2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress

A Two-Level Load Balancing Method With Dynamic Strategy for Cloud Computing

Yan-Wun Qiu, Jen-Ing G. Hwang
Dept. of Computer Science and Information Engineering
Fu Jen Catholic University
New Taipei City, Taiwan

***Abstract*—By providing computing resources to users on demand, cloud computing has brought convenient services to people's lives. However, there remain some challenging problems such as load balancing. This paper presents a dynamic two-level scheduling method for cloud balancing. The proposed method not only focuses on task scheduling, but also considers resource utilization. At Level 1, virtual machines (VMs) are added or deleted dynamically according to the workload. At Level 2, an appropriate mapping between the requested tasks and the VMs is determined. This two-level scheduling method was implemented using a simulation software package, CloudSim. Several possible scenarios were planned and simulated. The results showed that the proposed method attained acceptable performance on the measures of response time and resource utilization. This confirmed the efficiency and effectiveness of the proposed method.**

Keywords—Cloud computing; load balancing; resource utilization; task scheduling

I. INTRODUCTION

Over the past decade, cloud computing has emerged as a crucial type of information technology. Not only does cloud computing integrate network services, but it enables users to access those network services conveniently. Before cloud computing, enterprises invested money on hardware upgrades and time on software installation. By contrast, cloud computing requires no large investment in hardware, and software can be used immediately without installation delays.

Cloud computing helps both enterprises and individual users. Enterprises can lower their expenses by using cloud computing, and they may even increase their profits. For individual users, cloud services enable convenient access to critical services. A typical individual's routine computer use frequently relies on cloud services for tasks such as e-mail, Google searches, and e-commerce transactions. Therefore, cloud services are necessary, even indispensable, for many people.

A cloud's physical layer consists of numerous physical servers that connect each other to provide large-scale distributed computing. Cloud computing marshals vast amounts of computing resources so that cloud applications can process user requests immediately. A cloud's virtualization layer defines virtual machines (VMs) that can process users' requests. A cloud service is offered on a pay-per-use basis. Cloud services include software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Users choose the services that they need. For example, an e-banking service that enables transactions over the Internet is classified as SaaS; a service that leases an OS platform to software engineers is classified as PaaS; and a service that provides cloud infrastructure is classified as IaaS.

Cloud computing adds value in all the aforementioned ways, and has some additional advantages; however, some problems must still be solved. For example, load balancing is a challenging problem associated with cloud systems [1]. A cloud environment must handle unpredictable requests in a timely manner, even though traffic fluctuates between peak times and off-peak times. The cloud is a large-scale and complex environment that includes numerous enterprise services. The challenges of unpredictable user requests and the complexities of the cloud environment may imbalance the load. Hence, an appropriate mapping between resources and user requests is critical. Task scheduling is vital for resource allocation and load balancing in dynamic cloud environments; therefore, this paper presents a two-level scheduling method to solve the cloud load-balancing problem.

Most previous articles on task scheduling for load balancing have discussed effective strategies for task scheduling, but few have considered how to improve resource utilization. Therefore, the load balancing method proposed in the present study not only addresses task scheduling, but also considers how to increase resource utilization. The dynamic load balancing method was implemented using a cloud simulator, CloudSim [2]–[5].

The subsequent sections are as follows. Section II reviews related works and introduces CloudSim. Section III presents the proposed method. The experimental environment, configuration settings, and simulation results are presented in Section IV. Finally, Section V concludes the paper and describes future work.

II. RELATED WORK AND CLOUDSIM

This section reviews the literature on load-balancing methods and briefly describes CloudSim.

A. Related Research

A cloud system provides substantial computing capacity for users, but the challenging problem of load balancing remains to be solved. An imbalanced load causes some processing units to be overloaded while other processing units stand idle. The efficiency of the processing units decreases when resources are inefficiently utilized. Previous studies have proposed various methods to overcome this problem. Some artificial intelligence techniques for task scheduling include genetic algorithms [6], ant colony optimization [7,8], and honey bee methods [9]. Other useful methods include random sampling [10] to achieve faster

978-1-5090-4065-0/16 $31.00 © 2016 IEEE
DOI 10.1109/DASC-PICom-DataCom-CyberSciTec.2016.107

computing time. Studies such as [11]–[13] have focused on improving CPU efficiency, whereas [14] considered the coordination of multiple resources such as CPU, memory, and bandwidth.

