Honey Bee Based Load Balancing in Cloud Computing

Walaa Hashem, Heba Nashaat, and Rawya Rizk*
Electrical Engineering Department, Port Said University, Port Said, 42523, Egypt
hebanashaat@eng.psu.edu.eg, r.rizk@eng.psu.edu.eg
*Corresponding author: Rawya Rizk

Received November 19, 2016; revised August 24, 2017; accepted October 22, 2017; published December 31, 2017

Abstract

The technology of cloud computing is growing very quickly, thus it is required to manage the process of resource allocation. In this paper, load balancing algorithm based on honey bee behavior (LBA_HB) is proposed. Its main goal is distribute workload of multiple network links in the way that avoid underutilization and over utilization of the resources. This can be achieved by allocating the incoming task to a virtual machine (VM) which meets two conditions; number of tasks currently processing by this VM is less than number of tasks currently processing by other VMs and the deviation of this VM processing time from average processing time of all VMs is less than a threshold value. The proposed algorithm is compared with different scheduling algorithms; honey bee, ant colony, modified throttled and round robin algorithms. The results of experiments show the efficiency of the proposed algorithm in terms of execution time, response time, makespan, standard deviation of load, and degree of imbalance.

Keywords: Cloud computing; honey bee; load balancing; swarm intelligence; virtual machine

1. Introduction

Cloud computing provides shared processing resources and data. This can occur through the presence of a host application service provider so that the user does not need to buy a server or pay for the electricity of power and cooling. It's also convenient for communications and travels where remote workers, who can simply log in and use their applications wherever they are [1]. As increasing the number of users in cloud computing environment, the demand of shared resources is rapidly increased. Therefore, load balancing between these resources for scheduling tasks becomes a key challenge.

Load balancing is the process of distributing workloads and computing resources in a cloud computing environment. It allows enterprises to manage application or workload demands by allocating resources among multiple computers, networks or servers. Load balancing is often used to avoid the bottleneck, so that several characteristics of load balancing can be achieved such as: equal division of tasks across all hosts, facilitation in achieving service quality, improve overall performance of the system, reduce response time, and improve resource utilization [2].

Fig. 1 shows the load balancer of virtual machines (VMs). It assigns multiple tasks to VMs that execute them simultaneously by a way that guarantees a balance between these VMs. The primary goal of load balancing in a cloud environment is to balance the workload of the hosts in proportion to their capacities, which is measured in terms of their processor speed, available memory space, and bandwidth.

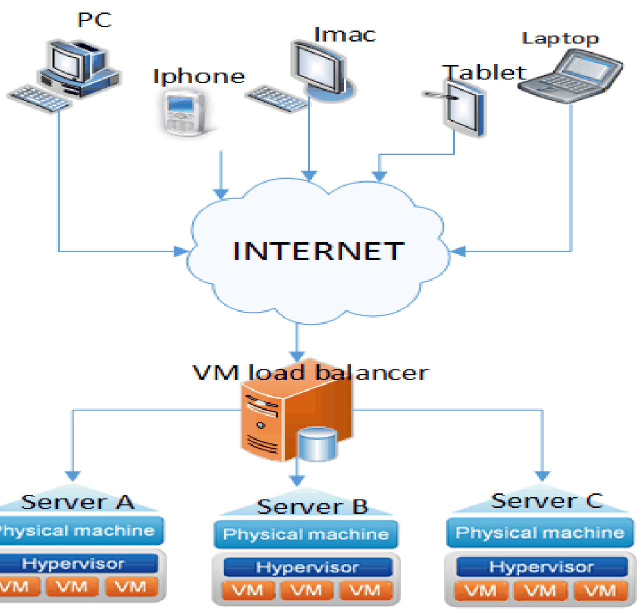

Fig. 1. VM Load Balancer.

Load balancing algorithms are classified into two types; static and dynamic. Static algorithms are much simpler as compared to dynamic algorithms. Static algorithms work properly only when hosts have low variations in the load, since they do not take into account the previous state or the behavior of a host while distributing the load. Dynamic load balancing algorithms are more suitable for widely distributed systems such as cloud computing [3,4]. Round robin (RR) [5] is a well-known straightforward static scheduling algorithm. It allocates tasks to each node in turn, without considering the resource quantity of each VM and the execution time of tasks. Modified throttled algorithm [6] is a dynamic load balancing algorithm that uniformly distributes the incoming tasks among available VMs. However it doesn't consider resource utilization during task allocation.

