Applied Soft Computing 13 (2013) 2292–2303

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Honey bee behavior inspired load balancing of tasks in cloud computing environments

Dhinesh Babu L.D.[a,*], P. Venkata Krishna[b]

[a] School of Information Technology and Engineering, VIT University, Vellore, India
[b] School of Computing Science and Engineering, VIT University, Vellore, India

ARTICLE INFO

Article history:
Received 30 August 2012
Received in revised form
28 December 2012
Accepted 30 January 2013
Available online 14 February 2013

Keywords:
Load balancing
Cloud computing
Priorities of tasks
Honey bee behavior
Performance evaluation

ABSTRACT

Scheduling of tasks in cloud computing is an *NP*-hard optimization problem. Load balancing of non-preemptive independent tasks on virtual machines (VMs) is an important aspect of task scheduling in clouds. Whenever certain VMs are overloaded and remaining VMs are under loaded with tasks for processing, the load has to be balanced to achieve optimal machine utilization. In this paper, we propose an algorithm named honey bee behavior inspired load balancing (HBB-LB), which aims to achieve well balanced load across virtual machines for maximizing the throughput. The proposed algorithm also balances the priorities of tasks on the machines in such a way that the amount of waiting time of the tasks in the queue is minimal. We have compared the proposed algorithm with existing load balancing and scheduling algorithms. The experimental results show that the algorithm is effective when compared with existing algorithms. Our approach illustrates that there is a significant improvement in average execution time and reduction in waiting time of tasks on queue.

1. Introduction

Cloud computing is an entirely internet-based approach where all the applications and files are hosted on a cloud which consists of thousands of computers interlinked together in a complex manner. Cloud computing incorporates concepts of parallel and distributed computing to provide shared resources; hardware, software and information to computers or other devices on demand. These are emerging distributed systems which follows a "pay as you use" model. The customer need not buy the software or computation platforms. With internet facility, the customer can use the computation power or software resources by paying money only for the duration he/she has used the resource. This forces the conventional software licensing policies to change and avoids spending of money for the facilities the customer does not use in a software package.

The customer is interested in reducing the overall execution time of tasks on the machines. The processing units in cloud environments are called as virtual machines (VMs). In business point of view, the virtual machines should execute the tasks as early as possible and these VMs run in parallel. This leads to problems in scheduling of the customer tasks within the available resources. The scheduler should do the scheduling process efficiently in order to utilize the available resources fully. More than one task is assigned to one or more VMs that run the tasks simultaneously. This kind of environments should make sure that the loads are well balanced in all VMs i.e., it should make sure that the tasks are not loaded heavily on one VM and some VMs do not remains idle and/or under loaded. In this case, it is the responsibility for the scheduler to balance the loads across the machines. A load balancing algorithm attempts to improve the response time of user's submitted applications by ensuring maximal utilization of available resources.

The main objective of load balancing methods is to speed up the execution of applications on resources whose workload varies at run time in unpredictable way [10]. Load balancing techniques are widely discussed in homogeneous as well as heterogeneous environments such as grids. There are basically two kinds of load balancing techniques. They are (i) Static and (ii) dynamic.

Static algorithms work properly only when nodes have a low variation in the load. Therefore, these algorithms are not suitable for cloud environments where load will be varying at varying times. Dynamic load balancing algorithms are advantageous over static algorithms. But to gain this advantage, we need to consider the additional cost associated with collection and maintenance of the load information.

Dynamic techniques are highly successful for load balancing of tasks among heterogeneous resources. Our proposed load balancing technique is also a dynamic technique which will not only balance the load but also take into account the priorities of tasks in the waiting queues of VMs.

In cloud computing environments, whenever a VM is heavily loaded with multiple tasks, these tasks have to be removed and

* Corresponding author. Tel.: +91 0416 2243091.
E-mail addresses: lddhineshbabu@vit.ac.in (Dhinesh Babu L.D.), pvenkatakrishna@vit.ac.in (P. Venkata Krishna).

1568-4946/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2013.01.025

submitted to the under loaded VMs of the same data center. In this case, when we remove more than one task from a heavy loaded VM and if there is more than one VM available to process these tasks, the tasks have to be submitted to the VM such that there will be a good mix of priorities i.e., no task should wait for a long time in order to get processed. Load balancing is done at virtual machine level i.e., at intra-data center level.

Our approach suggests that load balancing in cloud computing can be achieved by modeling the foraging behavior of honey bees. This algorithm is derived from a detailed analysis of the behavior that honey bees adopt to find and reap food. In bee hives, there is a class of bees called the scout bees which forage for food sources, upon finding one, they come back to the beehive to advertise this using a dance called waggle/tremble/vibration dance. The display of this dance, gives the idea of the quality and/or quantity of food and also its distance from the beehive. Forager bees then follow the Scout Bees to the location of food and then begin to reap it. They then return to the beehive and do a waggle or tremble or vibration dance to other bees in the hive giving an idea of how much food is left and hence resulting in either more exploitation or abandonment of the food source.

In the same manner, the removed tasks from over loaded VMs are considered as the honey bees. Upon submission to the under loaded VM, the task will update the number of various priority tasks and load of that particular VM to all other waiting tasks. This will be helpful for other tasks in choosing their virtual machine based on load and priorities. Whenever a high priority task has to be submitted to other VMs, it should consider the VM that has minimum number of high priority tasks so that the particular task will be executed at the earliest. Since all VMs will be sorted in ascending order based on load, the task removed will be submitted to under loaded VM. In essence, the tasks are the honey bees and the VMs are the food sources. Loading of a task to a VM is similar to a honey bee foraging a food source (a flower or a patch of flowers). When a VM is overloaded i.e., similar to the honey getting depleted at a food source, the task will be scheduled to an under loaded VM similar to a foraging bee finding a new food source. This removed task updates the remaining tasks about the VM status similar to the waggle/tremble/vibration dance performed by the honey bees to inform other honey bees in the bee hive. This task will update the status of the VM i.e., how many tasks are being processed by the VM and about the number and details of high priority tasks currently processed by the VM in a manner similar to the bees finding an abundant food source updating the other bees in the bee hive through its waggle dance. This updating will give a clear idea in deciding which task should be assigned to which VM based on the availability and load of the VMs similar to which honey bees should visit which food source based on whether honey is available at a flower patch or not. The proposed algorithm works well for load balancing of tasks in cloud computing environments.

The specific contributions of this paper include

- An algorithm for scheduling and load balancing of non-preemptive independent tasks in cloud computing environments inspired by honey bee behavior
- A literature survey about various existing load balancing algorithms and the merits/demerits of these techniques
- Correlation of the proposed HBB-LB algorithm with actual foraging behavior of honey bees using a clear flow diagram showing the behavioral control structures of honey bees and HBB-LB.
- An analysis and systematic study with mathematical evidence to show how the honey bee behavior inspired load balancing can work for cloud computing environments
- Performance analysis of the proposed algorithm and an evaluation of the algorithm with respect to other existing algorithms.

Rest of this paper is organized as follows: Section 2 discusses about the related works on existing load balancing techniques. Section 3 describes the foraging behavior of honey bees and how it relates to the proposed technique. Section 4 focuses on our proposed approach with detailed algorithm and Section 5 presents the experimental results along with performance evaluation of the algorithm in comparison with existing algorithms. Finally we conclude this paper highlighting the contributions and future enhancements in Section 6.

2. Related works

Load balancing is removing tasks from over loaded VMs and assigning them to under loaded VMs. Load balancing can affect the overall performance of a system executing an application. Load balancing algorithms can be classified in two different ways [2]:

Static load balancing algorithms: The decisions related to balancing of load will be made at compile time when resource requirements are estimated. The advantage of this algorithm is the simplicity with respect to both implementation and overhead, since there is no need to constantly monitor the nodes for performance statistics. Static algorithms work properly only when there is a low variation in the load for the VMs. Therefore, these algorithms are not well suited for grid and cloud computing environments where the load will be varying at various points of time.
Dynamic load balancing algorithms: Dynamic load balancing algorithms make changes to the distribution of work load among nodes at run-time; they use current load information when making distribution decisions [16].