Most load-balancing methods focus on allocating resources according to the loading of processing units, but they rarely consider how to enhance resource utilization. Therefore, this paper presents a two-level load-balancing method to improve processing efficiency and resource utilization. Level 1 of the proposed method handles the addition and deletion of VMs; Level 2 determines an appropriate mapping between tasks and VMs to increase processing efficiency. The details of the proposed method are discussed in Section III.

B. *CloudSim*

CloudSim is a toolkit for simulations in cloud computing environments. CloudSim provides several critical classes for representing data centers, VMs, applications, users, cloud computing resources, and policies for scheduling tasks and allocating resources.

Fig. 1 shows a work style of CloudSim [7] and illustrates the relationships between entities (e.g., Datacenter, Host, VM, Task, etc.). In general, there are *m* relatively independent users, *n* independent tasks, *k* VMs, and *p* datacenters.

Fig. 2 illustrates a simulated data flow among the entities of a CloudSim model [4]. At the beginning, each datacenter must register with the cloud information service (CIS) Registry (see Figs. 1 and 2), which records the information of that datacenter.

A DatacenterBroker (see Figs. 1 and 2) represents a broker who acts on behalf of the user to request an available cloud resource from the CIS Registry. The DatacenterBroker first negotiates with the CIS Registry, and then a Datacenter is mapped to the user to serve the request.

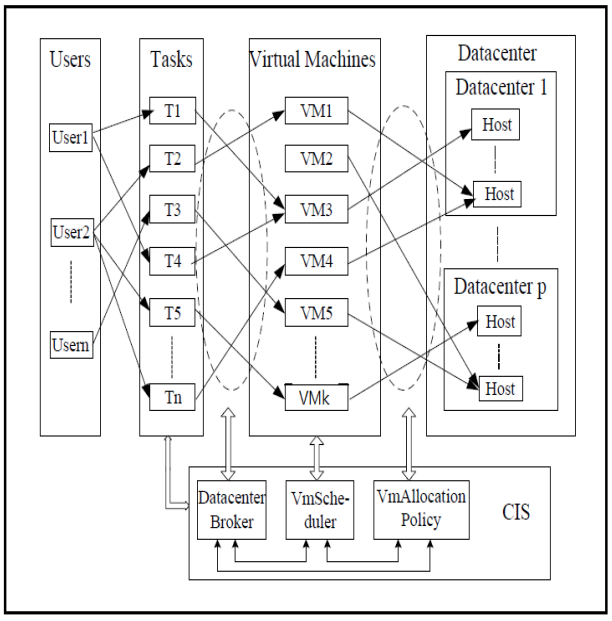


Fig. 1. CloudSim work style, adapted from [7].

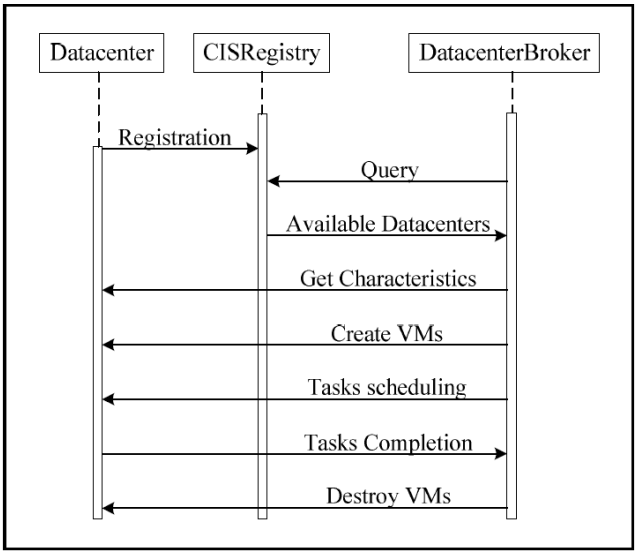


Fig. 2. Simulation data flow [4].

After obtaining characteristics from the Datacenter, the DatacenterBroker creates a VM and schedules the task. When the task is completed, the Datacenter reports the results to the DatacenterBroker, and then updates its information with the DatacenterBroker to facilitate the next task mapping. After all the user requests have been completed, the datacenter destroys all the VMs.

