric for comparing the four scheduling algorithms.

Figure 7 shows the comparison of the four scheduling algorithms. The response time requirement specified in the SLA is 2000 ms. In Figure 7, we found that the static scheduling algorithm (wrr) has the longest response time. The Capacity-based and wrr scheduling algorithms have near the same performance before the number of requests exceeds 510. After that, the disparities of the response time between them will become more obvious. The performance of the ANN is good when the number of requests is large. However, we found the average response time of the ANN-based algorithm is the worst and varies greatly before the average response time exceeds 80% (pre-reaction rate) of the response time, specified in the SLA. This is because the ANN-based algorithm will continue to distribute requests to a VM when the response time does not exceed 80% of the response time specified in the SLA. No matter what the number of requests is, the performance of

the proposed nn-dwrr is always the best. Figure 8 shows that the proposed nn-dwrr is 1.86 times faster than the wrr, 1.49 times faster than the Capacity-based, and 1.21 times faster than the ANN-based scheduling algorithms in terms of average response time.

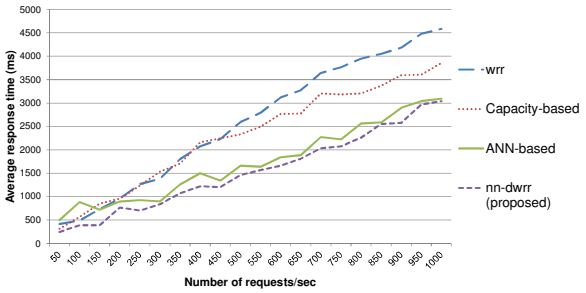

Figure 7. Comparison of four scheduling algorithms in terms of average response time.

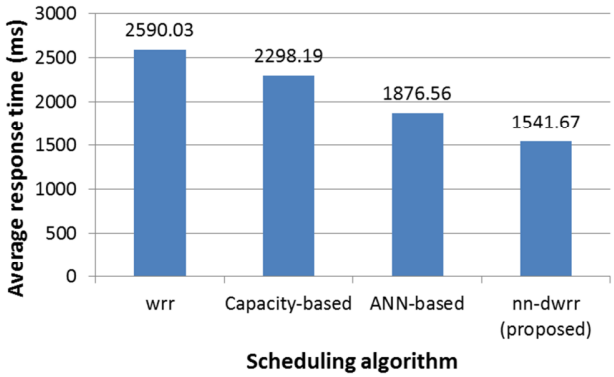

Figure 8. Average response time for the four scheduling algorithms.

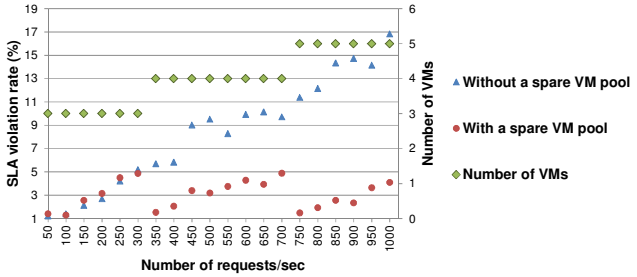

Figure 9. Comparison of SLA violation rates with and without a spare VM pool.

Figure 9 shows the comparison of the SLA violation rate with and without a spare VM pool in the proposed *tldlb* architecture, both running the proposed *nn-dwrr* algorithm. In this experiment, the threshold of the SLA violation rate was set to 5%. The SLA violation rate is defined as follows:

$$\text{SLA violation rate} = \frac{\text{Number of requests violated}}{\text{Number of total requests}}$$

The SLA Engine, as shown in Figure 4, will keep monitoring the response time of each request and then calculate the SLA violation rate. The SLA Engine would activate a spare VM when the SLA violation rate exceeds the threshold (5%, in this case). We found that the proposed *tldlb* can avoid exceeding the SLA violation rate of 5% by activating VMs from the spare VM pool. The proposed *tldlb* can indeed reduce the SLA violation rate by activating VMs in the spare VM pool in time.