Houle et al. [9] consider algorithms for static load balancing on trees treating that the total load is a fixed one. In [6], Hu et al. propose an optimal data migration algorithm in diffusive dynamic load balancing through the calculation of Lagrange multiplier of the Euclidean form of transferred weight. This work can effectively minimize the data movement in homogenous environments, but it does not consider heterogeneous environments. Genaud et al. [12] enhanced the MPI_Scatterv primitive to support master-slave load balancing by taking into consideration the optimization of computation and data distribution using a linear programming algorithm. However, this solution is limited to static load balancing.

In [7], a New Time Optimizing Probabilistic Load Balancing Algorithm in Grid Computing is presented. This algorithm chooses the resources based on better past status and least completion time. The main purpose of this algorithm is to establish load balancing and reduce the response time. In [1], a Task Load Balancing Strategy for Grid Computing is presented. In this paper, a hierarchical load balancing strategy and associated algorithms based on neighborhood property is discussed. This strategy privileges local balancing in first (load balance within sites without communication between sites). Then, upper hierarchical balancing will take place and so on. The main benefit of this idea is the decrease in the amount of messages exchanged between Grid resources. This system creates a hierarchical architecture that is totally independent of Grid architecture. In [5], titled "Dynamic Load Balancing in Grid Computing", like the previous paper, this paper presents a task load balancing model in Grid environment. It also details the system in a manner similar to the previous reference. The main characteristics of this strategy are: (i) It uses task-level load balancing; (ii) It privileges local tasks transfer to reduce communication costs; (iii) It is a distributed strategy with local decision making. This system transforms the Grid to tree structure independent of Grid topological structure complexity. The system will use this tree for load balancing.

In [8], A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing is presented. This paper considers three potentially viable methods for load balancing in large scale cloud systems. Firstly, a nature-inspired algorithm may be used for self-organization, achieving global load balancing via local server actions. Secondly, self-organization can be engineered based on random sampling of the system domain, giving a balanced load across all system nodes. Thirdly, the system can be restructured to optimize job assignment at the servers. Recently numerous nature-inspired networking and computing models have received a lot of research attention in seeking distributed methods to address increasing scale and complexity in such systems.

The honey-bee foraging solution in [3], is investigated as a direct implementation of a natural phenomenon. Then, a distributed, biased random sampling method that maintains individual node loading near a global mean measure is examined. Finally, an algorithm for connecting simile services by local rewiring is assessed as a means of improving load balancing by active system restructuring. In case of load balancing, as the web servers demand increases or decreases, the services are assigned dynamically to regulate the changing demands of the user. The servers are grouped under virtual servers (VS), each VS having its own virtual service queues. Each server processing a request from its queue calculates a profit or reward, which is analogous to the quality that the bees show in their waggle dance.

In [4], Dynamic Load Balancing Strategy for Grid Computing is presented addressing the problem of load balancing in Grid computing. As in [1,5] this paper also proposes a load balancing model based on a tree representation of a Grid. This load balancing strategy has two main objectives: (i) Reduction of the mean response time of tasks submitted to a Grid; and, (ii) Reduction of the communication costs during task transferring. This strategy deals with three layers of algorithms (intra-site, intra-cluster and intra-grid).

Load balancing algorithms can be defined based on the implementation of the following policies [11]:

- *Information policy*: specifies what workload information to be collected, when it is to be collected and from where.
- *Triggering policy*: determines the appropriate period to start a load balancing operation.
- *Resource type policy*: classifies a resource as *server* or *receiver* of tasks according to its availability status.
- *Location policy*: uses the results of the resource type policy to find a suitable partner for a server or receiver.
- *Selection policy*: defines the tasks that should be migrated from overloaded resources (source) to most idle resources (receiver).

Load Balancing can also be classified into more categories based on different behaviors of load balancing algorithms [5]: Centralized vs. distributed load balancing and application-level vs. system-level load balancing

In [13], A Routing Load Balancing Policy for Grid Computing Environments is presented. It uses routing concepts from computer networks to define a neighborhood and search the most adequate computers to divide applications' workload. This algorithm is designed to equally distribute the workload of tasks of parallel applications over Grid computing environments. Route algorithm is indicated for environments where there are several heterogeneous computers and parallel applications are composed of multiple tasks. When dealing with large scale systems, an absolute minimization of the total execution time is not the only objective of a load balancing strategy. The communication cost, induced by load redistribution, is also a critical issue. For this purpose, Yagoubi proposes in [14], a hierarchical load balancing model as a new framework to balance computing load in a Grid. This model suffers from bottlenecks. In [23], a Mathematical model of cloud computing framework using fuzzy bee colony optimization technique is presented. Honey Bee colony algorithm is used for web services in web servers that are scattered.

3. Honey bee foraging behavior

The artificial bee colony algorithm (ABC), an optimization algorithm based on the intelligent foraging behavior of honey bee swarm was proposed by Karaboga in 2005 [26,38]. This new Meta heuristic is inspired by the intelligent foraging behavior of honey bee swarm. The algorithm presented in the work is for numerical function optimization. The advantage of ABC is that the global search ability in the algorithm is implemented by introducing neighborhood source production mechanism [27]. Rao et al. [27] deals with radial distribution system network reconfiguration problem. This paper presents a method for determining the sectionalizing switch to be operated in order to solve the distribution system loss minimization problem. ABC algorithm were used in many fields such as digital signal processing [28], leaf-constrained minimum spanning tree problem [29], flow shop scheduling problem [30], block matching algorithm for motion estimation [39], optimization [42] and inverse analysis problems [31]. Tsai et al. present an interactive artificial bee colony supported passive continuous authentication system [41].

In [32], a new population-based search algorithm called the *Bees Algorithm* (*BA*) is presented. The algorithm mimics the food foraging behavior of swarms of honey bees. In its basic version, the algorithm performs a kind of neighborhood search combined with random search and can be used for both combinatorial optimization and functional optimization. Honey bees have developed the ability to collectively choose between nectar sources by selecting the optimal one: This source provides a maximum ratio of gain compared to costs [34]. The whole decentralized decision process is based on competition among dancing bees, which guide new (naive) bees to their foraging targets. In [37], authors have proposed Load balancing using bees algorithm.

Honey bee behavior is also used in web services. In [19], "On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers", authors propose a new honey bee allocation algorithm based on self-organized behavior of foragers in honey bee colonies. Hosting centers then must allocate servers among clients to maximize revenue. The allocation of servers to collect revenue in Internet hosting centers parallels the allocation of foragers to collect nectar in honey bee colonies. A hosting center with a certain number of servers hosting multiple Internet clients is analogous to a honey bee colony with a certain number of bees foraging at multiple sites in the surrounding countryside.

A model of self-organization that takes place within a colony of honey bees has been presented in [33]. This Insect foraging technique is used in the field of robotics. The main principles of social insect foraging behavior can find an application in a swarm of inexpensive insect-like robots [35,36].

According to Johnson and Nieh, Honey Bees are social insects where collective decisions are made via feedback cycles based on positive and negative signaling [24]. Fig. 1 shows a simplified flow diagram of behavioral control structure for foraging honey bees as presented by Brian R. Johnson & James C. Nieh. This behavioral model is a powerful and tested model describing the foraging behavior of honey bees. This model is based on the behavioral structure of forager bee developed by Han de Vries & Jacobus C. Biesmeijer [25]. An important means for communication among honey bees is through waggle dance, a dance which will give an idea to the waiting bees in the nest about a potential food source, its distance from the bee hive etc. In addition, honey bees use tremble and vibration dances also.