III. PROPOSED METHOD

This paper focuses on allocating resources and scheduling tasks for load balancing. In a cloud environment, the number of tasks fluctuates between peak times and off-peak times, and task requirements are diverse. Hence, we present a two-level strategy for improving response time and resource utilization. The remainder of this section introduces the concept of the two-level strategy and details the proposed algorithm.

A. *Two-Level Schedule Strategy*

The proposed method uses Host, VM, and Task objects to construct a two-level schedule strategy. As shown in Fig. 3, the Datacenter records the information and implements a strategy specified by the superuser, the physical servers provide computing capacity, and the VMs manage user requests. In the present CloudSim environment, the physical servers are represented by Host objects and the client requests are represented by Cloudlet objects.

Level 1 of the scheduler operates between Host and VM objects, and Level 2 operates between VM and Task objects. Level 1 handles adding and deleting VMs according to the task loading ratio; that is, the ratio of the number of utilized VMs to the total number of VMs. When the number of tasks grows rapidly and the workload increases, VMs are created dynamically to avoid overload. When the number of task requests declines, VMs are deleted dynamically to avoid wasting resources.

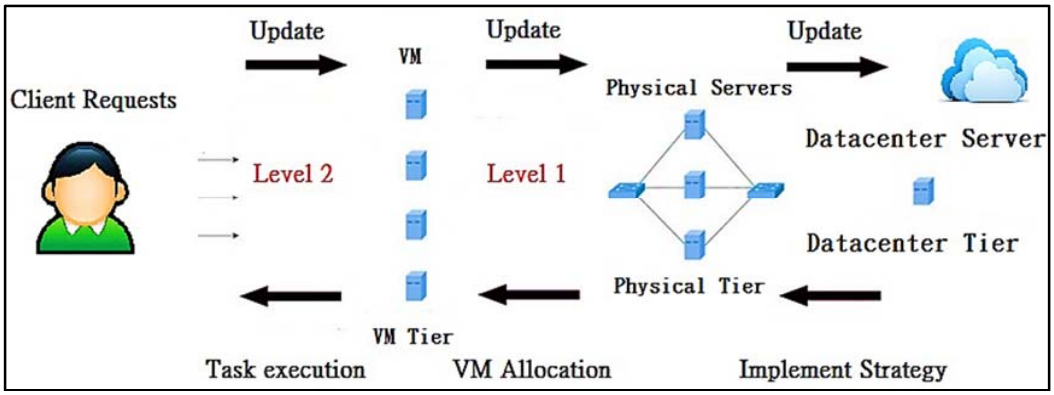


Fig. 3. Construct of two-level strategy.

Level 2 defines a mapping between Tasks and VMs. The following example, which is depicted in Fig. 4, illustrates how crucial a mapping can be. There are two VMs and two Tasks. The capacities of VMs 1 and 2 are 1000 and 100 MIPS, respectively. The lengths of Tasks 1 and 2 are 10 000 and 500 MI, respectively.

There are two mappings between the sets of VMs and Tasks. The first mapping is that Task 1 is assigned to VM 1 and Task 2 is assigned to VM 2. In this case, the makespan will be 10 s and the average response time will be 7.5 s. The second mapping is that Task 1 is assigned to VM 2 and Task 2 is assigned to VM 1. In this case, the makespan will be 100 s and the average response time will be 50.25 s. The makespan and average response time in the first mapping are shorter than those in the second mapping.

This example shows how an appropriate mapping between Tasks and VMs can improve processing efficiency. A larger task should be assigned to a more powerful VM, and a smaller task should be assigned to a less powerful VM. This leads our proposed method to use a greedy strategy for determining a mapping. In other words, tasks are generally stored in a queue with a first-come-first-served (FCFS) service discipline and then, based on their computational requirements, they are assigned to some available VM with minimum capacity.

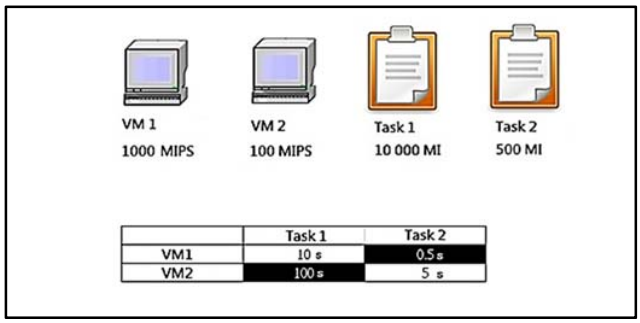


Fig. 4. Mapped strategy between tasks and virtual machines.

