IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 6, June 2016

CMS: An Improved Scheme for Dependency-Based Load Balancing in Cloud Environment

Neethu .M.S[1], Jayalekshmi .S[2]

M.Tech Student, Department of Computer Science & Engineering, LBSITW, Trivandrum, India [1]

Associate Professor, Department of Computer Science & Engineering, LBSITW, Trivandrum, India [2]

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 get 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.

Keywords: Load Balancing, Virtual Machine, Scheduling, Dependency.

I. INTRODUCTION

Cloud computing [13] 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 don't require long term contracts and are easier to scale up and down as per the demand. Prefect 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. Load balancing is a computer network method for distributing workloads across multiple computing resources.

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 suitable virtual machine. 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 [12] 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 machine. 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 approaches 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 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, under loaded 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 under loaded nodes and at high system loads receiver initiated component is successful in finding overloaded nodes.

II. RELATED WORKS

A. Fuzzy based Firefly Algorithm [2]

This algorithm was proposed with a goal to improve performance through partitioning the cloud inorder to balance the loads across the available partitions as shown in Fig.1. The nodes are to be classified into various groups like lightly, normal and heavily loaded. Then the tasks set are entered as input and given to the load balancer.

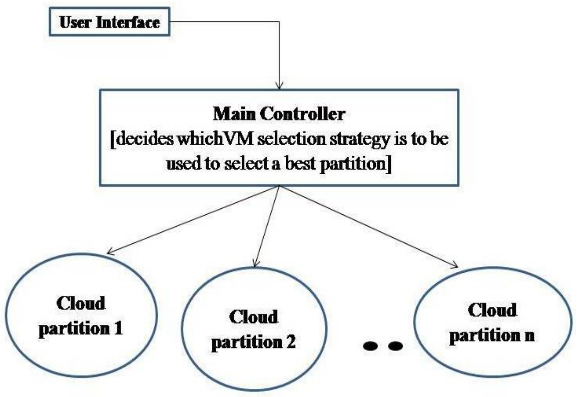

Fig. 1.Concept of partitioning

Fireflies "attract itself to light" and this is the main characteristic which is being used as the role of a VM for the implementation of the algorithm. A Balance Factor [2] is formulated which is given by:

$$BF=(\alpha*\beta)/(\delta*\varepsilon*\rho) \qquad (1)$$

where, α –Length of input file
β- Size of file in kilobytes
δ- Processing capacity of the VM supplied with the load
ε- Processing element number
ρ- Speed in number of CPU cycles
The value of balance factor always lies between 0 and 1. As a preliminary fuzzification process the algorithm assigns predefined values like high, medium, low to the tasks that arrives. The fuzzy engine defines the output using a defined rule and on defuzzification process using Smallest of Maximum (SOM) method, the output is obtained. A membership function [1] is calculated using Eq. (2):

$$Z\_SOM=MIN(\square\ Z \in Z1,Z2) \qquad (2)$$

where, Z-Output variable which is the minimum amongst the output variables Z1,Z2 etc.
Initially the VMs are allocated an ID from M_1 to M_n Cloudlets are created and each cloudlet is assigned with an ID from N_1 to N_p .Based on an available scheduling scheme the tasks are scheduled. Usually Round-robin scheme is used.

Now a best partition is selected and from that partition a VM is selected and its balance factor is calculated. When the value of BF is less than 1, allocate load to that VM and update the parameters. Otherwise when the BF value is greater than or equal to 1, the algorithm selects another VM. After allocation of load to a VM, if the time required is found to be greater than a predefined threshold value then fuzzy logic is applied.