Our proposed Honey Bee Behavior inspired Load Balancing is based on the above Brian R. Johnson and James C. Nieh behavioral

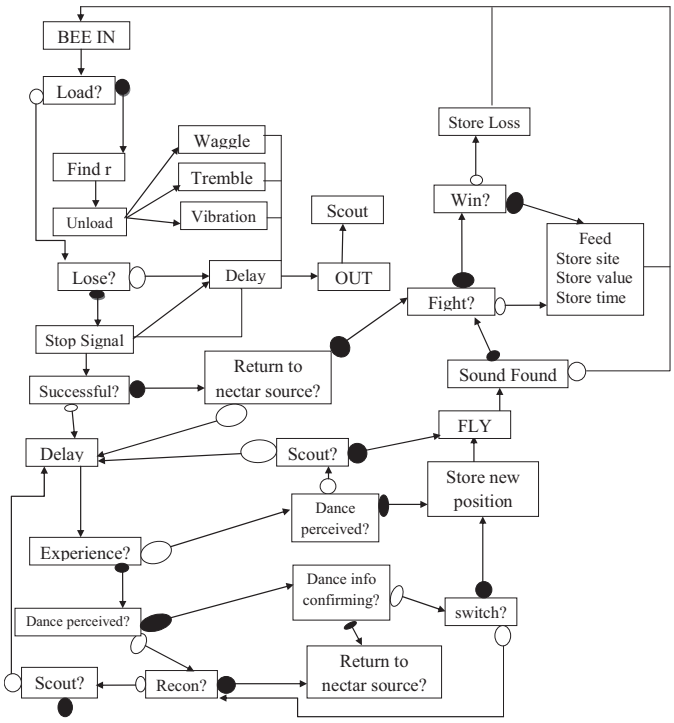

Fig. 1. Flow diagram of the behavioral control structure for foraging honey bees adapted from Johnson and Nieh [25] with minor changes.

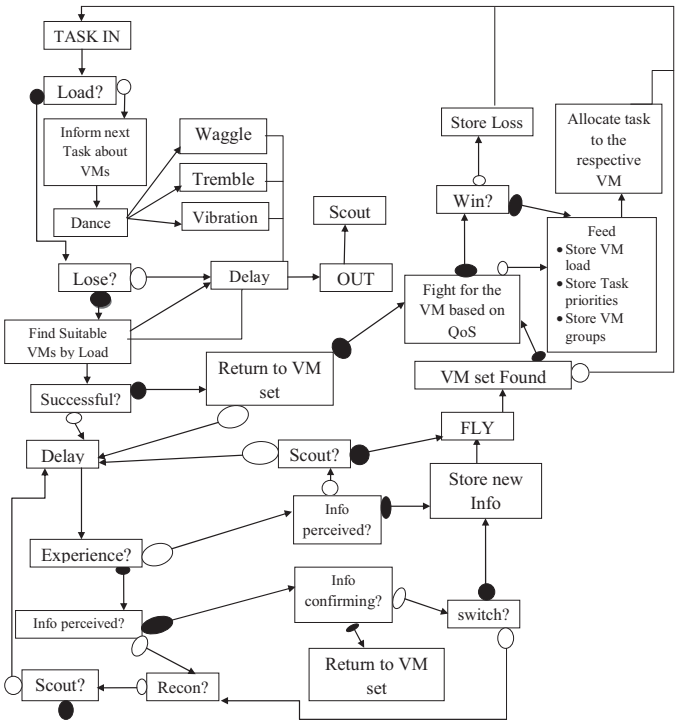

Fig. 2. Flow diagram of the behavioral control structure for Honey Bee Behavior inspired Load Balancing of Tasks in Cloud Computing inspired by foraging behavior of real honey bees adapted from Johnson and Nieh [25] with minor changes.

model of honey bees. In Fig. 1, arrows with a black filled start circle represent positive signals and arrows with hollow circle represent negative signals. Both situations, positive and negative require some sort of communication among honey bee swarms. We have mapped the above flow diagram showing the behavioral control structure for foraging honey bees into the proposed Honey Bee Behavior inspired Load Balancing algorithm. The proposed algorithm is completely inspired by the natural foraging behavior of honey bees and is illustrated in Fig. 2.

A task removed from overloaded VM has to find suitable under loaded VMs it can be allocated to. It has two possibilities i.e., either it finds the VM set (Positive signal) or it may not find the suitable VM (negative signal). There could be more than one VM (a set of VMs for allocation) which can accept this task. Now the task has to find best among these VMs based on the QoS criteria called task priority i.e., task finds the VM which has a less number of tasks with same kind of priority (high priority tasks finds the VM which has less number of high priority tasks). We call it as fight for the VM among tasks. When this fight is over, the winning task is allocated to the respective VM found and the details are updated.

If a task does not find a suitable VM, it goes for delay in allocation, during which it gets experience and it will start listening the information updates by other tasks (similar to honey bees listening various dances). Once it confirms the information, the process starts from finding the VM sets and after successful VM set identification, it will fight with other competing tasks to find it's most suitable VM and gets allocated to it. Information is updated once it ends its fight with other tasks (whether it loses or wins). As newer tasks arrive, the cycle starts until all tasks are allocated to VMs and the scheduling system is well balanced based on load as well as priorities.

4. Honey bee behavior inspired load balancing (HBB-LB) algorithm

Cloud computing deals with assigning computational tasks on a dynamic resource pool of virtual machines online according to different requirements from user or the system [15]. The service requests from the clients for diverse applications can be routed at any data center to any end server in the cloud. The routing of service requests to the diverse servers is based on cloud management policies depending on load of individual servers, closeness to databases etc. The two frequently used scheduling principles in a non pre-emptive system are the First-in-First-out (FIFO) and Weighted Round Robin (WRR) policies. These policies may end up with different degrees of loads on each and every VM. This may lead to load difference between VMs computing in parallel. This creates additional problems of reduction in response time, wastage of resources and so on.

These kinds of situations leads us to give more importance to the dynamic load balancing techniques which solves the problem of load imbalance between VMs. Load Balancing techniques are effective in reducing the makespan and response time.

Makespan can be defined as the overall task completion time. We denote completion time of task T_i on VM_j as CT_{ij}. Hence, the makespan is defined as the following function [17]:

$$\text{Makespan} = \max\{CT_{ij} | i \in T, \quad i = 1, 2, \ldots n \text{ and } j \in \text{VM}, \; j = 1, 2, \ldots m\} \tag{1}$$

Response time is the amount of time taken between submission of a request and the first response that is produced. The reduction in waiting time is helpful in improving responsiveness of the VMs.

4.1. Mathematical Model

Let $\text{VM} = \{\text{VM}_1, \text{VM}_2, \ldots \text{VM}_m\}$ be the set of m virtual machines which should process n tasks represented by the set $T = \{T_1, T_2, \ldots, T_n\}$. All the machines are unrelated and parallel and are denoted as R in the model. We schedule non-preemptive independent tasks to these VMs. Non-preemptive tasks are denoted as npmtn. Non-preemption of a task means that processing of that task on a virtual

machine cannot be interrupted (assuming that failure does not occur).

We denote finishing time of a task T_i by CT_i. Our aim is to reduce the makespan which can be denoted as CT_{max}. So our model is R|npmtn|CT_{max}.

Processing time of a task T_i on virtual machine VM_j can be denoted as P_{ij}.

Processing time of all tasks in a VM_j can be defined by Eq. (2).

$$P_j = \sum_{i=1}^{n} P_{ij} \quad j = 1, \ldots, m \tag{2}$$

By minimizing CT_{max}, we get Eq. (3). From Eq. (2) and (3) we can imply Eq. (4).

$$\sum_{i=1} P_{ij} \le CT_{max} \quad j = 1, \ldots, m \tag{3}$$

$$\Rightarrow P_j \le CT_{max} \quad j = 1, \ldots, m \tag{4}$$

At the time of load balancing, the tasks will be transferred from one VM to other in order to reduce CT_{max} as well as response time. Processing time of a task varies from one VM to other based on VM's capacity. In case of transferring, completion time of a task may vary because of load balancing.Optimally,

$$CT_{max} = \left\{ \max_{i=1}^{n} CT_i, \max_{j=1}^{n} \sum_{i=1}^{n} P_{ij} \right\} \tag{5}$$

Our load balancing technique, HBB-LB is a dynamic technique which not only balances the load but also considers the priorities of tasks in the waiting queues of VMs. Our algorithm is an extension of existing dynamic load balancing techniques merged with the concept of honey bee behavior.