However, when none of the existing VMs satisfies the request for a certain task and then creating a new VM may be impossible due to overloaded servers, this certain task must wait in the queue. The details of the proposed approach is described in the following subsection.

B. *Proposed Algorithm*

This subsection describes how the proposed two-level scheduling method maps a task to an appropriate VM and how it dynamically adds or deletes VMs for unpredictable task requests.

Assume that C is an ordered list that stores the requested Tasks based on their arrival times. Let V be a set that contains all of the existing VMs, and let H be a set that contains all of the Hosts. A host is called Predicted-Underloaded-Host if its loading is less than a user-defined parameter threshold, q, after adding a VM. In our experiments, the threshold q was set to 80%. The details of the algorithm are as follows:

Step 1 is to initialize the parameters and build the Datacenters, Hosts, VMs, and Cloudlets. Because both Datacenters and Hosts are at the physical tier, both numbers of Datacenters and Hosts are assumed to be fixed and unchanged for the entire process.

Steps 2–7 involve performing request tasks stored in the ordered list C. The data structure of C is a queue with a FCFS service discipline.

Step 3 performs the second level of the schedule algorithm, an appropriate mapping between a requested task and some available VM with minimum capacity is determined. This step can be described in more detail as follows. Whenever a Task is requested, the second level of the scheduler assigns this Task to the available VM with the lowest capacity that can satisfy the Task. If no such VM exists, then the scheduler creates a new VM that has the capacity to execute this Task. However, if all the Hosts are overloaded, then the VM will not be created, and the Task will stay and wait at the front of the ordered list C. In summary, the second level of the scheduler operates between VM and Task objects, and the proposed method determines an appropriate mapping between Tasks and VMs to increase processing efficiency.

Steps 4 and 5 perform the first level of the schedule algorithm, VMs are added or deleted dynamically according to the workload. In Step 4, if all of the VMs are occupied, then to avoid the possibility that a future Task may not find an available VM, the first level of the scheduler creates a VM. In Step 5, the first level of the scheduler deletes any VMs with idle times larger than an idle threshold Y whose value is determined by the superuser. In our experiments, the idle threshold Y was set to 100 s. In summary, the first level of the scheduler operates between Host and VM objects, and the proposed method provides the mechanism to increase resource utilization by adjusting the number of VMs dynamically.

In Step 6, all the information in the Datacenter is updated, such as remaining tasks in C and current VMs in V.

Two-Level Scheduler Algorithm

Step1. Initialize the parameters and build the Datacenters, Hosts, VMs, and Cloudlets.

Step2. **While** (C is not empty) **DO** Steps 3-7

// Step 3 performs the second level of the schedule algorithm, an appropriate mapping between a requested task (Cloudlet) and some available VM with minimum capacity is determined.

Step3. **If** (There is any available VM in V)
Allocate the first Cloudlet to the VM with minimum capacity.
Remove the first Cloudlet from C.
Else if (There exists a Predicted-Underloaded-Host in H)
Create a proper type of VM.
Allocate the first Cloudlet to the VM.
Remove the first Cloudlet from C.
// The first Cloudlet waits at the beginning of C, if none of the Hosts is Predicted-Underloaded-Host.

// Steps 4 and 5 perform the first level of the schedule algorithm, VMs are added or deleted dynamically according to the workload.

Step4. // Check the status of Cloudlets and VMs.
If (The No. of currently executed Cloudlets = The No. of the existing VMs) and (There exists a Predicted-Underloaded-Host in H)
Create a new VM.

Step5. //Check idle VMs.
Destroy and remove any VMs from V if these VMs are idle.

Step6. //Update the information.
Update set V, set H and ordered list C.

Step7. Go to Step 2.

Step8. Output results and destroy all VMs.

Fig. 5. Two-Level scheduler algorithm.

Steps 2–7 are repeated if any Task exists in C. Otherwise execute Step 8 to output results and destroy all VMs.

Fig. 5 depicts the proposed algorithm.

IV. EXPERIMENT

This section presents the experimental results. The proposed method was implemented using CloudSim, the basic concepts and architecture of which are introduced in Section II.

A. Simulation Setting in CloudSim

This subsection presents the simulation coded in CloudSim. Several VMs were initialized to process tasks at the beginning of each simulation. The proposed method involved adjusting the number of VMs dynamically. The dynamic mechanism added or deleted the VMs when user requests were growing or decreasing.

