Dependency Based Scheme for Load Balancing in Cloud Environment

Neethu.M.S, Jayalekshmi.S

Abstract*— Cloud computing provides an opportunity to dynamically share the resources among the users through virtualization technology. In this paper, a scheme for load balancing is proposed on the basis of dependency among the tasks. CMS consists of three algorithms including Credit-based scheduling for independent tasks, Migrating Task and Staged Task Migration for dependent tasks. The Credit-based method is used for scheduling the independent tasks considering both user priority and task length. Each task will be assigned a credit based on their task length and its priority. In the actual scheduling of the task, these credits values will be considered. Task Migration algorithm is used to guarantee balancing of loads among the virtual machines. Task migration is done such that the tasks gets migrated from heavily loaded machines to comparatively lighter ones. Thus, no rescheduling is required. For dependent tasks, the dependencies between tasks are considered and the technique termed as data shuffling is used. In data shuffling, a job is divided into several tasks according to the execution order. The method used here is that the tasks in one stage run independently, while the tasks in different stages must be executed serially. Finally the system is simulated and experiments are conducted to evaluate the proposed methods. This work also concentrates on a simulated study among some common scheduling algorithms in cloud computing on the basis of the response times. The algorithms being compared with the work includes: Random, Random Two Choices (R2C) and On-demand algorithms. The evaluations demonstrate that Credit-based scheduling algorithm significantly reduces the response time.

Index Terms*— Load Balancing, Virtual Machine, Task Scheduling, Dependency.

I. INTRODUCTION

Cloud computing is a kind of Internet-based computing, where shared resources, data and information are provided to computers and other devices on-demand. Cloud provides three types of services: Software as Service(SaaS), Platform as Service (PaaS), Infrastructure as Service (IaaS). Cloud computing provides facilities for dynamically accessing the virtualized assets in the form of services. Mainly clouds are of two types: Private and Public.

Cloud solutions are simple and they do not require long term contracts and are easier to scale up and down as per the demand. Perfect planning and migration services are needed to ensure a successful implementation. Both Public and Private Clouds can be deployed together to leverage the best of both.

Revised Version Manuscript Received on August 19, 2016.

Neethu.M.S, PG Student, Department of Computer Science & Engineering, LBSITW, Thiruvananthapuram, India.

Jayalekshmi.S, Associate Professor, Department of Computer Science & Engineering, LBSITW, Thiruvananthapuram, India.

Load balancing is one of the central issues [6] in cloud computing. It is a mechanism that distributes the dynamic local workload evenly across all the nodes in the whole cloud to avoid a situation where some nodes are heavily loaded while others are idle or doing little work. Some of the jobs may be rejected due to the overcrowding. Hence various load-balancing algorithms have been proposed in which live migration of load is done in virtual machines to avoid the under utilization.

Depending on the current state of the virtual machine, load balancing algorithms can be categorized into two types : static and dynamic algorithms. A load balancing algorithm which is dynamic [13] in nature does not consider the previous state or behavior of the system, that is it depends on the current behavior of the system. Static algorithms do not consider the current status of a virtual machines. The static algorithm uses a method where the final selection process of a VM is already predefined and cannot be changed during process execution to make changes in the VM load.

Another classification of load balancing based on the behavior of the algorithm can be of three types :centralized, distributed and hierarchical. In centralized approach, a single node is responsible for managing the whole system. It reduces time but creates great overhead and recovery is difficult. In distributed approach, each node independently builds its own load vector and decisions are made using this. It is widely used for distributed systems.

Hierarchical approach operates in master-slave mode. Based on initiator three types of algorithms are possible : sender initiated, receiver initiated and symmetric. Node with the higher load initiate load balancing in sender-initiated method. At the same time in receiver-initiated, underloaded node initiates load balancing. Symmetric uses the concept of both sender and receiver initiated approaches. So the idea used in symmetric method is that at low system loads sender initiated node is more successful in finding underloaded nodes and at high system loads receiver-initiated component is successful in finding overloaded nodes.

In this work, the various schemes for load balancing are studied and the method of incorporating new schemes into it is explored. The measures used to evaluate the impact of this schemes shows that it is more efficient.

