Dynamic Task Scheduling in Cloud Computing Based on Greedy Strategy

Liang Ma[1,2], Yueming Lu[1,2], Fangwei Zhang[3], and Songlin Sun[1,2]

[1] School of Information and Communication Engineering,
Beijing University of Posts and Telecommunications, Beijing, China
[2] Key Laboratory of Trustworthy Distributed Computing and Service (BUPT),
Ministry of Education, Beijing, China
mal327@sina.com, {ymlu,slsun}@bupt.edu.cn
[3] School of Humanities, Beijing University of Posts and Telecommunications, Beijing, China
zhangfangwei@bupt.edu.cn

Abstract. Task scheduling is essentially an NP-completeness problem in cloud computing and the existing task scheduling strategies can't fully meet its demands. In this paper, a feasible and flexible dynamic task scheduling scheme *DGS* is proposed, which dynamically allocates virtual resources to execute computing tasks and promptly completes the scheduling and execution process by using improved greedy strategy. The simulation platform CloudSim is expanded to realize the proposed scheme and the simulation results show that *DGS* can speed up the tasks' completion time and improve the utilization of cloud resources to achieve load balance.

Keywords: cloud computing, dynamic task scheduling, greedy strategy, load balance, completion time.

1 Introduction

In cloud computing [1], virtualization technology hides the heterogeneity of the resources. They are no longer physical entities but instead a huge resource pool that consists of abundant virtual machines (VMs).The basic mechanism of cloud computing is to distribute computing tasks to the virtual resource pool, which enables a variety of applications to gain computing power, storage and a variety of software services according to their needs [2]. Thus an appropriate task scheduling strategy is needed as a support.

The traditional scheduling schemes usually assume all computing nodes available for processing which is not reliable in some scenarios[3],such as round-robin algorithm, max-min algorithm, min-min algorithm and least-connection scheduling algorithm [4].So they don't meet the characteristics of cloud computing. Fortunately, many efficient scheduling strategies have been proposed by the major cloud computing vendors such as IBM's Tivoli, Amazon's EC2, Microsoft's Dryad and Google's MapReduce[5]. But a uniform standard to evaluate the methods is yet to be achieved. Task scheduling in cloud computing is essentially an NP-completeness problem while heuristic intelligent scheduling algorithms have been quite mature in seeking the optimal

Y. Yuan, X. Wu, and Y. Lu (Eds.): ISCTCS 2012, CCIS 320, pp. 156–162, 2013.

solution, such as genetic algorithm, simulated annealing algorithm, ant colony algorithm and so on [6].But with the growing number of tasks and resources, the complexity of these algorithms will become very high.

The current research of dynamic resource allocation and task scheduling mainly focused on dividing onerous tasks into multiple subtasks for optimal solution and migrating VMs for load balance. But in fact the divided subtasks usually can't conduct parallel computing due to strong backward and forward linkages. Furthermore, cloud platform may be suspended because of the mandatory shutdown of the associated VMs during migration process [7].

To address the problems, *DGS* is proposed which is described in detail in section 2.It monitors and calculates the actual amount of resources required by applications, dynamically adjusts virtual resources to increase resource utilization and achieve load balance. Meanwhile, improved greedy strategy is used to dynamically distribute tasks to appropriate computing nodes in order to respond to users quickly. We also conduct performance studies in CloudSim [8] environment and compare the performance of *DGS* with round-robin algorithm and min-min algorithm in section 3.

2 A Feasible and Flexible Dynamic Task Scheduling Scheme *DGS*

According to the above explanation, *DGS* mainly focused on dynamically allocating VMs and distributing tasks by greedy strategy. In order to realize *DGS*, a task scheduling node is divided into four modules to carry on scheduling process, including Service Request Module (SRM), VMs Monitoring and Managing Module (VMM), Routing Analysis Module (RAM) and Task Scheduling Module (TSM).