B. Honey Bee Behavior Inspired Load balancing [3]

The algorithm [3] was proposed to achieve load balancing by improving through put and minimizing the waiting time of the tasks. The tasks removed from VM were treated as honeybees. The bee colony consists of three types of bees. They are Scout bees, Forager bees and Onlooker bees. Scout bees are those bees which carry out random searches and on finding a bee hive it informs the forager bees. Forager bees are those going to the food source which is visited by scout bees.
HBB-LB [2] is a dynamic load balancing technique. Capacity of a virtual machine is given by (3):

$$C_j=pe_{numj} \times pe_{mipsj} \times vm_{bwj} \qquad (3)$$

where , pe_{numj} – Number of processors in VM_j
pe_{mipsj} – Million instructions per second of all processors in VM_j
vm_{bwj} - Communication bandwidth ability of VM_j
Thus the capacity of all virtual machines is give by sum of capacities ^{Ci} of each virtual machine.
Load on a VM is the total length of the tasks assigned to it. From this the standard deviation of load is calculated. The load balancing decision is taken using two steps . The first one is finding the state of VM from the calculated value of standard deviation. When the calculated standard deviation of load is between 0 and 1, system can be said to be balanced. The second step deals with finding the overloaded VMs by checking whether the current workload of a VM exceeded the maximum capacity for that particular VM.

C. Ant Colony Based Load Balancing Algorithm [4]

The main aim of this algorithm [4] is to search for an optimal path between the source of food and colony of the ant on the basis of their behaviour. Ants keep record of each and every node that they visits and record that data for future decision making. As a result they deposit pheromones during their movement. On allocation of VMs, the concept is that each ant works independently and represents a VM "looking" for a host to get allocated. A master table is created which has the details of the loads of each host and is termed as pheromone table.

At first a list of all the hosts are created which can be allocated with the VMs. The ant's moves through the network continuously encountering overloaded and underloaded nodes. Along the traversal through the nodes the ants updates the pheromone table. When an ant encounters an overloaded node in its movement, such that it has previously encountered an underloaded node then it goes back to check the underloaded node to check whether the node is still under loaded or not. If it still finds that the node is underloaded then the load is distributed to that node and this process is known as exponential back-off strategy.

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 6, June 2016

Each time a job is assigned to a VM, a round-robin scheduling was used. The algorithm uses indirect communication to exchange information.

D. Stochastic-Hill Climbing Algorithm [5]
This algorithm helps in allocating jobs to servers or VMs. Stochastic-Hill Climbing algorithm [5] is a variant of Incomplete method. There are main two concepts, a candidate generator and an evaluation criteria. Candidate generator maps one solution candidate to a set of possible successors. Evaluation criteria ranks each valid solution.
The algorithm [5] can be summarized as follows and the flowchart can be depicted as in Fig.2. An index table is maintained for storing the state of VM servers. The states can be either VMBUSY/AVAILABLE. At the start all VMs are available. When a new task arrives a VM is randomly generated by its unique identifier. The allocation table is checked to know the status of the VM. If it is found unallocated, the task is allocated to it and the table is updated with the current information. Otherwise using a random function, generate another VM such that it is able to handle the task efficiently. When the VM finishes processing, VM de-allocation is done. The process gets repeated again on obtaining new tasks.

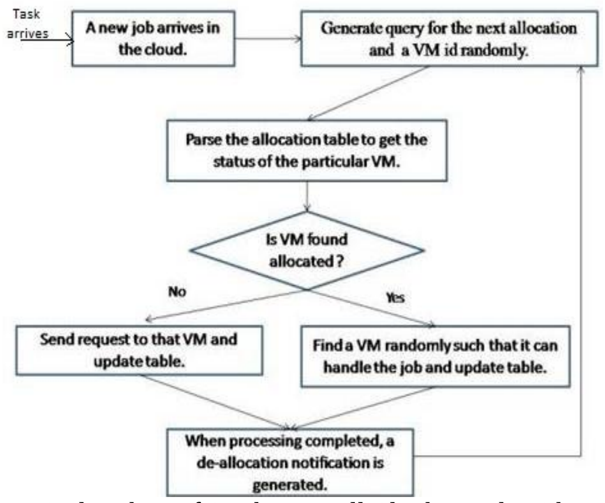
Task arrives
A new job arrives in the cloud.
Generate query for the next allocation and a VM id randomly.
Parse the allocation table to get the status of the particular VM.
Is VM found allocated ?
No
Yes
Send request to that VM and update table.
Find a VM randomly such that it can handle the job and update table.
When processing completed, a de-allocation notification is generated.