II. PROPOSED SYSTEM

The parallel job [15] scheduling should address two challenges: low response time and job correlation. Response time is one of the most critical issue for a parallel system. Inter-communicated jobs and resource-related jobs are very common in parallel systems. A scheduling algorithm should

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

pay attention to the dependencies of the tasks; tasks dependent on other tasks or system resources have to suspended until the preconditions are satisfied. Task dispatch is the fundamental factor to achieve a good scheduling result.

The proposed scheme was introduced to address one of the challenges faced in scheduling the tasks. The challenge being addressed here is to improve response time. The main sections in this scheme can be summarized as follows:

- For independent tasks, a Credit-based scheduling [12] method is proposed. This approach considers two parameters : Task Length and User Priority. The algorithm is based on credit system. Each task is assigned a credit based on their task length and priority. In the actual scheduling of the task, these credits will be considered.
- A Migrating Task algorithm is used in CMS to guarantee load balance for a virtual machines. The task migration procedure is initiated when an idle VM is reported such that there exists another VM which has worked for a long time. Task migration is a big burden for any parallel system. To deal with this issue, maintain a list of the last tasks dispatched to each VM. Thus, a VM migration helps to reduce the response time.
- For Staged Task Migration (STM) [8] the dependencies among tasks are considered and the method schedule the tasks associated with each other. A job is divided into stages according to tasks execution sequence and a Data Shufing process is used to represent interactions between stages. A task migration loop is also designed in this to guarantee the most balanced workload for each stage.

A. *Credit-based Scheduling of Independent Tasks*

This method[12] considers mainly two parameters: Length of the tasks and Priority of the tasks given by user. The scheduling [11] is based on a credit system such that the each task is assigned a credit value based on their length and priority.

- ***Calculation of task length credit:***

The cloud system execute tasks having different length. If the tasks are arranged based on the increasing order of length, tasks having shorter length will reside at the beginning of the array and the task having highest length will be present at the last. The algorithm [12] for credit calculation on the basis of task length is as follows:

Find the length of each task as TL_t
Calculate the average length of tasks as avg_{len}
Calculate the task length difference for each task, $TLD_t = |avg_{len} - TL_t|$
Calculate the range values: v1=high_len/5
v2=high_len/4
v3= v2+ v1
v4= v3+ v2
for each task T_t in the batch of tasks do
if $TLD_t \leq v1$ then
length_credit=5
else
if v1 $< TLD_t \leq v2$ then
length_credit=4
else
if v2 $< TLD_t \leq v3$ then
length_credit=3
else
if v3 $< TLD_t \leq v4$ then
length_credit=2
else
if v4 $\geq TLD_t$ then
length_credit=1
end if
end for
Return length_credit

Figure.1 Algorithm for calculation of task length credit

For the scheduling purpose, the algorithm should take tasks from both front and back, giving it a bit more stability. The credit system based on task length will work as follows: The first step is involved in finding the length of each task(TL_i). The next step is calculating the average of tasks length. Lets the value is avg_{len}. The third step begins with the calculation of difference in length with respect to avg_{len}. Let the task set be T_1, T_2, T_3, ...etc. Here equation 2 is used for finding the difference in length with the average length.

$$TLD_i = |avg_{len} - TL_i| \quad (1)$$

This data is useful when tasks are arranged in an array in an increasing order of task length. The algorithm calculates the Task Length Difference(TLD) for each tasks in the batch. Range values v1,v2,v3,v4 are calculated such that these values lies between the lowest and highest possible values of the length of the tasks. The length credit value is calculated and the possible value is from 1 to 5. The proposed algorithm neither takes task with larger length nor task with lower length. It takes each tasks from the middle. This method [11] schedules tasks from the middle of the list such that it neither takes task with larger length nor task with lower length.

- ***Calculation of task priority credit:***

Task priority is also important for scheduling tasks. Each task may have different priorities, which are represented as values assigned to each task and the value can be the same for more than one task. The scheduling algorithm based on task priority [12] has the problem of treating tasks with similar priority. In the propose$_i$d approach this does not arise as a problem because even though we are giving credits to each tasks based on their priority, the final scheduling will be based

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

on total credit which is based on task length and its priority.