The tasks removed from overloaded VMs act as Honey Bees. Upon submission to the under loaded VM, it will update the number of various priority tasks and load of tasks assigned to that VM. This information will be helpful for other tasks i.e., whenever a high priority has to be submitted to VMs, it should consider the VM that has minimum number of high priority tasks so that the particular task will be executed earlier. Since all VMs are sorted in an ascending order, the task removed will be submitted to under loaded VMs.

Current workload of all available VMs can be calculated based on the information received from the datacenter. Based on this, standard deviation has to be calculated to measure deviations of load on VMs.

4.1.1. Capacity of a VM

$$C_j = pe_{numj} \times pe_{mipsj} + vm_{bwj} \tag{6}$$

where processing element, pe_{numj} is the number processors in VM_j, pe_{mipsj} is million instructions per second of all processors in VM_j and vm_{bwj} is the communication bandwidth ability of VM_j.

4.1.2. Capacity of all VMs

$$C = \sum_{i=1}^{m} C_i \tag{7}$$

Summation of capacity of all VMs is the capacity of data center.

4.1.3. Load on a VM

Total length of tasks that are assigned to a VM is called load.

$$L_{V,M_i,t} = \frac{N(T,t)}{S(VM_i,t)} \tag{8}$$

Load of a VM can be calculated as the Number of tasks at time t on service queue of VM_i divided by the service rate of VM_i at time t.

Load of all VMs in a data center is calculated as

$$L = \sum_{i=1}^{m} L_{VM_i} \tag{9}$$

Processing time of a VM:

$$PT_i = \frac{L_{VM_i}}{C_i} \tag{10}$$

Processing time of all VMs:

$$PT = \frac{L}{C} \tag{11}$$

Standard deviation of load:

$$\sigma = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (PT_i - PT)^2} \tag{12}$$

4.1.3.1. Load balancing decision. After finding the workload and standard deviation, the system should decide whether to do load balancing or not. For this, there are two possible situations i.e., (1) Finding whether the system is balanced (2) Finding whether the whole system is saturated or not (The whole group is overloaded or not). If overloaded, load balancing is meaningless.

1. Finding State of the VM group
   If the standard deviation of the VM load (σ) is under or equal to the threshold condition set (Ts) [0–1] then the system is balanced [13]. Otherwise system is in an imbalance state. It may be overloaded or under loaded.

```
If σ ≤ Ts
    System is balanced
    Exit
```

2. Finding Overloaded Group
   When the current workload of VM group exceeds the maximum capacity of the group, then the group is overloaded. Load balancing is not possible in this case.

```
If L > maximum capacity
        Load balancing is not possible
Else
        Trigger load balancing.
```

4.1.3.2. VM grouping. The virtual machines will be grouped based on their loads. The groups are Overloaded VMs, under loaded VMs and balanced VMs. Each set contains the number of VMs. Task removed from one of overloaded VM set has to a make decision to get placed in one of several low loaded VMs based on the load and tasks available in the under loaded VM. In our technique, this task is considered as a honey bee and low loaded VMs are considered as the destination of the honey bees. The information the bees (tasks) update are load on a VM, load on all VMs, number of tasks in each VM, the number of VMs in each VM group (under loaded VM, over loaded VM, etc.,) and task priorities in each VM. Load balanced VMs are not used in switching of tasks. Once the task switching is over, the balanced VMs are included into the load balanced VM set. Once this set has all the VMs, the load balancing is successful i.e., all tasks are balanced.

4.1.3.3. Task transfer. If the decision is to balance the load, the scheduler should trigger the load balancing aspect. In order to perform load balancing, we have to find overloaded VMs, demand (load requirement), low-loaded VMs and supply (available load). After this, remove the tasks from overloaded VMs. In order to find the best VM to queue the removed task, we have to find the task priority. Tasks which are removed earlier (Scout bee) from over loaded VMs are helpful in finding the correct low loaded VM for current task (Forager bee). This Forager bee then becomes Scout bee for

next task. This process continues until the load balancing task is successful. VM selection is done as follows:

4.1.3.4. VM Selection of different prioritized tasks.

$$T_h \rightarrow \mathrm{VM}_d | \min\left(\sum T_h\right) \in \mathrm{VM}_d \quad (13)$$

$$T_m \rightarrow \mathrm{VM}_d | \min\left(\sum T_h + \sum T_m\right) \in \mathrm{VM}_d \quad (14)$$

$$T_l \rightarrow \mathrm{VM}_d | \min\left(\sum T\right) \in \mathrm{VM}_d \quad (15)$$

where T_h, T_m, T_l are the tasks of high, middle and low priority cadres respectively.

The priorities of tasks can be categorized in 3 cadres (high, middle, and low). When a high priority task has to be submitted to one of the under loaded machines, it has to consider the high priority tasks already submitted to that machine. This will ensure that the high priority task will find the machine which has less number of high priority tasks.

4.1.3.5. Algorithm HBB-LB. Workload information about each VMs in set VM.

1. Find capacity and loads of all VMs based on equations (2),(3),(4) and (5) Check system is balanced or not:
   If σ <= Ts
   System is balanced
   Exit.
2. Load Balancing Decision:
   If L > maximum capacity
   Load Balancing is not possible
   Else
   Trigger Load Balancing.
3. Group VMs based on load as LVM,BVM and OVM
4. Load Balancing:

Supply of Each machine in UVM is

$$Supply\ of LVM_j = Maximum\ capacity - \frac{\text{Load}}{\text{Capacity}}$$

Demand of Each machine in OVM is

$$Demand\ of\ OVM_j = \frac{\text{Load}}{\text{Capacity}} - Maximum\ Capacity.$$

Sort VMs in OVM by descending order.
Sort VMs in LVM by ascending order.
While LVM≠φ and OVM≠φ
For s=1 to # (OVM) do
Sort tasks in VMs by selection criterion (priority)
For each task T in VMs find machine VM_d ∈ LVM such as
If(T is non preemptive)

$$T_h \rightarrow VM_d | min(\sum T_h) \in VM_d\ and\ Load_{VM_d} \leq Capacity_{VM_d}$$

$$T_m \rightarrow VM_d | min(\sum T_h + \sum T_m) \in VM_d\ and\ Load_{VM_d} \leq Capacity_{VM_d}$$

$$T_l \rightarrow VM_d | min(\sum T) \in VM_d\ and\ Load_{VM_d} \leq Capacity_{VM_d}$$

If (T is preemptive)

$$T_h \rightarrow VM_d | min(\sum T_h) \in VM_d$$

$$T_m \rightarrow VM_d | min(\sum T_h + \sum T_m) \in VM_d$$

$$T_l \rightarrow VM_d | min(\sum T) \in VM_d$$

Update the number of tasks assigned to VM_d.
Update the number of priority tasks assigned VM_d.
Update Load on both VMs,VM_d.
Update sets OVM,LVM,BVM
Sort VMs in OVM by descending order.
Sort VMs in LVM by ascending order.

Here we define three sets based on load of the VMs. They are

LVM (Low loaded VM)—The set contains the VMs of load loaded.
OVM(Overloaded VM)—The set contains all overloaded VMs
BVM(Balanced VM)—Remaining all VMs are balanced and they are available in this set.

Table 1
Makespan in seconds before load balancing and after load balancing with HBB-LB.