Table I presents the parameter settings of entities (Datacenter, Host, VM, and Task) in the CloudSim simulations. Both the number of Datacenters and number of Hosts were fixed in each simulation. One Datacenter and five Hosts were used in the entire process of each simulation. The parameters of each Host were as follows: 12 CPU cores within each Host; 1024 MIPS capacity for each CPU; 24 576 MB of memory for each Host; and 20 Mbits of bandwidth for each Host. There were two groups of VM entities. One group contained more powerful VMs with two CPU cores that shared 4096 MB of memory; each core had 1024 MIPS of CPU capacity. The other group had less powerful VMs, each of which had only one CPU core, 512 MIPS of CPU capacity, and 2048 MB of memory. All the VMs had 1 Mbit of bandwidth. Three more powerful and two less powerful VMs were created at the beginning of each simulation. The operation style was SpaceShared and the threshold of idle time was 100 s for each VM. The SpaceShared policy allocated specific CPU cores to specific VMs.

TABLE I.
VALUES OF ENTITIES IN EACH SIMULATION

Entities	Parameters	Values
Datacenters	Number of Datacenters	1
Hosts	Number of Hosts	5
	CPU Capacity	1024 MIPS
	Memory	24 576 MB
	Bandwidth	20 Mbits
	Number of Cores	12
VMs	CPU capacity	512 / 1024 MIPS
	Memory	2048 / 4096 MB
	Bandwidth	1Mbit
	Operation Style	SpaceShared
	Number of Cores	1 or 2
	Idle Time Threshold (= Y)	100 s
Tasks	Total Number of Tasks	100 / 400 / 700
	Length of Task	10 000 / 20 000 / 30 000 MI
	Number of Cores Requirement	1 or 2

Each experiment involved three test runs with a different number of Tasks (Cloudlets): 100, 400, and 700. Each test contained a uniform distribution of three different lengths of tasks: 10 000, 20 000, and 30 000 MI. For example, the first test of each experiment involved 100 tasks, of which 34 tasks had a length of 10 000 MI, 33 tasks had a length of 20 000 MI, and 33 tasks had a length of 30 000 MI. In addition, the three task types required a distinct number of CPU cores. The tasks with a length of 10 000 and 20 000 MI requested only one core; that with a length of 30 000 MI requested two cores.

B. Experimental Results

This subsection presents the experimental results to verify the effectiveness of the proposed method. The experiments were divided into the following two categories:

1) Compare the proposed scheme with various round-robin methods
2) Investigate the number of VMs used in the proposed method

Because the number and distribution of user requests are not predictable in cloud computing, each category of experiments involved simulating three cases to represent three unique situations:

Case 1: User requests are distributed periodically. In this simulation, the server receives a request every 5 s. This case simulates a server that receives user requests regularly.

Case 2: User requests are distributed randomly. In this simulation, the server randomly receives a request every 1–10 s. This simulates a server that receives user requests unpredictably.

Case 3: The distribution of user requests is divided into three stages and the server receives the same amount of requests in each stage. The server receives a request every 5 s in Stage 1, every 1 s in Stage 2, and every 50 s in Stage 3. Note that the number of user requests increases rapidly in Stage 2 and declines rapidly in Stage 3.

The first group of experiments involved evaluating the performance of the proposed method and three round-robin schemes (RR-5VM, RR-15VM, and RR-25VM) in terms of response time and resource utilization. RR-5VM, RR-15VM, and RR-25VM had 5, 15, and 25 VMs, respectively. Each round-robin technique was a static method with a fixed number of VMs.

Each case was tested by three runs with a different number of Tasks: 100, 400, and 700; the corresponding response times are shown in Figs. 6, 7, and 8. All experimental observations in these figures demonstrate that the response times were reduced as the number of VMs increased for the round-robin techniques. For example, the response time of RR-VM25 was 44 s, RR-VM15 was 57 s, and RR-VM5 was 257 s for the 100-task test run for Case 1 (Fig. 6). This phenomenon occurred for each test run with different numbers of Tasks in all three experimental cases (Figs. 6–8).