Fig.2 Flowchart of Stochastic Hill Climbing Algorithm

E. Load Balancing Model based on Cloud Partitioning [8]
The model [8] is based on partitioning the cloud. There is a main controller which deals with the various partitions. So when a job arrives it is the main controller who decides which partition to get it allotted.

The cloud partition status is determined before it gets assigned with a job. The various status are: idle, normal and overloaded. The function of cloud partition balancer is to collect information about the status of the nodes in that particular partition. For finding the status of a node , the load degree has to be calculated. This can be obtained from the static and dynamic parameters of a system. When the load degree is zero, the system is idle and when it exceeds the higher value of load degree then it is said to be overloaded. In other conditions, the node is said to be in normal state.

The load degree values are entered into a Load Status Table [8] by the load balancer of a particular partition. When a job arrives at a cloud partition, the table is updated by the balancers. The nodes with status value idle uses round robin scheduling method for the jobs that are ordered based on the load degree from the lowest to the highest. The system builds a circular queue and walks through the queue again and again. Jobs will then be assigned to nodes with low load degrees.

F. Autonomous Agent Based Load Balancing (A2LB) [9]
A main concept used in all algorithms is that whenever a node gets overloaded, the load balancer has to distribute these tasks such that the load gets balanced on another node. A2LB [9] uses three agents: Load agent, Channel Agent and Migration Agent.

The main aim of a Load Agent (LA) is to calculate the load on every available virtual machine after a new job is allocated in the data centre. This agent is uses a table termed as VM_Load_Fitness table which contains the records of specifications of all virtual machines of a data centre. Channel Agent (CA) controls the transfer policy, selection policy and location policy. Migration Agents (MA) are initiated by channel agent.

Load agent determines the workload and calculates the fitness value. When the fitness value is below a threshold, load balancing needs to be performed. When the load agent finds that the status of a VM is critical, it will intimate and send the specification of that VM to the channel agent. The channel agent will initiate the migration agents to other data centres for searching the virtual machines that satisfies the similar specifications. Migration agents will travel one way. When it finds a destination data centre, migration agent will first send an acknowledgement message to its parent channel agent. Then it will check with load agent of that data centre for finding the virtual machines having similar configuration as desired. If no such VM exists at that data centre, migration agent sends a <Not-Applicable> message back to its parent channel agent and waits for <self_destroy> instruction from it. When receiving responses from various migration agents, channel agent maintains them in response analysis table and thus live migration is achieved.

III.PROPOSED SYSTEM

The parallel job 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 pay attention to the dependencies of the tasks; tasks dependent on other tasks or system resources have to suspend until the preconditions are satisfied. In this paper, a hybrid scheduling scheme is proposed. We use a Credit-based scheduling method to reduce communication overhead between the virtual machines. A task migration algorithm is designed to keep the workload balanced. A data shuffling mechanism is employed for dependent tasks.

A. Credit-based Scheduling of Independent Tasks

This method 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 task to be scheduled on the available virtual machines will be of different length. When the tasks are arranged on the increasing order of the length, tasks with shorter length will be present in the beginning of the list and that of highest length will reside at the last.

The credit calculation for task length will work such that it takes tasks from both front and back as follows:

- Step 1: Find the length of each task as TLi.
- Step 2: Calculate the average length of tasks as avg_{len}.
- Step 3:Calculate the difference in length with respect to avg_{len} as $TLD_i = | avg_{len} - TL_i |$, (4)

where, TLD_i is the task length difference of task i.

- Step 4: Four values v1,v2,v3 and v4 are calculated in the range of task length from the array length of the tasks:

$$v1=high\_len/5 \quad (5)$$
$$v2=high\_len/4 \quad (6)$$
$$v3= v2+ v1 \quad (7)$$
$$v4= v3+ v2 \quad (8)$$

where, high_len is the highest value of task length.