Suppose there are 5 tasks, then there will be 5 different credits. So we can say that there will be 10 different credits when dealing with 10 tasks. The fact is that these credits [12] are not set by default and hence will change based on the priority that is assigned by the user. The steps can be summarized as follows:

```
for each task T_i where i=1,2...n do
    Find the task with highest priority as T_k
end for
Find a division_factor
for each task T_i with priority T_pri do
    Find Pri_frac(i)=T_pri /division_factor
    Set priority_credit(i) as Pri_frac(i)
end for
```

Figure.2 Algorithm for calculation of task priority credit

The primary step in the above algorithm is finding the highest priority number. Second step in the algorithm is choosing the division factor for finding Pri_frac for each task. For example if highest value of priority is a two digit number then choose division factor as 100. If it is 3 digit then division factor is 1000. Third step in the algorithm is calculating the Pri_frac for each task. This can be calculated by dividing priority value of each task with division factor of corresponding task. Finally this value(Pri_frac) will be assigned to each task as priority credit.

$$\text{Total\_credit}_i = \text{length\_credit}_i \times \text{priority\_credit}_i \quad (2)$$

For each task i, the Total credit$_i$ represents the credit based on both length and priority. In order to dispatch a task to the virtual machine with the least load, traditional approaches utilize the length of task queue to represent the queuing load of virtual machines. That is predicting the waiting time of a task based solely on queue length is typically ineffective. But the total processing time of the queued tasks is unknown. Credit based mechanism can ensure the scheduler to assign the task to an idle slave, it is not necessarily the one with the least load. For the Credit-based scheduling method, its dispatch considers the task in the order after the total credit is calculated. Finally tasks will be scheduled such that those having highest credit value will be scheduled first.

B. Migrating Task

Load balancing removes the situation of large difference in resource usage level by avoiding virtual machines from getting overloaded in the presence of low loaded machines. Live migration [8] can be used to balance the load across the systems. In this method, the tasks that were finally obtained after the calculation of credit will be scheduled in the order of highest credit value and they will be allocated with virtual machines which has the least load at the current time. After scheduling, the status of VMs are checked and the if there is any heavily loaded VM then the task scheduled to that VM can be migrated to another low loaded VM. So fair allocation of the available resources can be satisfied.

In order to dispatch a task to the slave with the least load, traditional approaches utilize the length of task queue to represent the queuing load of slaves. But several studies indicates that predicting the waiting time of a queued task based solely on queue length is typically ineffective. Though the Credit-based mechanism can ensure the scheduler to assign the task to an idle VM, it is not

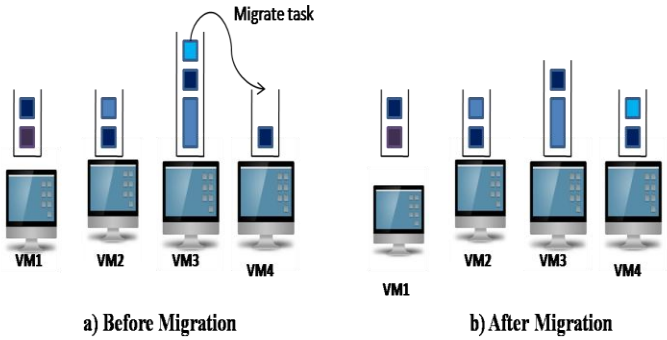

Figure 3. An illustration of scheduling before and after task migration

necessarily the one with the least load. For the Credit-based scheduling method, its dispatch may lead to long response time of task in some cases. Figure 3(a) shows that the scheduler may make the wrong decision because it does know the processing time of the current running task on the VM. On the contrary, if the task is running with heavy workload, the scheduler may make similar mistakes since the queue length cannot represent the real waiting time directly. The algorithm for Migrating Tasks can be summarized as follows:

```
for each virtual machine, VM_i in VM_1,VM_2,...,VM_n do
    Calculate the total length of tasks scheduled to it as VM_i_length
    Calculate the execution time for the last scheduled task for each VM_i as VM_i_T_j_execstart
    and execution finish time as VM_i_T_j_execfinish.
end for
if there exists any VM_k whose VM_k_T_j_execstart <VM_i_T_j_execstart then
    Move the task T_j to VM_k for execution
end if
```

Figure.4 Algorithm for calculation of task priority credit

In short the above algorithm works after scheduling is completed using Credit-based method. All the VMs will have tasks allotted to it. Maintain a list of the last tasks scheduled to each VM. Find the VM which is heavily loaded i.e the VM which has comparatively heavier load than the remaining VMs and let it be VM$_i$ and the task be T$_j$. At the sametime find another VM that is having low load and let it be VM$_k$. Check and make sure that the last scheduled task in VM$_i$ will yield execution start time earlier in VM$_k$ than in VM$_i$ for the same task T$_j$. This is because of the reason that it can reduce the overall response time. If the condition is satisfied, migrate the task from VM$_i$ to VM$_k$.