No. of tasks	Before load balancing (s)	After load balancing (s)
10	11.4	5.7
20	27.5	15.5
30	36	19.6
40	57	28

5. Experimental results

A cloud computing system has to handle several hurdles like network flow, load balancing on virtual machines, federation of clouds, scalability and trust management and so on. Research in cloud computing generally focus on these issues with varying importance. Clouds offer a set of services (software and hardware) on an unprecedented scale. Cloud Services have to handle the temporal variation in demand through dynamic provisioning or de-provisioning from clouds. Considering all these, we cannot directly use the cloud computing system. Experimenting new techniques or strategies in real cloud computing operations is not practically possible as such experiments will compromise the end users QoS requirements like security, cost, speed. Chang et al. discuss about fast access security in Ubuntu clouds [40]. There is a need for a good simulator for experimental purposes. One such a simulator

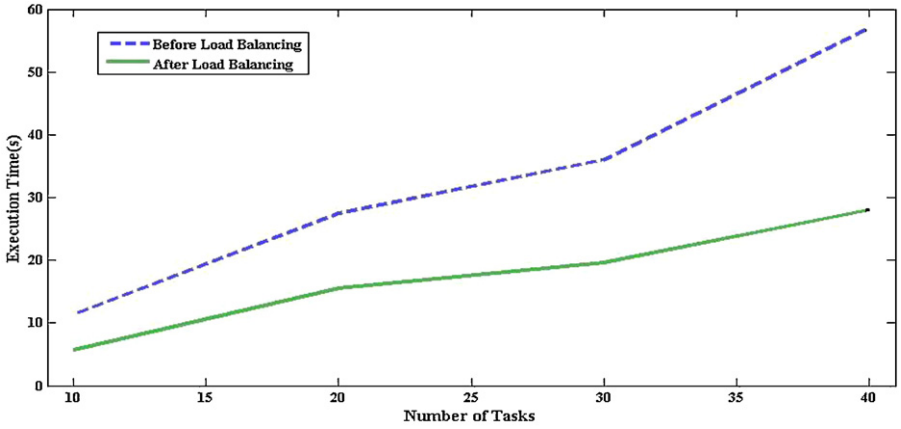

Fig. 3. Comparison of makespan before and after load balancing using HBB-LB.

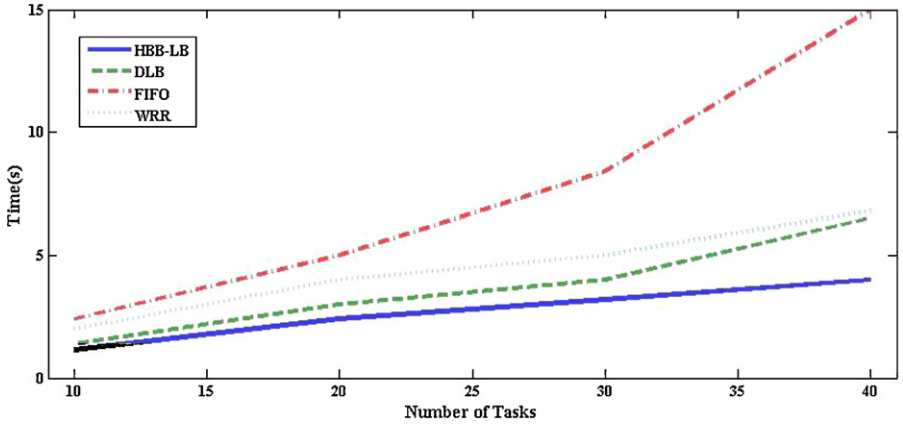

Fig. 4. Response time of VMs in seconds for HBB-LB, DLB, FIFO and WRR.

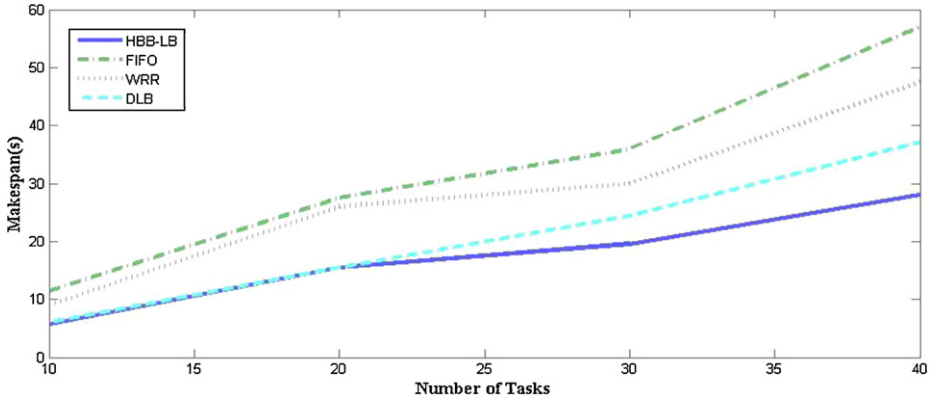

Fig. 5. Comparison of makespan for HBB-LB, FIFO, WRR and DLB algorithms.

is CloudSim [20–22]. This simulator is a generalized simulation framework that allows modeling, simulation and experimenting the cloud computing infrastructure and application services [20].

In this section, we have analyzed the performance of our algorithm based on the results of simulation done using CloudSim. We have extended the classes of CloudSim simulator to simulate our algorithm. In the following illustrations, we have compared the makespan of Weighted Round Robin(WRR), FIFO, Dynamic Load Balancing (DLB) [1,4] and our algorithm(HBB-LB) in different low and over loaded ratios.

Table 1 illustrates the makespan before load balancing and after load balancing with HBB-LB.

Fig. 3 illustrates the comparison of Makespan before and after Load balancing using HBB-LB. The *X*-axis represents the number of tasks and the *Y*-axis represents the Makespan (task execution

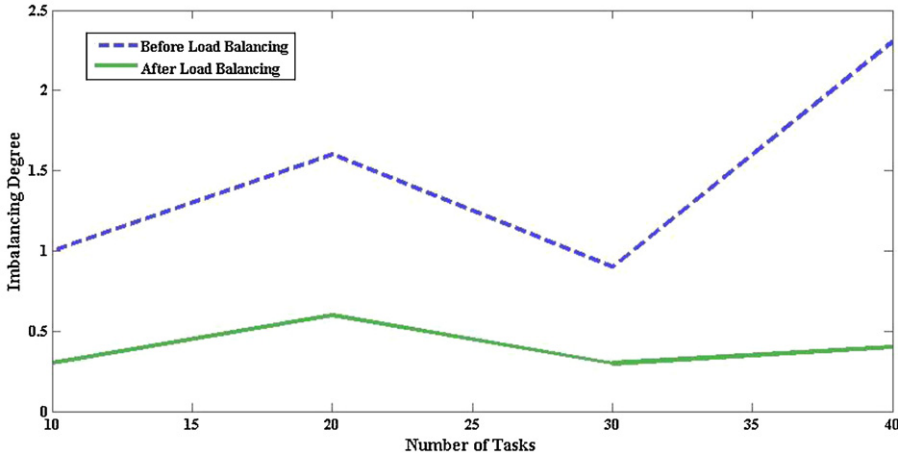

Fig. 6. Degree of imbalance between VMs before and after load balancing with HBB-LB.

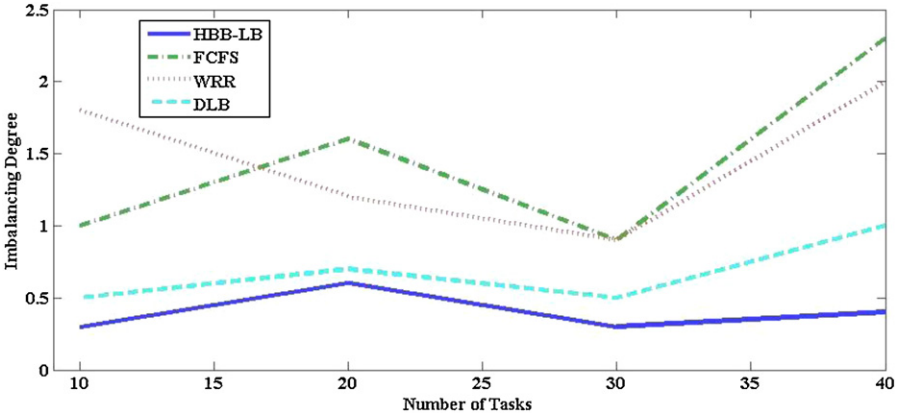

Fig. 7. Comparison between algorithms (HBB-LB, FCFS, WRR and DLB) based on degree of imbalance.