2.1 Quantification of Tasks and VMs

When the cloud receive the requests of users, TSM will firstly quantify them to computable tasks by the following data structure.

$$\begin{aligned} &Task\{TaskId,\ TaskIP,\ Task\ Load,\ Computing\ Capacity,\\ &Communication\ Traffic,\ Storage\ Content\}; \end{aligned} \tag{1}$$

In (1), *Computing Capacity, Communication Traffic* and *Storage Content* indicate the requirements of each task to the *Computing Power*, *Network Throughput* and *Storage Capacity* of a VM. Corresponding to it, VMs will also be quantified like this.

$$\begin{aligned} &VM\{VM\ Id,\ VM\ IP,\ CPU,\ Memory,\ Computing\ Power,\\ &Network\ Throughput,\ Storage\ Capacity,\ Running\ Task\}; \end{aligned} \tag{2}$$

In this paper, we define the completion time of a task executing on a VM like formula (3).

$$\begin{aligned} Completion\ time = &\ Computing\ Capacity\ /\ Computing\ Power\\ &+ Communication\ Traffic\ /\ Network\ Throughput; \end{aligned} \tag{3}$$

Assuming the load of a task is L_{task} and the maximum workload of a VM is L_{max}, then the total load of users' requests is L_{total} and the maximum workload of these existed VMs is $L_{bare.}$ Assuming the current workload of a VM is L_{normal}, we define two load factors $Rate_{total}$ and $Rate_{normal}$ to describe the load condition of the whole system and

each VM respectively. At the same time, a variable $Rate_{threshold}$ is defined to indicate the threshold rate which is the main criterion to identify whether the system or a VM is overloaded. It's an empirical value here.

$$L_{total} = \sum L_{task}; \tag{4}$$

$$L_{bare} = \sum L_{max}; \tag{5}$$

$$Rate_{total} = L_{total} / L_{bare}; \tag{6}$$

$$Rate_{normal} = L_{normal} / L_{max}; \tag{7}$$

When the problem of load balance is discussed, the load of a server L is often calculated by formula (8), where L_{cpu}, L_{memory}, $L_{network}$ and $L_{storage}$ mean the load of *CPU*, *Memory*, *Network Throughput* and *Storage Capacity*; $\alpha, \beta, \gamma, \delta \in [0,1]$ and $|\alpha + \beta + \gamma + \delta| = 1$, they represent different weights according to the types of applications.

$$L = \alpha * L_{cpu} + \beta * L_{memory} + \gamma * L_{network} + \delta * L_{storage}; \tag{8}$$

2.2 Dynamic Resource Allocating Method

The main idea of dynamical allocation is to use virtual machine as the minimum resource allocation unit [9]. VMM uses pre-prepared image files with different functions and certain network bandwidth to create new different VMs. As mentioned above, we will dynamically allocate virtual resources in three situations.

First of all, if VMM detects that there is no VM in cloud or the computing power of these VMs is limited, new VMs will be created. It firstly calculates the value of L_{total}, L_{bare} and $Rate_{total}$ through formula (4) (5) (6). Meanwhile the variable $Rate_{threshold}$ here is defined as 0.7to avoid the overload of the whole system when new tasks arrive. If $Rate_{total}$ is smaller than $Rate_{threshold}$, then the existed virtual resources could swimmingly execute the tasks and new VMs don't need to be allocated, or else the number of VMs that we have to create is N so as to ensure the whole network will not overload. $N$is depicted as follows.

$$N = (L_{total} - L_{bare}) / L_{max}; \tag{9}$$

Secondly, after the fist-timescheduling by greedy strategy, if VMM notices some VMs running in the critical state, it will create new VMs and conduct second-time scheduling. In formula (7), we compute each VM's $Rate_{normal}$ and compare it to $Rate_{threshold}$ which is defined as 1.0 here to avoid the downtime of a VM. If $Rate_{normal}$ is more than $Rate_{threshold}$, the VM is overloaded and VMM will create a new and exactly the same one to it except *VM Id*. Then TSM will assign these tasks in the task queue on the overloaded VM to the two VMs in turn, as shown in Fig.1. If it is still overloaded, the above procedure will be repeated.

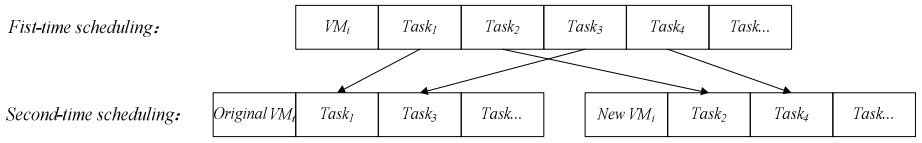

Fig. 1. An example of second-time scheduling

Finally, before executing these assigned tasks, idle VMs will be released to reduce the network load and avoid the waste of resources.

Through the three steps, both the whole network'sworkload and single resource's workload are taken into consideration. In thissituation,resources in cloud will be sufficiently used and network congestion will be avoided.