- Step 5: For all submitted tasks in the set ;Ti

```
        $TLD_i = | avg_{len} - TL_i |$
         If $TLD_i \leq v1$
              then credit =5
          else if $v1 < TLD_i \leq v2$
              then credit =4
          else if $v2 < TLD_i \leq v3$
              then credit =3
          else if $v3 < TLD_i \leq v4$
              then credit =2
          else $v4 > TLD_i$
              then credit =1
```

- Return credit as length_credit.

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:

When the tasks are scheduled, there is a problem of treating them with similar priority. Each task may be assigned different priority, which can be represented as a value assigned to each task and this value can be the same for more than one task. 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 [11] 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:

```
Step 1: For each task $T_i$ find the task with the highest priority.
Step 2: Find a division_factor
Step 3: For each task with priority $T_{pri}$
            find $Pri\_frac(i)=T_{pri}$ /division_factor        (9)
          Set priority_credit as Pri_frac
Step 4:End For.
```

The div_fac is chosen such that if highest value of priority is a two digit number then choose div_fac is 100. If it is 3 digit then division_part is 1000. The two credits calculated are used to find the total credit as

$$Total\_credit_i=length\_credit_i \times priority\_credit_i \quad (10)$$

For each task i, the $Total\_credit_i$ represents the credit based on both length and priority. 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 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. This can be summarized as follows:

Input: Set of VMs $\{VM_1,VM_2,\ldots.VM_n\}$ and the batch of tasks after applying credit calculation $\{T_1,T_2,\ldots T_m\}$

Output: Migrating task on buzy slave to low loaded VM.

- Step 1: For each VM_i in $\{VM_1,VM_2,\ldots.VM_n\}$
- Step 2: Calculate the total length of tasks scheduled to it as VM_i_length
- Step 3: End for
- Step 4: Calculate the execution time for the last scheduled task for each VM_i as T_j_execstart and execution finish time as T_j_execfinish.
- Step 5: If there exists any VM_k whose T_j_execfinish< T_j_execstart then move the task T_j to the VM_k for execution

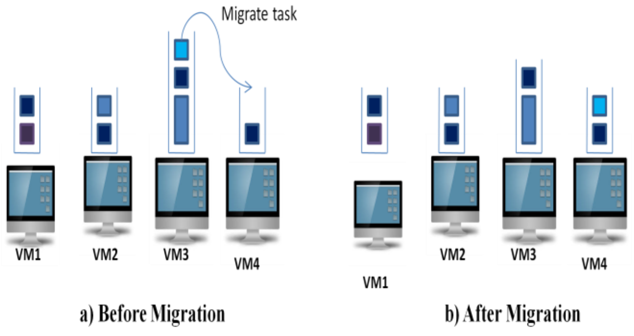

Fig.3 migrating tasks inorder to balance the loads across the available VMs

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 6, June 2016

C. Staged Task Migration for Dependent Tasks

Suppose there are n numbers of dependent tasks (T_1, T_2, T_3… T_n). The scheduling problem with dependent task [14] is that a child task cannot start its execution until all its parent tasks have finished their execution. The tasks in one stage run independently, while the tasks in different stages must be executed serially. To dispatch tasks with Data Shuffling [7], a queue named Shuffling FIFO is used which holds the tasks in the order to be executed. The steps can be summarized as follows:

For a new incoming task T_i in the current stage, choose a VM with low workload and dispatches the task to it. Then the queues in virtual machines will not be changed and hence the optimal scheduling is achieved. At this time, the tasks of the later stage are popped out from the FIFO and dispatched to virtual machines without waiting for the completion of the tasks in the current stage.