C. Staged Task Migration for Dependent Tasks

Suppose there are n numbers of dependent tasks (t1, t2, t3 ... tn). The scheduling problem with dependent task [8] is that a child task cannot start its execution until all its parent tasks have finished their execution. The tasks in one stage run

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

independently, while the tasks in different stages must be executed serially. Figure.5 shows the way the tasks are arranged in the order it is to be scheduled.

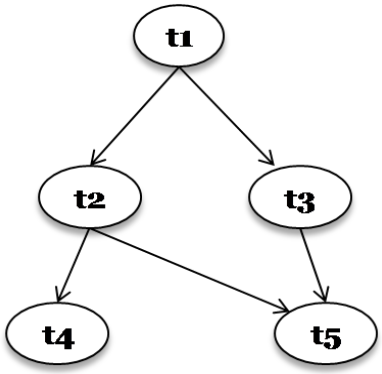

Figure.5 Ordering the dependent tasks for scheduling.

To dispatch tasks with Data Shuffling[8], a queue named Shuffling FIFO is used which holds the tasks in the order to be executed. If this method is not used, then a task in later stage is dispatched to an idle slave before all tasks in current stage finishes its execution. So the execution of tasks gets blocked and thus no other tasks can be processed at that time. This can ultimately lead to increase in response times and lower system performance. Also the task migration will be ineffective when the task to be migrated is blocked. The algorithm for Staged Task Migration below assumes tasks to be in two stages

$T_1=t_{1,1}, t_{1,2} ..... t_{1,m}$ and $T_2=t_{2,1}, t_{2,2} .... t_{2,n}$.

This algorithm called Staged Task Migration (STM) is proposed to guarantee the scheduling and processing performance of dependent tasks. As it is shown in Algorithm , STM consists of two procedures: on task and migration. For a new incoming task in the current stage, the scheduler chooses a VM with low workload and dispatches the task to it. If the task is in the later stage, it is buffered into the FIFO queue for scheduling. The second procedure in STM is termed migration and in this the migration will be detected and processed. Then the queues in slaves will not be changed in the current stage and the optimal scheduling is achieved. At this time, the tasks of the later stage are popped out from the FIFO and dispatched to VMs without waiting for the completion of the tasks in the current stage. Considering figure 3.2 above from tasks t1,t2 and t3, task t1 comes under stage T_1 and tasks t2,t3 comes under stage T_2.

Assume FIFO as a queue holding tasks of second stage T_2

Let T_c be the current stage and initialize $T_c=T_1$

Procedure **on_task(t)**

if $te\ T_c$ **then**

Find a VM which is either having state as idle or comparatively low loaded and let it be denoted by VM_i

Move the task, t to VM_i for execution

else

FIFO.push(t) , task t is pushed to FIFO.

end if

End **on_task(t)**

Procedure **migration()**

for each virtual machine, VM_i in $VM_1,VM_2,...,VM_n$ **do**

Calculate the total length of tasks scheduled to it as VM_i_length

Calculate the execution time for the last scheduled task for each VM_i as VM_i_t_execstart and execution finish time as VM_i_t_execfinish.

end for

if there exists any VM_k whose VM_k_t_execfinish <VM_i_t_execstart **then**

Move the task t to VM_k for execution

end if

End **migration()**

$T_c=T_1$

on_task(FIFO.pop())

Figure.6 Algorithm for dependent task scheduling and migration

III. EXPERIMENTAL EVALUATION

Net Beans IDE (8.1release), Java and CloudSim3.0.3 are used for implementing the system. Java is a highlevel object-oriented programming language which is platform independent and simplified to eliminate features that cause common programming errors. NetBeans is a commonly used Integrated Development Environment (IDE) for Java. Cloudsim [1] is a discrete simulation framework which enables seamless modeling, simulation and experimenting on designing cloud computing infrastructures. Implementation and performance analysis of the algorithms were done by extending the various classes of CloudSim. The simulator provides a test environment for evaluating the assumptions made by the researchers and it is free of cost.