Conventional load balancing algorithms have many drawbacks in cloud environment due to the changing workload dynamics. To address these challenges, Swarm Intelligence algorithms (SI), such as ant colony optimization (ACO), and artificial bee colony (ABC), are provided in recent decades [7]. They achieve a great progress in the dynamic situation of cloud computing. So many researches tended to study the algorithms based on SI to balance load among cloud environment such as foraging for food. However, some of these algorithms have drawbacks such as causing many hosts overloaded, and getting low throughput.

The objective of this paper is to propose a load balancing algorithm aims to distribute the dynamic workload smoothly to all the hosts in the cloud to gain an improvement in both the utilization of resources and the speed of execution time. It allocates the incoming tasks to all available VMs. In order to achieve fairness and avoid congestion, the proposed algorithm allocates tasks to the least loaded VM and prevents the allocation of tasks to a VM when the variation of this VM processing time from average processing time of all VMs becomes more than or equal to a threshold value. This leads to a reduction of the overall response time and the processing time of hosts. In the proposed algorithm, variation of processing time of VM is the key limiting factor during the task allocation process because it avoids underutilization and over utilization of VMs. It also has a highly effect of the standard deviation that preserves the load balance of all system.

In this paper, a Load Balancing Algorithm based on Honey Bee behavior (LBA_HB) is proposed. It is completely inspired by the natural foraging behavior of honey bees. The allocated task updates the remaining tasks about the VM status in a manner similar to the bees finding an abundant food source, updating the other bees in the bee hive through its waggle dance [8]. The proposed LBA_HB algorithm has been simulated using CloudSim [9]. The proposed algorithm is compared with both conventional and SI based load balancing algorithms; round robin, modified throttled, ant colony, and honey bee algorithms. The results of experiments show the efficiency of LBA_HB in terms of response time, makespan, standard deviation of load, and degree of imbalance.

The rest of the paper is organized as follows. In Section 2, related study of load balancing in cloud computing is introduced. Section 3 presents the proposed load balancing algorithm in cloud computing. In Section 4, the algorithm implementation using CloudSim is explained and the simulation results are introduced. Finally, conclusion is introduced in Section 5.

2. Related Works

Millions of users share cloud resources by submitting their computing tasks to the cloud system. Scheduling these millions of tasks is a challenge to cloud computing environment. Researches in SI discovered that cooperation of groups of similar agents in the cloud computing environment can solve complicated problems. The most popular load balancing algorithms in SI field are ACO and ABC. SI based scheduling algorithms survey was presented in [10] for a number of tasks on distributed environment. It uniformly compares between tasks' applications in distributed computing environments based on a derived comparison framework. It represents SI schedulers, which deal with optimizing one or more scheduling metrics in distributed environments, such as makespan and load balancing.

Cloud task scheduling policy based on ACO algorithm was presented in [11]. The main goal of this algorithm is minimizing the makespan of the tasks. ACO is random optimization search approach that is used for allocating the incoming jobs to VMs. It uses a positive feedback mechanism, inner parallelism, and extensible. However, there are some drawbacks such as the overhead which results due to using more than one control parameter to map the relative importance of quantity of pheromone and the desirability of each movement. In addition, the stagnation phenomenon that results in finding the same solution exactly when searching for certain individuals. Soft computing based algorithm on ACO was introduced in [12] which uses the concept of foraging and trailing pheromones for searching over loaded and under loaded nodes. As compared to original ACO approach where ants build their own solutions and afterward build into a whole solution, ants in this algorithm continuously update a single result set instead of updating their individual solutions. An ant colony based load balancing strategy was proposed in [13]. In this algorithm, ants are formed and detached in the cloud seeking under loaded VMs in order to achieve the balance. However, it does not consider fault tolerance issues and all jobs are predicted with same priority.

ABC algorithm which is based on the foraging behavior of bees is presented in [14]. In this algorithm, the artificial bees are classified into three groups: employed bees, onlookers and scouts. Employed bees represent the first half of the colony, while the onlooker occupies the other half. A number of drawbacks of this algorithm include lack of use of secondary information, the possibility of losing relevant information, high number of objective function evaluations, slow down when used in sequential processing, and the population of solutions increases the computational cost.

Load balancing algorithm in cloud computing environments based on behavior of honey bee foraging strategy was proposed in [15]. The tasks are sent to the under loaded machine and like foraging bee the next tasks are also sent to that VM till the machine gets overloaded as flower patches exploitation is done by scout bees. However, this algorithm does not consider VM bandwidth and VM cost during load balancing in inter datacenter level.