2.3 Task Scheduling Based on Greedy Strategy

Greedy strategy usually makesthe optimal choice with an optimization measure based on the current situation butregardless of any possible overall situation.It doesn't need to backtrack and the complexity is relatively simple.

Before allocating, TSM will firstly create a Tasks–VMs taboo table, as shown in Fig.4. The left column of the table are tasks which are sorted by computing capacity in decreasing order by SRM and the right column is VMs list.In each row, TSM calls formula (1)(2)(3) to compute the *Completion time*for each task on all VMs and sort the VMs in ascending order.

Table 1. An example of Tasks-VMs taboo table

Tasks	*VMs*				
$Task_2$	VM_2	VM_3	VM_5	VM_1	*VM…*
$Task_1$	VM_4	VM_2	VM_3	VM_5	*VM…*
$Task_3$	VM_3	VM_5	VM_2	VM_1	*VM…*
$Task_4$	VM_2	VM_3	VM_5	VM_1	*VM…*
…	…				

Then we attempt to assign the task to the VM in the first column of virtual machines list from the first line to ensure thattasks with larger computation quantity will be executed on the VMs with more powerfulexecuting ability. Normally, the chosen VM with strong computing power has already gotsome tasks working on it. Sothe completion time of this being currently assigned task should includetheexecution time of these tasks assigned before if it is assigned to the VM.

Then the current program will be compared to others in whichthis task is assigned to other slightly worse VMs.If the current program is optimal compared to other results, the program is feasible, otherwise we have to choose other optimal programs. In this case, the more sophisticated tasks will be preferentially executed to solve time bottleneck caused by themselves.By recording the parameter *Running Task* of a VM, when there are multiple solutions that can achieve optimal, the task will be assigned to the VM which has the lower resource utilization rate.

In this way, itcannot only quickly complete users' requests but also take fully advantage of the current resources existed in cloud so as to implement asimple load balance.

3 Evaluation

In our work, the simulation platform CloudSim is expanded and applied to realize *DGS*.To have a better view of the advantages of *DGS*, a comparison betweenround-robin algorithm

(RR) and min-min algorithm (MM) is conducted.In initial situation, there are four types of tasks and 15VMsin the system.

Fig.3 shows the comparison of three schemes' completion time inhandling different number of tasks.If the downtime of each VM is not considered, thenthe completion time will grow gradually with the increasing of the number of tasks, just like MM and RR in this figure. But in actual experiments, when the number of tasks increases to 120, many VMs have been overloaded and stopped working. Then *DGS* began to create some new VMs to ensure each VM can work normally and its number is 8 here. Due to creating new VMs to execute the tasks on overloaded VMs, the completion time will be shorter. So the completion time of *DGS*will not grow obviously with the increasing of the number of tasks and it is relatively smoother in Fig.2.

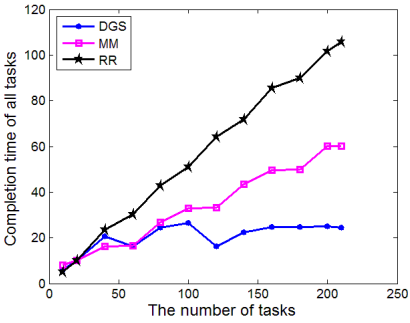

Fig. 2. The comparison of three schemes'

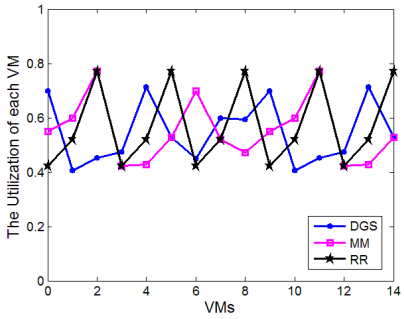

Fig. 3. The comparison of three scheme's completion time resource utilization with 60 tasks

Fig.3 shows the comparison of three schemes' resource utilization when there are 60 tasks. In this situation, both the whole system and each VM are not overloaded. The resource utilization of three schemes is similar in the figure. So *DGS* doesn't have significant advantageswhen the number of tasks is small.

As shown in Fig.4, when the number of tasks reaches 120, many VMs will be overloaded with the use of MM or RR. And here we set theirutilization to 1, just like the second VM in this figure. Thenthe load of the whole system will become extremely unbalanced. At this time,*DGS* creates8 VMs to make sure no VM is overloaded and the utilization of each VM is relatively more balanced.