The simulation was designed with 10 physical servers. A server has an Intel E5 CPU which includes 8 physical cores and 64 GB memory. Each server is virtualized into required Virtual Machines (VM), each VM has 1 core, 2 GB memory and Ubuntu Linux Operating system. The system was designed to simulate a total of 100 virtual machines. The task response time consists of three periods of time. They are scheduling time, waiting time and processing time. Thus response time can be said to the difference of the execution

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

finish time and execution start time of a task in a VM.

As part of evaluating the system, comparisons were carried out on Random, Random Two Choices(R2C) and On-Demand schedulings[8] and Credit-based scheduling. The idea of random algorithm is to randomly select VMs to assign the selected jobs. The status of the selected Virtual Machine can be heavy or low load, but this algorithm does not consider This context. Hence, this may result in the selection of a VM under heavy load and the job requires a long waiting time before service is obtained. So the complexity of this algorithm is quite low and the processing is in the order of first come first serve.

A variation of this algorithm is Random Two Choices (R2C) [10], that randomly chooses two VMs and assigns the task to the fastest one, i.e., the one with the lowest maximum response time. In On-Demand [7] method of scheduling ,each virtual machine monitors its task queue. When it detects that the a particular virtual machine has enough resources for a new task, it will send an On-Demand request to the broker that

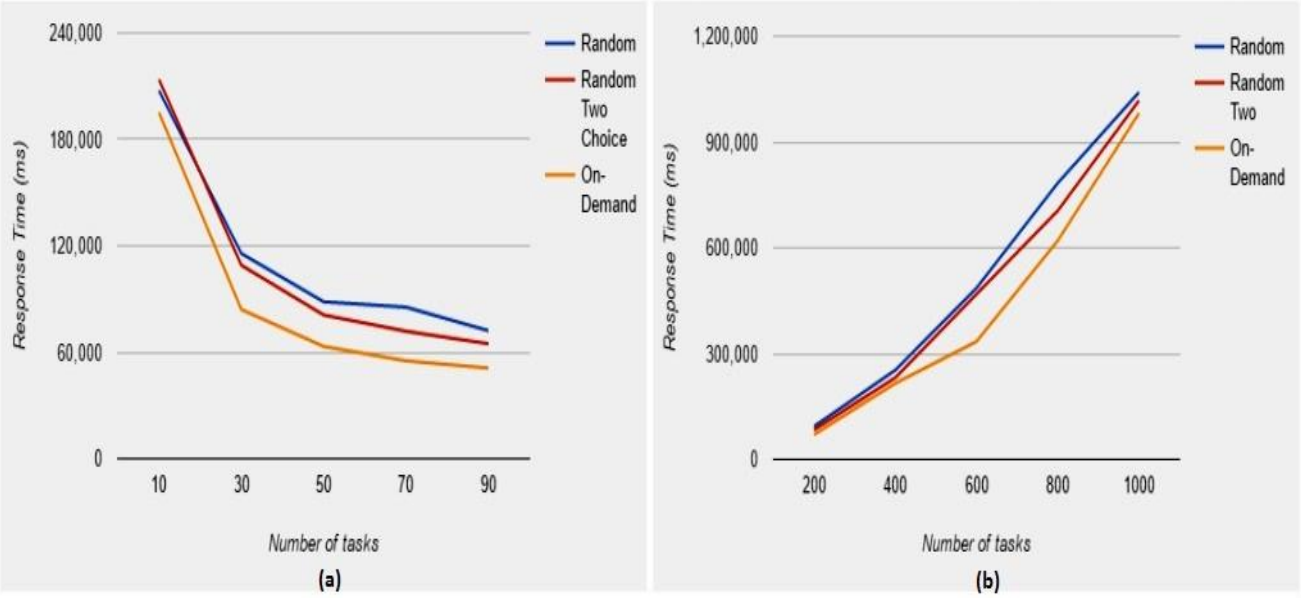

Figure.7 Comparison of Random, Random Two Choices (R2C) and On-Demand Scheduling methods

keeps a lightweighted metadata of the tasks. Then a new task will be scheduled to that virtual machine.