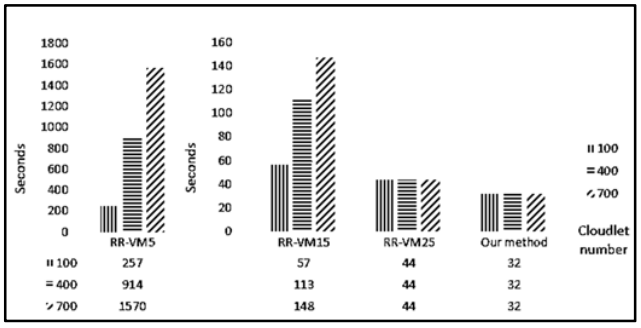


Fig. 6. Average Response time for Case 1.

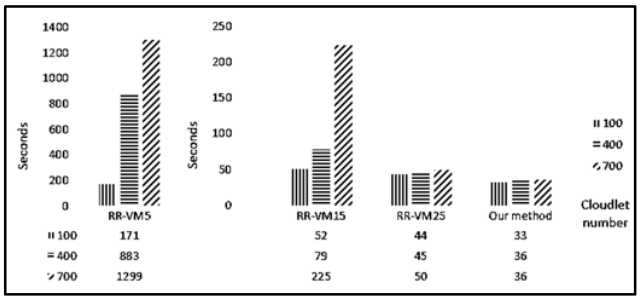


Fig. 7. Average response time for Case 2.

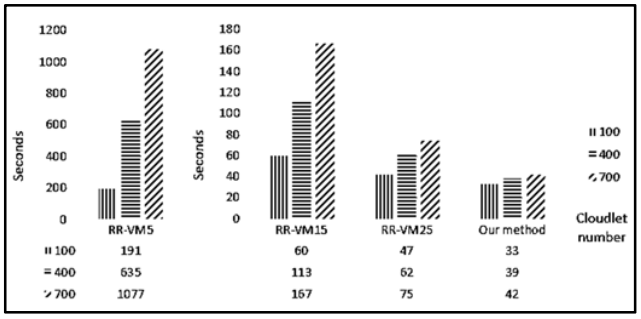


Fig. 8. Average response time for Case 3.

Obviously, a limited number of VMs cannot handle an extremely high load of Tasks. Some Tasks must wait in a queue for available VMs, hence the increase in the response times. Among all the round-robin methods, only RR-VM25 performed satisfactorily. However, the proposed method was superior to RR-VM25 for each test run within each case.

Figs. 9–11 present the resource utilization performance for the three cases, with the results of the three test runs (100, 400, and 700 Tasks) shown in the respective figures.

In Fig. 9, RR-VM5 attained the optimal resource utilization rate, because the VMs were too busy to be idle in Case 1. RR-VM15 and RR-VM25 had more computational capacity for handling the tasks, but both of their resource utilization rates were less than that of RR-VM5 in Case 1. However, the response time of RR-VM5 was the worst (see Fig. 6). Although the resource utilization rate of the proposed method was ranked

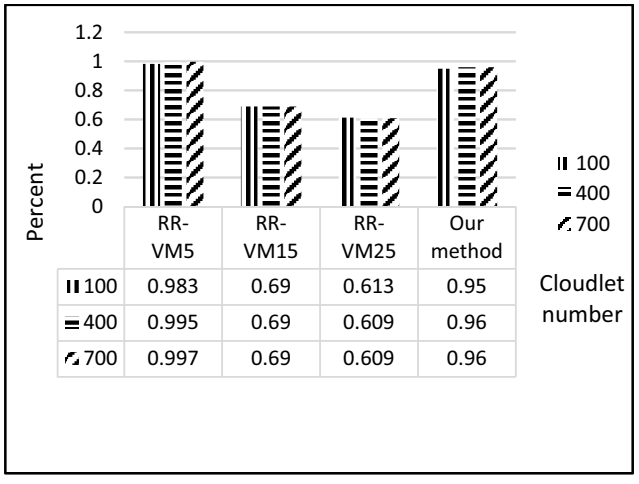
second, the response time of the proposed method was the shortest (see Fig. 6).

Fig. 10 presents the experimental results for Case 2, showing that the resource utilization in this case is similar to that in Case 1. Thus, RR-VM5 attained the optimal performance, followed by the proposed method, RR-VM15, and RR-VM25. Notably, the response time of RR-VM5 was again the worst (see Fig. 7). In summary, the response time performance of the proposed method was superior to that of the other methods, and the resource utilization was ranked second for both Cases 1 and 2.