When the number of tasks increase to 210, the whole system will be overloaded. Then users' requests will not be respond timely either MM or RR is used. But with *DGS* it can work normally no matter what the number of requests is. In this situation, *DGS* will firstly allocate some new VMs to ensure the whole system is able to work and the number here is 5. Then another 5 new VMs are created during the scheduling process to avoid the overload of each VM. As a result, there are 25 VMs here after the accomplishment of scheduling. As shown in Fig.5, the resource utilization of each VM is high and under the critical point.

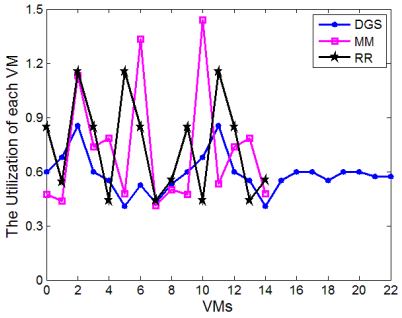

Fig. 4. The comparison of three schemes'

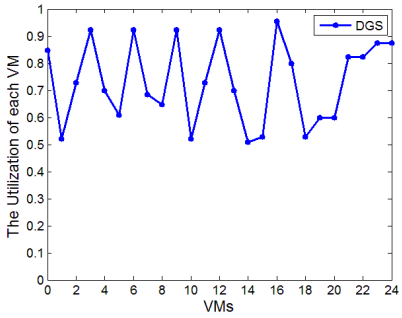

Fig. 5. The resource utilization of DGS resource utilization with 120 tasks with 210 tasks

From the simulation, we can see *DGS* plays a significant role if the number of tasks is large. It can promptly deal with the requests of users and relatively achieve load balance at the same time. With the growing number of tasks, *DGS* will have better-performance and adapt to the characteristics of cloud computing better. But the premise is to consume more hardware resources because more VMs are allocated during the process.

4 Conclusions

In this paper, an efficient dynamic task scheduling scheme based on greedy strategy named *DGS* is proposed to deal with the requests of users as soon as possible on the premise of load balance. Through the expansionofCloudSim, *DGS*has been simulated and the experiment results show that the new scheme can dynamically allocate VMs to reach load balance rate and rapidly finish executing tasks by greedy strategy.Moreover, it's easy to implement in the realistic situation and will achieve great effect. During the research we haven't taken the cost of virtual resources and the prediction of users' requests into consideration.That will be studied in the future research.

Acknowledgments. This research was supported in part by National 863 Program (No. 2011AA01A204), National 973 Program (No. 2011CB302702), P. R. China. Thanks for the great help.

References

1. Armbrust, M., Fox, A., Griffith, R., et al.: Above the Clouds: A Berkeley View of Cloud Computing. Technical Report, No. UCB/EECS-2009-28 (2009)
2. Baomin, X., Chunyan, Z., Enzhao, H., Bin, H.: Job scheduling algorithm based on Berger model in cloud environment. J. Advances in Engineering Software (2011)
3. Xiangzhen, K., Chuang, L., Yixin, J., Wei, Y., Xiaowen, C.: Efficient dynamic task scheduling in virtualized data centers with fuzzy prediction. Journal of Network and Computer Applications 34(4), 1068–1077 (2011)

4. Chauhan, S.S., Joshi, R.C.: A Weighted Mean Time Min-Min Max-Min Selective Scheduling Strategy for Independent Tasks on Grid. In: 2010 IEEE 2nd International Advance Computing Conference on (IACC), pp. 4–9. IEEE Press, Patiala (2010)
5. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM - 50th Anniversary Issue: 1958–2008 51(1), 107–113 (2008)
6. Abdulal, W., Ramachandram, S.: Reliability-Aware Genetic Scheduling Algorithm in Grid Environment. In: 2011 International Conference on Communication Systems and Network Technologies (CSNT), pp. 673–677. IEEE Press, Katra (2011)
7. Warneke, D., Kao, O.: Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud. IEEE Transactions on Parallel and Distributed Systems 22(6), 985–997 (2011)
8. Calheiros, R.N., Ranjan, R., Rose, C.A.F.D., Buyya, R.: CloudSim: A Novel Framework for modeling and Simulation of Cloud Computing Infrastructures and Services. Technical report (2009)
9. Weiwei, L., James, Z.W., Chen, L., Deyu, Q.: A Threshold-based Dynamic Resource Allocation Scheme for Cloud Computing. J. Procedia Engineering 23, 695–703 (2011)