Cost effective load balancing based on honey bee behavior in cloud environment was proposed in [16]. It selects optimal VM by comparing the cost of executing a task on one VM with that of all other VMs and expected running time of that task in one VM with that in all other VMs. Finally, it selects a VM which has minimum value of minimization function and assigns the task to it. The minimization function is computed based on running time and monitory cost. This technique causes high number of migrations that decrease the quality and the performance of the overall system.

An enhanced bee colony algorithm for load balancing in cloud was proposed in [17]. This approach eliminates the tasks from over-loaded VMs and assigns it to the most suitable under-loaded VMs. That approach also considers the tasks' priorities in the VMs queues, as it selects the task with least priority for migration in order to reduce imbalance. So no tasks are needed to wait longer time in order to get processed. However, the number of task migrations is high which adversely affects the performance of the cloud.

A honey bee behavior inspired load balancing (HBB-LB) algorithm is presented in [18]. This algorithm aims to balance load across VMs and minimize the makespan in cloud environment. In this algorithm, the VMs are grouped into three groups based on their loads; over-loaded VMs, under-loaded VMs, and balanced VMs. The algorithm switches jobs from over-loaded VMs, and takes the decision of submitting them to one of the under-loaded VMs. A job is considered as a honey bee and the under-loaded VMs are considered as the destination of the honey bees. The information that bees update are load on a VM, load on all VMs, number of jobs in each VM, and number of VMs in each set. Once the jobs switching process is over, the balanced VMs are included into the balanced VM set. Once this set has all the VMs, the load balancing process ends. In this algorithm, the load balancing process starts when the system becomes unbalanced which yields to migrating tasks from overloaded VM to under loaded VM, thus increasing the number of task migrations. The HBB-LB algorithm balances the load of VMs when all the system becomes unbalanced. This is done by checking the value of the standard deviation of load which expresses the balance of load for VMs of all the system. Once this value becomes more than or equal to specific condition, the load balancing process starts which migrates tasks from overloaded VM to under loaded VM. Then, the running task is interrupted for some times and resumed later which is called a pre-emptive system.

In this paper, the proposed LBA_HB algorithm tends to balance the load of VMs during task allocation by checking the variation of each VM processing time from average processing time of all VMs. Once the value of specific VM becomes more than or equal to a predefined threshold, it means that this VM becomes overloaded at this time. Then, the algorithm starts the load balancing process which limits allocation to overloaded VM. Then, the running task is executed till completion which is called a non-pre-emptive system.

3. The Proposed Load Balancing Algorithm Based on Honey Bee Behavior

In bee hives, foraging honeybees give information to other bees about the position of the food source they have visited. A potential forager bee starts her career as an unemployed naive worker, that is, she has as no information of a food source in the field yet. She can start search for a source and thus become a scout (explorer). The initiation to fly out and start foraging is not due to following a waggle dance but due to some unknown internal, motivational factor or perhaps to some unknown external cue. Alternatively, a bee can start searching for a source as a response to attending a waggle dance and thus becomes a recruit. So the distinction between a recruit and a scout is that the recruit has stored estimated positional information in her memory, whereas the scout has not. As soon as a bee finds a source, it registers the essentials of this source in its memory and starts exploiting it, the bee is then an employed forager (exploiter) [8].

The proposed LBA_HB is completely inspired by the natural foraging behavior of honey bees. The allocated task updates the remaining tasks about the VM status 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 task updates the status of the VM availability and the load of the VMs.

3.1. Metrics of LBA_HB algorithm

The main goal of the proposed LBA_HB is to distribute workload in the way that avoid underutilization and over utilization of the resources. It allocates the incoming task to a VM which meets two conditions; number of tasks currently processing by this VM is less than number of tasks currently processing by other VMs and the deviation of this VM processing time from average processing time of all VMs is less than a predefined threshold value. The notations used in the LBA_HB are shown in **Table 1**.