In figure.7(a) the comparison of response time for the three algorithms were done varying the number of VMs keeping the number of tasks as constant (For this simulation number of tasks is kept as 1000). The X-axis represents the number of virtual machines and Y-axis represents the response time in milliseconds. It is clearly evident that the response time is greatly reduced for On-Demand scheduling algorithm than the other two algorithms.

Figure.7(b) shows the comparison of response time for the three algorithms varying the number of tasks and keeping the number of VMs as constant (For this simulation number of virtual machines is kept as 100). The X-axis represents then number of tasks and Y-axis represents the response time in milliseconds. It is clearly evident that the response time is greatly reduced for On-Demand algorithm.

Another evaluation was performed comparing On-Demand and Credit-based method before and after applying migration.

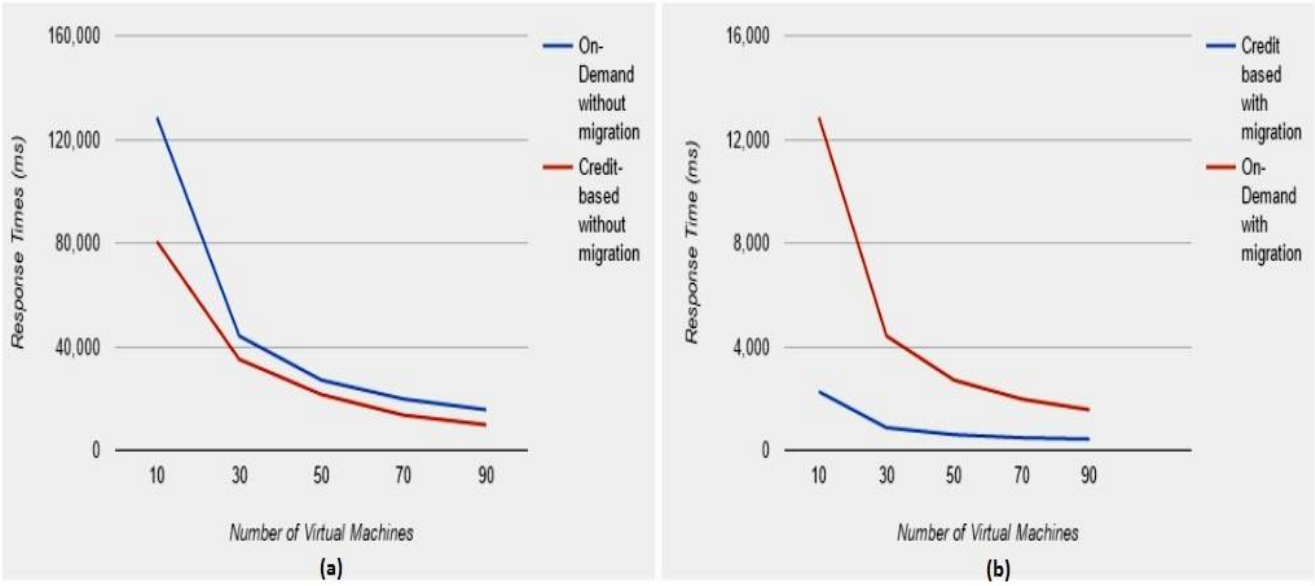

Figure.8 Comparison On-Demand and Credit-based Scheduling methods before and after applying migration

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

In figure.8 the comparison of response time for the two algorithms were done varying the number of VMs keeping the number of tasks as constant (For this simulation number of tasks is kept as 1000). The X-axis represents the number of virtual machines and Y-axis represents the response time in milliseconds. The length of tasks were generated randomly and the same set were given as input for length for the tasks. The credits in Credit-based scheduling were also generated randomly. From this Figure 8 it is evident that the response time is greatly reduced for Credit-based scheduling algorithm than On-Demand algorithm.

From this it is clear that using Credit-based scheduling achieves better load balancing compared to using On-Demand scheduling algorithm. Staged Task Migration for dependent tasks was compared against normal First Come First Serve (FCFS) method for dependent tasks. Same data dependent task model was considered for both these algorithms and the response time results shows that it takes only 20280ms for Staged Task Migration algorithm than FCFS algorithm which takes 67370ms. Hence the Staged Task Migration method can be said to be efficient.