Fig. 11 presents the experimental results for Case 3. The proposed method attained the optimal performance for all tests; all Task loads and resource utilization rates were higher than 75%. The reason is that the proposed method adds or deletes VMs dynamically according to the number of Tasks; however, round-robin techniques, which involve a fixed number of VMs, are a static method. The results confirmed the effectiveness of the proposed method for Case 3.

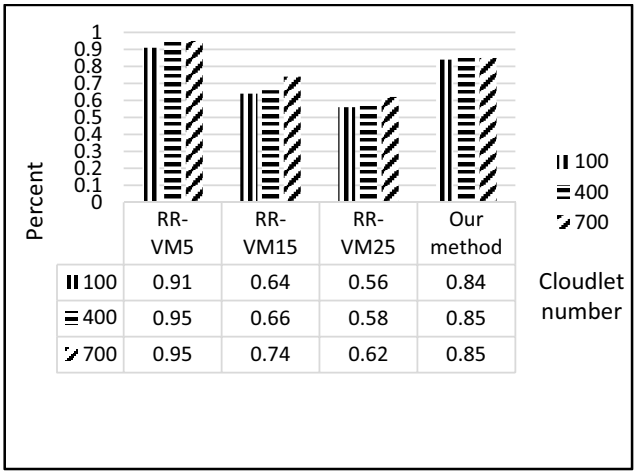
The second category of experiments tested the dynamic allocation of the proposed method. Each case used 700 tasks and utilized a number of VMs confined to a fixed range. Fig. 12 presents the results of three cases representative of the three situations. The number of VMs ranged between 5 and 13 in Case 1 (periodically distributed task requests). The number of VMs ranged between 5 and 30 in Case 2, because the random distribution of sending requests increased the maximum number of VMs to 30. In Case 3, the number of VMs ranged between 2 and 47. Recall that there are three stages to send tasks in Case 3; the number of VMs had to be increased to 47 when the number of user requests increased rapidly; the number of VMs was reduced to two when the number of user requests dropped markedly.

According to these experimental results, the proposed two-level scheduling method performed satisfactorily and was able to handle various situations, even a complex scenario such as Case 3.



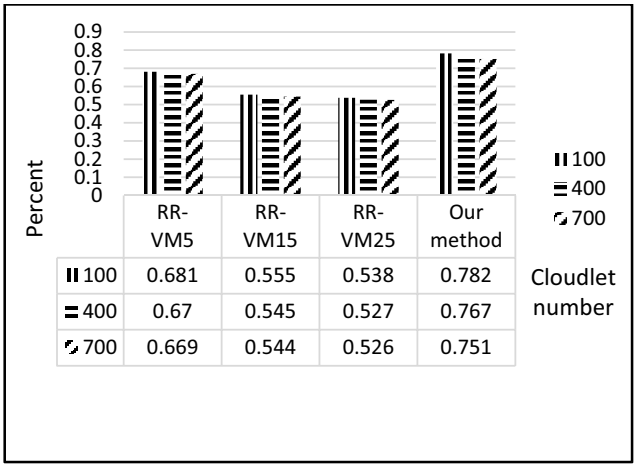
	RR-VM5	RR-VM15	RR-VM25	Our method
100	0.983	0.69	0.613	0.95
400	0.995	0.69	0.609	0.96
700	0.997	0.69	0.609	0.96

Fig. 9. Resource utilization for Case 1.



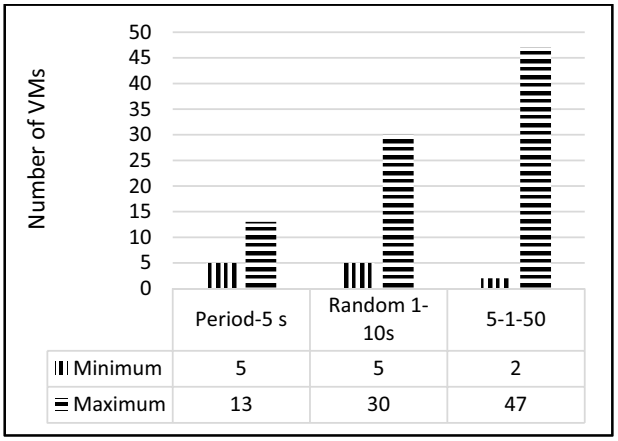
	RR-VM5	RR-VM15	RR-VM25	Our method
100	0.91	0.64	0.56	0.84
400	0.95	0.66	0.58	0.85
700	0.95	0.74	0.62	0.85

Fig. 10. Resource utilization for Case 2.