V. CONCLUSION

We have presented an SLA-aware decentralized load balancer architecture, tldlb, which can reduce the SLA violation rate. If active VMs are overloaded, the proposed tldlb avoids SLA violations by activating VMs in a spare VM pool. In addition, we also proposed a novel neural network-based load balancing algorithm, nn-dwrr, to distribute incoming requests to appropriate VMs. Experimental results have shown that the proposed nn-dwrr is 1.86 times faster than the wrr, 1.49 times faster than the Capacity-based, and 1.21 times faster than the ANN-based load balancing algorithms, in terms of average response time. Since our load balancing algorithm is simple and efficient, it is well-suited for cloud computing environments to service more requests with less response time.

ACKNOWLEDGEMENTS

The support by the Inventec under Contracts 100C202 and 101C179 and by the National Science Council under Grants NSC99-2221-E-009-081-MY3, NSC101-2219-E-009-001 and NSC102-2221-E-009-090-MY3 is grateful acknowledged. The authors would like to thank Mr. Jonz Lee from the Inventec for his valuable comments that helped improve the quality of this paper.

REFERENCES

[1] "Cloud Computing – Wiki," [Online]. Available: http://en.wikipedia.org/wiki/Cloud_computing/.

[2] "Amazon S3 Growth," [Online]. Available: http://www.datacenterknowledge.com/wp-content/uploads/2011/01/amazon-s3_growth_2010.jpg/.

[3] Z. Zhang and X. Zhang, "A Load Balancing Mechanism Based on Ant Colony and Complex Network Theory in Open Cloud Computing Federation," in the *Proceedings of International Conference on Industrial Mechatronics and Automation*, pp. 240-243, 2010.

[4] W. Y. Lin, G. Y. Lin, and H. Y. Wei, "Dynamic Auction Mechanism for Cloud Resource Allocation," in the *Proceedings of Cluster, Cloud and Grid Computing*, pp. 591-592, 2010.

[5] "Amazon Elastic Load Balancing," [Online]. Available: http://aws.amazon.com/elasticloadbalancing.

[6] "Rackspace - Cloud Load Balancers On-Demand," [Online]. Available: http://www.rackspace.com/cloud/cloud_hosting_products/loadbalancers.

[7] "Service-Level Agreement – Wiki," [Online]. Available: http://en.wikipedia.org/wiki/Service-level_agreement.

[8] R. Rajavel, "De-Centralized Load Balancing for the Computational Grid Environment," in the *Proceedings of International Conference on Communication and Computational Intelligence*, pp. 419-424, Dec. 2010.

[9] S. C. Wang, K. Q. Yan, W. P. Liao, and S. S. Wang, "Towards a Load Balancing in a Three-level Cloud Computing Network," in the *Proceeding of the IEEE International Conference on Computer Science and Information Technology*, vol. 1, pp. 108-113, July 2010.

[10] "Linux Virtual Server," [Online]. Available: http://www.linuxvirtualserver.org.

[11] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing," in the *Proceeding of the IEEE International Conference on Advanced Information Networking and Applications Workshops,* pp. 551-556, Apr. 2010.

[12] T. L. Pao, and J. B. Chen, "Remaining Capacity Based Load Balancing Architecture for Heterogeneous Web Server System," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 58-63, 2006.

[13] V. Nae, A. Iosup, and R. Prodan, "Dynamic Resource Provisioning in Massively Multiplayer Online Games," *IEEE Transactions on Parallel and Distributed Systems,* vol. 22, no. 3, pp. 380-395, Mar. 2011.

[14] Y. Zhang, J. Pang, R. Zhao, and Z. Guo, "Artificial Neural Network for Decision of Software Maliciousness", in *Proceedings of Intelligent Computing and Intelligent Systems*, pp. 622-625, 2010.

[15] J. Hu, J. Gu, G. Sun, and T. Zhao, "A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment," in *Proceedings of International Symposium on Parallel Architectures, Algorithms and Programming*, pp. 89-96, Dec. 2010.