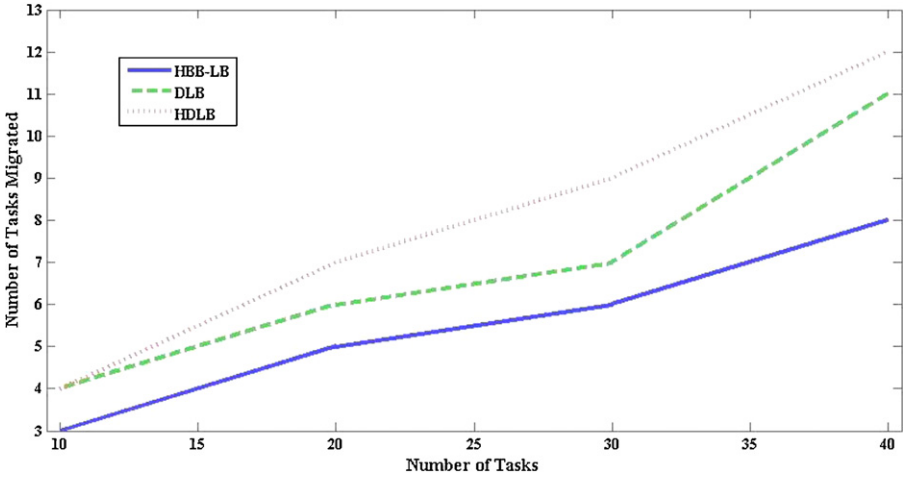

Fig. 8. Comparison of number of task migrations when there are 3 VMs.

and completion time) in seconds. With dynamic load balancing using honey bee behavior inspired load balancing (HBB-LB), the makespan is reduced considerably. With more number of tasks, the difference in makespan time is quite high and HBB-LB provides the best results. Fig. 4 illustrates the response time of VMs in seconds for HBB-LB, DLB, FIFO and WRR Algorithms. The *X*-axis represents number of tasks and the *Y*-axis represents time in seconds. It is evident that HBB-LB is more efficient compared with other three methods.

Fig. 5 shows the comparison of Makespan for HBB-LB, FIFO and WRR, DLB. The *X*-axis shows the number of tasks and the *Y*-axis shows makespan in seconds. It is clearly evident from the graph that HBB-LB is more efficient when compared with other 3 algorithms. We used around 500 tasks for our comparisons. We have

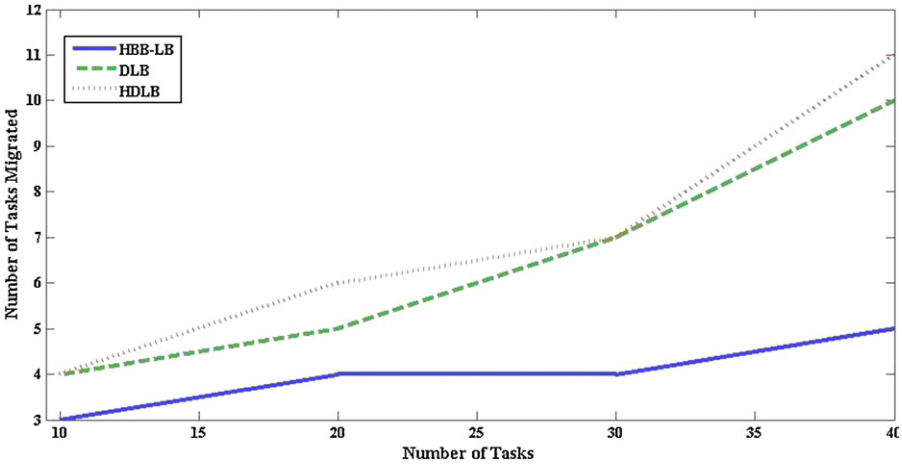

Fig. 9. Comparison of number of task migrations when there are 4 VMs.

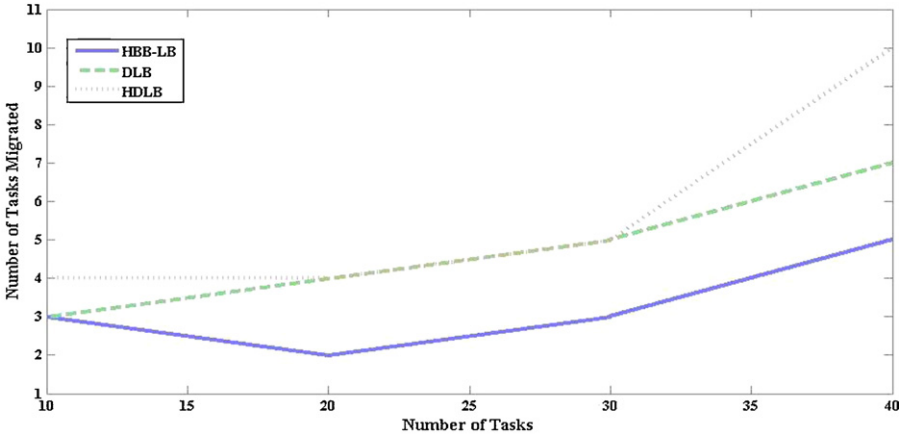

Fig. 10. Comparison of number of task migrations when there are 5 VMs.

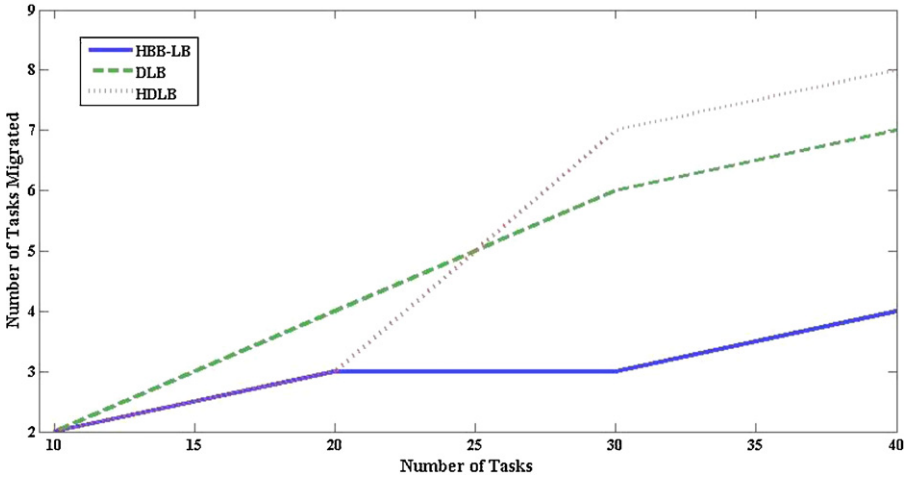

Fig. 11. Comparison of number of task migrations when there are 6 VMs.

also compared the degree of imbalance [18] in load between VMs for all 4 algorithms.

6. Degree of imbalance

$$\mathrm{DI} = \frac{T_{\max} - T_{\min}}{T_{\mathrm{avg}}} \tag{16}$$

where $T_{\max}$ and $T_{\min}$ are the maximum and minimum T_i among all VMs, T_{avg} is the average T_i of VMs. Our load balancing system reduces the degree of imbalance drastically.

Fig. 6 shows the degree of imbalance between VMs before and after load balancing with HBB-LB. The *X*-axis represents number of tasks and the *Y*-axis represents the degree of imbalance. It is clearly evident that after load balancing with HBB-LB, the degree of imbalance is greatly reduced. Fig. 7 shows the comparison of degree of imbalance between HBB-LB, FIFO, DLB and WRR Algorithms. The *X*-axis represents number of tasks and the *Y*-axis represents the degree of imbalance. HBB-LB is more efficient and has a lesser degree of imbalance when compared with other three algorithms.

We have also compared the task migration balancing between VMs finally. Task migration is number of tasks reassigned between VMs. All these results show that our algorithm performs better than the DLB and HDLB algorithms.