	RR-VM5	RR-VM15	RR-VM25	Our method
100	0.681	0.555	0.538	0.782
400	0.67	0.545	0.527	0.767
700	0.669	0.544	0.526	0.751

Fig. 11. Resource utilization for Case 3.



	Period-5 s	Random 1-10s	5-1-50
Minimum	5	5	2
Maximum	13	30	47

Fig. 12. Number of VMs with 700 cloudlets in each case.

V. CONCLUSIONS

This paper presents an efficient and effective two-level scheduling method for cloud balancing. The proposed method focuses on improving response time and increasing resource utilization. The two-level method assigns each request to some available VM that has the minimum capacity to satisfy the requirement; more powerful VMs are held in reserve until requests with more demanding requirements arrive. This strategy attempts to reduce the average response time. The proposed method also provides a dynamic mechanism for adding or deleting VMs to increase the resource utilization.

The experiments were implemented using CloudSim. According to the results, the proposed algorithm attained very satisfactory results for two measures (i.e., response time and resource utilization rate) in several simulated scenarios. In the future, the proposed method will be extended to improve handling resources other than CPUs (e.g., memory or bandwidth).

REFERENCES

[1] K.A. Nuaimi, N. Mohamed, M.A. Nuaimi, and J. Al-Jaroodi, "A survey of load balancing in cloud computing: Challenges and algorithms," in Second Symposium on Network Cloud Computing and Applications (NCCA), IEEE, pp. 137-142, 2012.

[2] R.N. Calheiros, R. Ranjan, C.A.F. De Rose, and R. Buyya, "CloudSim: A novel framework for modeling and simulation of cloud computing infrastructures and services," in Technical Report, GRIDS-TR-2009-1, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, 2009.

[3] B. Wickremasinghe, R.N. Calheiros, and R. Buyya, "CloudAnalyst: A CloudSim based visual modeller for analysing cloud computing environments and applications," in 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 446-452, 2010.

[4] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F De Rose and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms." Software: Practice and Experience, vol. 41, pp. 23-50, 2011.

[5] B. Ghalem, F.Z. Tayeb, and W. Zaoui, "Approaches to improve the resources management in the simulator CloudSim," in ICICA 2010, LNCS 6377, pp. 189–196, 2010.

[6] Jinhua Hu, Jianhua Gu, Guofei Sun, Tianhai Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in Third International Symposium on Parallel Architectures, Algorithms and Programming, IEEE, pp. 89-96, 2010.

[7] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang, "Cloud task scheduling based on load balancing ant colony optimization," in Sixth Annual Chinagrid Conference (ChinaGrid), IEEE, pp. 3-9, 2011.

[8] S.H. Li, and J.I.G. Hwang. "Bidirectional ant colony optimization algorithm for cloud load balancing," in Proceedings of the 2nd International Conference on Intelligent Technologies and Engineering Systems (ICITES2013), LNEE 293, pp. 907-913, 2014.

[9] S. Nakrani, and C. Tovey, "On honey bees and dynamic server allocation in internet hosting centers." Adaptive Behavior, Vol. 12(3-4), pp. 223-240, 2004.

[10] O.A. Rahmeh, P. Johnson, and A. Taleb-Bendiab, "A dynamic biased random sampling scheme for scalable and reliable grid networks." INFOCOMP Journal of Computer Science, Vol. 7(4), pp. 1-10, 2008.

[11] Y. Fang, F. Wang, and J. Ge, "A task scheduling algorithm based on load balancing in cloud computing," in Web Information Systems and Mining, LNCS 6318, pp. 271–277, 2010.

[12] 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 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), Vol. 1, pp. 108-113, 2010.

[13] S. Sadhasivam, N. Nagaveni, R. Jayarani, and R.V. Ram, "Design and implementation of an efficient two-level scheduler for cloud computing environment," in 2009 IEEE International Conference on Advances in Recent Technologies in Communication and Computing, pp. 884-886, 2009.

[14] W. Tian, Y. Zhao, Y. Zhong, M.Xu, and C. Jing, "A dynamic and integrated load-balancing scheduling algorithm for Cloud datacenters," in 2011 IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), pp. 311-315, 2011.