IV. CONCLUSION

Load balancing is the process of distributing workloads across multiple computing resources. In this work, a scheme of load balancing is proposed on the basis of dependency among the tasks. Credit-based scheduling and Migrating Task methods were used for dealing with independent tasks. The Credit-based method used for scheduling the independent tasks was designed to consider both user priority and task length. Task Migration algorithm was used to guarantee balancing of loads among the virtual machines. The experimental results obtained demonstrate that the proposed techniques works efficiently decreasing the response time.

For dependent tasks, the dependencies between tasks are considered and the technique termed as data shuffling is used. The Staged Task Migration (STM) method used in this scheme is that the tasks in one stage run independently, while the tasks in different stages must be executed serially. This method was compared against FCFS and the experimental results demonstrate that the STM method significantly reduces the response time. In future, this algorithms can be enhanced to balance the loads of the tasks considering various QoS factors in a pre-emptive manner.

REFERENCES

1. Buyya, R., Ranjan, R., and Calheiros, R.N. " Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities" , International Conference on High Performance Computing and Simulation, HPCS 2009.
2. N. Susila, S. Chandramathi, Rohit Kishore, "A Fuzzy-based Firefly Algorithm for Dynamic Load Balancing in Cloud Computing Environment", Journal of Emerging Technologies in Web Intelligence, vol. 6, no. 4,pp.435-440, IEEE November 2014.
3. DineshBabu.L.D,P.VenkataKrishna,"HoneyBeeinspiredloadbalancing oftasks in cloud computing environment", Applied Soft Computing, vol.13,pp.2292-2303 ,Elsevier 2013.
4. Elina Pacini,Cristian Mateos,Carlos Garcia Garino, "Balancing throughput and response time in online scientific clouds via Ant Colony Optimization", Advances in Engineering Software, vol.8,pp.31-47 ,Elsevier 2015.
5. Brototi Mondala, Kousik Dasgupta, Paramartha Dutta," Load Balancing in Cloud Computing using Stochastic Hill Climbing-A Soft Computing Approach", Procedia Technology, vol.4, pp.783-789, Elsevier 2012.
6. Kousik Dasgupta, Brototi Mandal, Paramartha Dutta, Jyotsna Kumar Mondal, Santanu Dam," A Genetic Algorithm (GA) based Load Balancing Strategy for Cloud Computing", International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA), Elsevier, 2013.
7. B. R. Kandukuri, R. Paturi V, A. Rakshit, "Cloud Security Issues", IEEE International Conference on Services Computing, pp. 517-520, IEEE 2009.
8. Yu Liu, Changjie Zhang, Bo Li, Jianwei Niu ." DeMS: A hybrid scheme of task scheduling and load balancing in computing clusters", Journal of Network and Computer Applications, Elsevier 2015.
9. GaochaoXu, Junjie Pang, and Xiaodong Fu, "A Load Balancing Model Basedon Cloud Partitioning for the Public Cloud", vol.18 ,pp. 34-39,IEEE 2013.
10. Aarti Singha, Dimple Junejab, Manisha Malhotraa ,"Autonomous Agent Based Load Balancing Algorithm in Cloud Computing ", International Conference on Advanced Computing Technologies and Applications (ICACTA2015), vol.45, pp.832-841 , Elsevier 2015.
11. Michael Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing", IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 10, pp.1094-1104 ,IEEE 2001.
12. Antony Thomas, Krishnalal G, Jagathy Raj V P ,"Credit Based Scheduling Algorithm in Cloud Computing Environment", Procedia Computer Science, vol.46, pp. 913 920, Elsevier 2015.
13. Venubabu Kunamneni.," Dynamic Load Balancing for the Cloud", International Journal of Computer Science and Electrical Engineering (IJCSEE), ISSN No. 2315-4209, vol-1 issue-1, 2012
14. L. Wang, GregorLaszewski, Marcel Kunze, Jie Tao, "Cloud Computing: A Perspective Study", New Generation Computing-Advances of Distributed Information Processing, pp. 137-146, vol. 28, no. 2, 2008.
15. Ousterhout K, Wendell P, Zaharia M, Stoica I, "Batch sampling: low overhead schedulingforsub-secondparalleljob", Berkeley: University of California; 2012.
16. Weiwei Chen, Ewa Deelman, "Work flow Sim: A Toolkit for Simulating Scientific Work flows in Distributed Environments", The 8th IEEE International Conference on E Science (E Science 2012), Chicago, 2012.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.