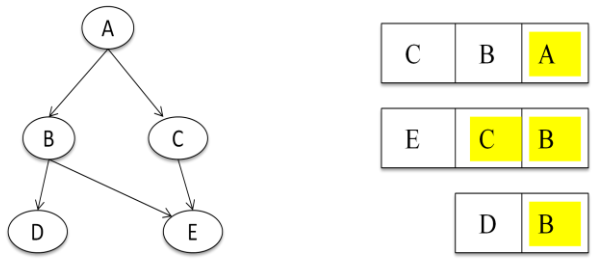

Fig.4 Ordering the dependent tasks for scheduling.

Figure.4 shows the way the tasks are arranged in the order it is to be scheduled. Thus the final ordering for scheduling is that as soon as task A completes its execution, B and C can be scheduled for their execution. Whenever B completes execution, task D can be scheduled for execution but task E can be scheduled only if tasks B and C completes their execution.

IV. RESULTS AND DISCUSSIONS

In real cloud, experimenting new strategies and methods is very difficult due to security, reliability, cost and speed. Hence a good simulator is required for this purpose. CloudSim toolkit [1] is one such simulator designed for the same. CloudSim is a discrete simulation framework that allows modeling, simulation and experimenting various the cloud computing services.

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 proposed approach was implemented using CloudSim toolkit. 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.

A. Scheduling and Migration of Independent Tasks

As part of evaluating the system, comparisons were carried out on Random, Random Two Choices (R2C) and On-Demand schedulings[7] 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 lowload, 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 Two Random Choices (2RC) [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 keeps a lightweighted metadata of the tasks. Then a new task will be scheduled to that virtual machine.

In figure.5 (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 simu lation 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.5 (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 the 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. In figure.6 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 was 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 6 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.

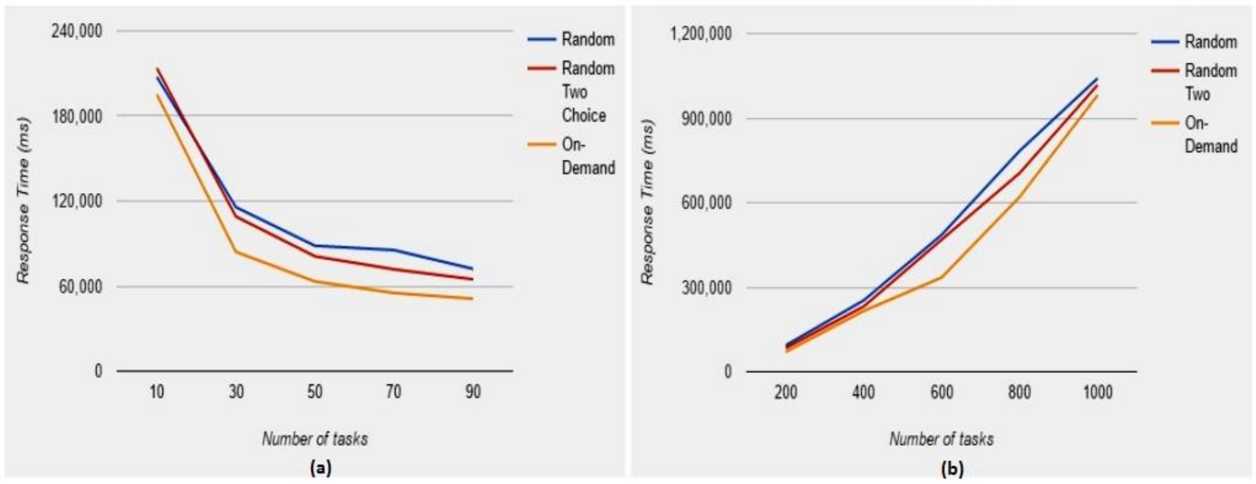

Fig.5 Comparison of Random, RandomTwoChoices(R2C) and On-Demand Scheduling methods