Figs. 8–12 shows task migration when numbers of VMs are varied from 3 to 7 for HBB-LB, DLB and HDLB techniques. In all the five cases it is clearly evident that the task migration is very less compared with other two popular techniques irrespective of the number of VMs. Fig. 13(a)–(d) shows the comparison of task migration vs. number of virtual machines when number of tasks are

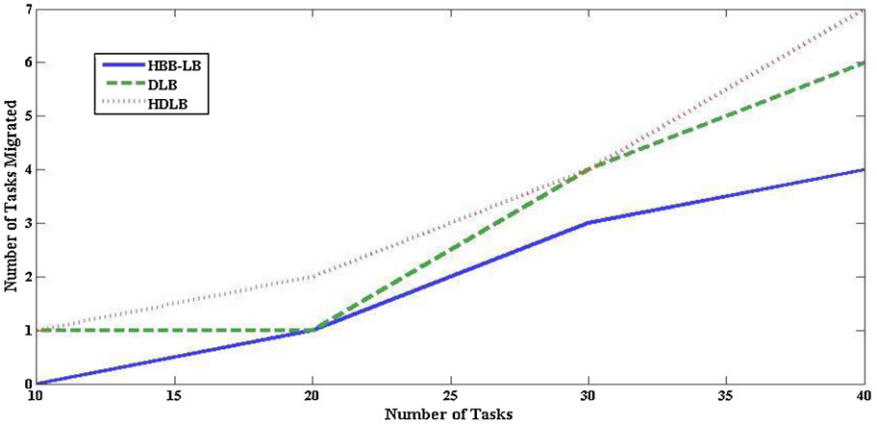

Fig. 12. Comparison of number of task migrations when there are 7 VMs.

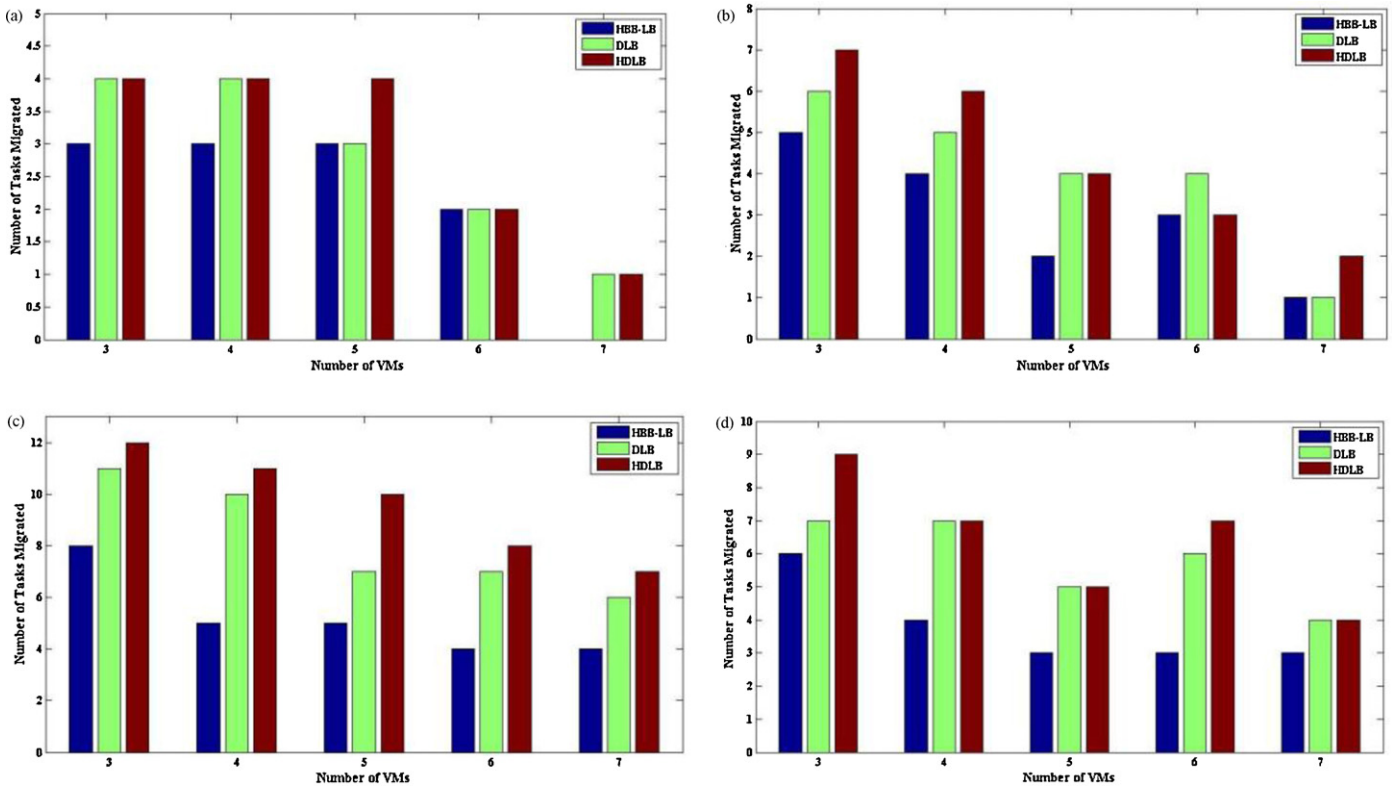

Fig. 13. (a) Comparison of number of task migrations vs. number of virtual machines for a set of 10 tasks. (b) Comparison of number of task migrations vs. number of virtual machines for a set of 20 tasks. (c) Comparison of number of task migrations vs. number of virtual machines for a set of 30 tasks. (d) Comparison of number of task migrations vs. number of virtual machines for a set of 40 tasks.

varied from 10 to 40. Results illustrate that HBB-LB is more efficient with lesser number of task migrations when compared with DLB and HDLB techniques.

7. Conclusion

In this paper, we have proposed a load balancing technique for cloud computing environments based on behavior of honey bee foraging strategy. This algorithm not only balances the load, but also takes into consideration the priorities of tasks that have been removed from heavily loaded Virtual Machines. The tasks removed from these VMs are treated as honey bees, which are the information updaters globally. This algorithm also considers the priorities of the tasks. Honey bee behavior inspired load balancing improves the overall throughput of processing and priority based balancing focuses on reducing the amount of time a task has to wait on a queue of the VM. Thus, it reduces the response of time of VMs. We have compared our proposed algorithm with other existing techniques. Results show that our algorithm stands good without increasing additional overheads. This load balancing technique works well for heterogeneous cloud computing systems and is for balancing non-preemptive independent tasks. In future, we plan to extend this kind of load balancing for workflows with dependent tasks. This algorithm considers priority as the main QoS parameter. In future, we plan to improve this algorithm by considering other QoS factors also.