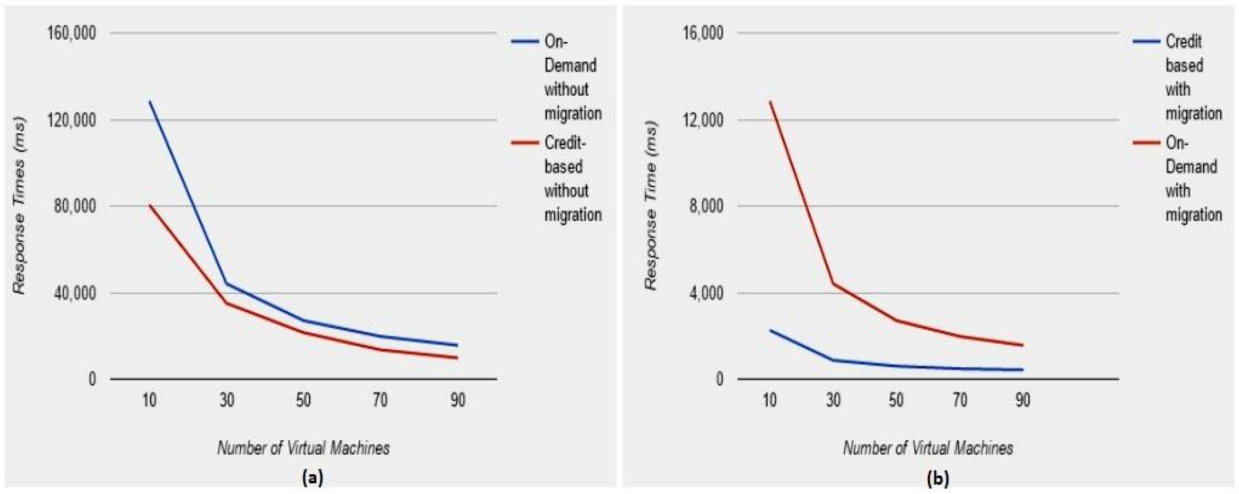

Fig.6 Comparison On-Demand and Credit-based Scheduling methods

B. Staged Task Migration for Dependant Tasks

The algorithm for Data Shuffling was implemented in WorkflowSim [15] which is an extension to CloudSim simulator. WorkflowSim has simple models of task execution that consider task dependencies which is not supported by CloudSim alone. The response time for a set of dependent tasks were calculated. It was found to be less than the response time, if the dependent tasks were scheduled in the order of First Come First Serve.

Staged Task Migration for dependent tasks was compared against normal FirstCome 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.

V. CONCLUSION

The paper considers two cloud scenarios. First scenario is based on independent tasks. The second scenario is based on dependent tasks. On evaluating the simulation results it is concluded that, the proposed algorithm for independent tasks works efficiently than the other two methods namely Random and Random Two choices. It is also observed that the response time of task is decreasing after a certain value in the number of tasks. In future, the proposed scheme can be enhanced so as to consider other parameter like deadline and QoS factors.

The algorithm for dependant tasks works well for the given tasks. In future, extend this algorithm to balance the loads of dependent 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] Dinesh Babu .L.D, P.Venkata Krishna,"Honey Bee inspired load balancing of tasks 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 ,vo.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] B. R. Kandukuri, R. Paturi V, A. Rakshit, "Cloud Security Issues", IEEE International Conference on Services Computing, pp. 517-520, IEEE 2009.
[7] 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.
[8] Gaochao Xu, Junjie Pang, and Xiaodong Fu , "A Load Balancing Model Based on Cloud Partitioning for the Public Cloud" , vol.18 ,pp. 34-39,IEEE 2013.
[9] 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.
[10] 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.
[11] 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.
[12] Venubabu Kunamneni" Dynamic Load Balancing for the Cloud" International Journal of Computer Science and Electrical Engineering (IJCSEE) ISSN No. 2315-4209, Vol-1 Iss-1, 2012
[13] 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.
[14] Ousterhout K, Wendell P, Zaharia M, Stoica I, " Batch sampling: low overhead scheduling for sub-second parallel job", Berkeley: University of California; 2012.
[15] Weiwei Chen, Ewa Deelman , "WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments", The 8th IEEE International Conference on eScience 2012 (eScience 2012), Chicago, 2012.