References

[1] B. Yagoubi, Y. Slimani, Task load balancing strategy for grid computing, Journal of Computer Science 3 (3) (2007) 186–194.
[2] A. Revar, M. Andhariya, D. Sutariya, M. Bhavsar, Load balancing in grid environment using machine learning-innovative approach, International Journal of Computer Applications 8 (10 (Oct)) (2010) 975–8887.
[3] M. Randles, A. Taleb-Bendiab, D. Lamb, Scalable self governance using service communities as ambients, in: Proceedings of the IEEE Workshop on Software and Services Maintenance and Management (SSMM 2009) within the 4th IEEE Congress on Services, IEEE SERVICES-I 2009, July 6–10, Los Angeles, CA (to appear), 2009.
[4] B. Yagoubi, Y. Slimani, Dynamic load balancing strategy for grid computing, transactions on engineering, Computing and Technology 13 (May) (2006) 260–265.
[5] B. Yagoubi, M. Medebber, A load balancing model for grid environment, computer and information sciences, 2007. iscis 2007, in: 22nd International Symposium on, 7–9 Nov, 2007, pp. 1–7.
[6] Y. Hu, R. Blake, D. Emerson, An optimal migration algorithm for dynamic load balancing, Concurrency: Practice and Experience 10 (1998) 467–483.
[7] M. Moradi, M.A. Dezfuli, M.H. Safavi, Department of Computer and IT, Engineering, Amirkabir University of Technology, Tehran, Iran, A New Time Optimizing Probabilistic Load Balancing Algorithm in Grid Computing IEEE 978-1-4244-6349-7/10/©2010.
[8] M. Randles, D. Lamb, A. Taleb-Bendiab, A comparative study into distributed load balancing algorithms for cloud computing, in: Proceedings of 24th IEEE International Conference on Advanced Information Networking and Applications Workshops, Perth, Australia, April, 2010, pp. 551–556.
[9] M. Houle, A. Symnovis, D. Wood, Dimension-exchange algorithms for load balancing on trees, in: Proc. of 9th Int. Colloquium on Structural Information and Communication Complexity, Andros, Greece, June, 2002, pp. 181–196.
[10] D.L. Eager, E.D. Lazowska, J. Zahorjan, Adaptive load sharing in homogeneous distributed systems, The IEEE Transactions on Software Engineering 12 (5) (1986) 662–675.
[11] H.D. Karatza, Job scheduling in heterogeneous distributed systems, Journal of Systems and Software 56 (1994) 203–212.
[12] S. Genaud, A. Giersch, F. Vivien, Load balancing scatter operations for grid computing, in: Proceedings of the 12th Heterogeneous Computing Workshop (HCW'2003), Nice, France, April, 2003, pp. 101–110.
[13] R.F. de Mello, L.J. Senger, L.T. Yang, A routing load balancing policy for grid computing environments, in: 20th International Conference on, vol. 1, 18–20 April, Advanced Information Networking and Applications, 2006. AINA 2006. (2006) 6.
[14] B. Yagoubi, Distributed load balancing model for grid computing, in: African Conference on Research in Computer Science and Applied mathematics, October, 2008, pp. 631–638.
[15] C. Zhao, S. Zhang, Q. Liu, J. Xie, J. Hu, Independent tasks scheduling based on genetic algorithm in cloud computing, wireless communications. networking and mobile computing, 2009. WiCom '09, in: 5th International Conference on, 24–26 Sept, 2009, pp. 1–4.
[16] N. Malarvizhi, V. Rhymend Uthariaraj, Hierarchical load balancing scheme for computational intensive jobs in Grid computing environment, in: Advanced Computing, 2009. ICAC 2009. First International Conference on, 13–15 Dec, 2009, pp. 97–104.
[17] P. Brucker, Scheduling Algorithms, 2nd ed., Springer-Verlag, Berlin, Heidelberg, Germany, 1997.
[18] K. Li, G. Xu, G. Zhao, Y. Dong, D. Wang, Cloud task scheduling based on load balancing ant colony optimization, in: Chinagrid Conference (ChinaGrid), 2011 Sixth Annual, 22–23 Aug, 2011, pp. 3–9.
[19] S. Nakrani, C. Tovey, On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers, Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems 12 (3–4 (Sep–Dec)) (2004) 223–240.
[20] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (2011) 23–50, http://dx.doi.org/10.1002/spe.995.
[21] R.N. Calheiros, R. Ranjan, C.A.F.D. Rose, R. Buyya, CloudSim: a novel framework for modeling and simulation of cloud computing infrastructures and services, Computing Research Repository, vol. abs/0903.2525, 2009.
[22] R. Buyya, R. Ranjan, R.N. Calheiros, Modeling simulation of scalable cloud computing environments and the cloudsim toolkit: challenges and opportunities, in: Proceedings of the 7th High Performance Computing and Simulation Conference (HPCS 2009, ISBN: 978-1-4244-4907-1, IEEE Press, New York, USA), Leipzig, Germany, June 21–24, 2009.
[23] K. Mukherjee, G. Sahoo, Mathematical model of cloud computing framework using fuzzy bee colony optimization technique, in: Proceedings of the 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies, December 28–29, 2009, pp. 664–668.
[24] B.R. Johnson, J.C. Nieh, Modeling the adaptive role of negative signaling in honey bee intraspecific competition, Journal of Insect Behavior 23 (2010) 459–471.
[25] H. de Vries, J.C. Biesmeijer, Modelling collective foraging by means of individual behaviour rules in honey-bees, Behavioral Ecology and Sociobiology 44 (1998) 109–124.
[26] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey, 2005.
[27] R.S. Rao, S.V.L. Narasimham, M. Ramalingaraju, Optimization of distribution network configuration for loss reduction using artificial bee colony algorithm, International Journal of Electrical Power and Energy Systems Engineering 1 (2008) 116–122.
[28] N. Karaboga, M.B.C. etinkaya, A novel and efficient algorithm for adaptive filtering: artificial bee colony algorithm, Turkish Journal of Electrical Engineering & Computer Sciences Vol. 19 (2011) 175–190.
[29] A. Singh, An artificial bee colony algorithm for the leaf constrained minimum spanning tree problem, Applied Soft Computing Journal 9 (2) (2009) 625–631.
[30] Q.K. Pan, M.F. Tasgetiren, P. Suganthan, T. Chua, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Information Sciences 181 (12) (2011) 2455–2468.
[31] F. Kang, J. Li, Q. Xu, Structural inverse analysis by hybrid simplex artificial bee colony algorithms, Computers and Structures 87 (13) (2009) 861–870.
[32] D.T. Pham, E. Kog, A. Ghanbarzadeh, S. Otri, S. Rahim, M. Zaidi, The bees algorithm—a novel. Tool for complex optimisation problems, in: IPROMS 2006 Proceeding 2nd InternationalVirtual Conference on Intelligent Production Machines and Systems, Oxford, Elsevier, 2006.
[33] T.D. Seeley, The Wisdom of the Hive, Harvard University Press, Cambridge, MA, 1995.
[34] T.D. Seeley, Honey bee foragers as sensory units of their colonies, Behavioral Ecology and Sociobiology 34 (1994) 51–62.
[35] J.-L. Deneubourg, S. Goss, R. Beckers, G. Sandini, A. Babloyantz, Self-Organization, Emergent Properties, and Learning, Plenum Press, New York, 1991, p. 267.
[36] B.B. Werger, M.J. Mataríc, From animals to animats 4, in: P. Maes, M. Mataríc, J.-A. Meyer, J. Pollack, S.W. Wilson (Eds.), Fourth Intern. Conf. on Simulation of Adaptive Behavior, MIT Press, Bradford Books, 1996, p. 625.
[37] A.M. Bernardino, E.M. Bernardino, J.M. Sánchez-Pérez, M.A. Vega-Rodríguez, J.A. Gómez-Pulido, Efficient Load Balancing Using the Bees Algorithm, Trends in Applied Intelligent Systems, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2011.
[38] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing 8 (1) (2008) 687–697.
[39] E. Cuevas, D. Zaldívar, M. Pérez-Cisneros, H. Sossa, V. Osuna, Block matching algorithm for motion estimation based on Artificial Bee Colony (ABC), Applied Soft Computing (2012), http://dx.doi.org/10.1016/j.asoc.2012.09.020.
[40] B. Chang, H. Tsai, C.F. Huang, Z.Y. Lin, C.M. Chen, Fast access security on cloud computing: ubuntu enterprise server and cloud with face and fingerprint identification, in: Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science, Springer, Berlin, Heidelberg, 2012, pp. 451–457.
[41] P.-W. Tsai, M.K. Khan, J.-S. Pan, B.-Y. Liao, Interactive artificial bee colony supported passive continuous authentication system, Systems Journal, IEEE, vol. PP, no.99, pp.1, 0. http://dx.doi.org/10.1109/JSYST.2012.2208153
[42] P.W. TSai, J.S. Pan, B.Y. Liao, S.C. Chu, Enhanced artificial bee colony optimization, International Journal of Innovative Computing, Information and Control 5 (12) (2009) 5081–5092.

Dhinesh Babu L.D. received B.E in Electrical and Electronics Engineering and M.E degree in Computer Science and Engineering from the University of Madras in 1998 and 2001, respectively. He is working toward his PhD at VIT University. He is currently a faculty in Software Engineering Division of the School of Information Technology and Engineering at VIT University, Vellore, India. He has served as Division Leader of Software Engineering Division. His research interests include Cloud Computing, Grid and Distributed Computing, Computer and Software Security, Software Engineering, ERP, Business Information Systems and Supply chain Management.

P. Venkata Krishna received the B.Tech. degree in electronics and communication engineering from Sri Venkateswara University, Tirupathi, India, the M.Tech. degree in computer science and engineering from the Regional Engineering College, Calicut, India, and the Ph.D. degree from VIT University, Vellore, India. He is currently a Professor in the School of Computing Sciences, VIT University. His research interests include mobile, wireless systems, grid computing and cloud computing.