Table 1. List of Notations

Symbol	Definition
σ	Load standard deviation
α	Threshold value.
$Count\_REQ_{VM}(j)$	The count of requests currently executed by VM(*j*).
DI	The degree of imbalance.
$host\ (i)$	The host number *i*
$List\_VM_{host}$	data structure contain list of VM in each host
$List\_VM_{host}(i)$	list of VM in host(*i*)
$List\_state_{host}$	data structure contain State [available-busy] of each host
$List\_state_{VM}$	data structure contain State [available-busy] of each VM
m	The number of VMs in specified host
n	The number of hosts in specified data center
$N\_PR_{host}(i)$	The number of processors in host(*i*)
$N\_PR_{vm}(j)$	The number of processors in VM(*j*)
PT_{Avg_host}	Average processing time for all hosts
PT_{Avg_vm}	Average processing time of all VMs
$PT_{host}(i)$	Processing time of host(*i*)
PT_{max_vm}	The maximum processing time among all VMs.
PT_{min_vm}	The minimum processing time among all VMs.
$PT_{VM}(j)$	Processing time of VM(*j*)
$REQ_{length}(k)$	The length of request number *k*
$REQ_{VM}(j)$	The current allocated request to VM(*j*).
$SND_{vm}(j)$	Standard normal deviate of VM(*j*)
$S\_PR_{host}(i)$	The processor speed of host(*i*) (MIPS)
$S\_PR_{vm}(j)$	The processor speed of VM(*j*) (MIPS)
$State_{host}(i)$	State of host(*i*)
$State_{VM}(j)$	State of VM(*j*)
$TL_{host}(i)$	Total length of tasks submitted to host(*i*)
$TL_{VM}(j)$	Total length of tasks submitted to VM(*j*)
$VM\ (j)$	The VM number *i*

Cloud computing consists of a set of data centers, each data center consists of a set of *n* hosts, each host consists of a set of *m* VMs. Each data center contains VM load balancer which responsible for finding a suitable host and a suitable VM in a chosen host to allocate the next task by finding some metrics. These metrics are calculated according to Equations (1-6) as follows:

- The processing time of host(*i*):

$$PT_{host}\ (i) = \frac{TL_{host}(i)}{N\_PR_{host}(i) \times S\_PR_{host}(i)} = \frac{\sum_{k=1}^{x1} REQ_{length}(k)}{N\_PR_{host}(i) \times S\_PR_{host}(i)} \tag{1}$$

- Average processing time of all hosts:

$$PT_{Avg\_host} = \frac{1}{n}\sum_{i=1}^{n} PT_{host}\ (i) \tag{2}$$

- Processing time of VM(*j*):

$$PT_{VM}(j) = \frac{TL_{VM}(j)}{N\_PR_{VM}(j) \times S\_PR_{VM}\ (j)} = \frac{\sum_{k=1}^{x2} REQ_{length}(k)}{N\_PR_{VM}(j) \times S\_PR_{VM}(j)} \tag{3}$$

- Average processing time of all VMs:

$$PT_{Avg\_VM} = \frac{1}{m}\sum_{j=1}^{m} PT_{VM}(j) \quad (4)$$

- Load standard deviation:

  In statistics and probability theory, standard deviation (σ) shows how much variation or dispersion exists from the average, which is defined as follows:

$$\sigma = \sqrt{\frac{1}{m}\sum_{j=0}^{m}(PT_{VM(j)} - PT_{Avg\_VM})^2} \quad (5)$$

- Standard normal deviate of VM(j):

  In mathematics and statistics, deviation is a measure of difference between the observed value and the mean. The sign of deviation (positive or negative), reports the direction of that difference (it is larger when the sign is positive, and smaller if it is negative). The magnitude of the value indicates the size of the difference.

  The division of distance of one data point from its mean to the standard deviation of the distribution is known as normal deviate or the standardized value. A unit deviation with zero mean is standard normal deviation and it shows the variation from the average mean or the expected value.

$$SND_{vm}(j) = \frac{(PT_{VM}(j) - PT_{Avg\_VM})}{\sigma} \quad (6)$$

- The availability of VM(j):

  The availability of VM is decided when the variation of VM processing time from average processing time of all VMs is equal to or less than a threshold value.

3.2. LBA_HB Control Structure

In this section, The LBA_HB control structure is explained. A flow diagram that describes the control structure for the LBA_HB is depicted in **Fig. 2**. It consists of four stages called A, B, C, and D. The input of the Stage A is a group of tasks that are called workflow. This stage prepares the individual tasks to go through the other stages of the algorithm by calculating the attributes of the current workflow that includes the number of tasks and the instruction length of each task.

The algorithm uses these attributes for two main functions; to prioritize among VMs and also to avoid the overloaded VMs to be allocated. The priority of the allocation of each task in the suitable VM mainly considers the load of each VM that depends on the instruction length of the tasks. The algorithm realizes load balance among VMs by limiting more allocation to overloaded VMs that have variation value in processing time more than or equal to a threshold.

The four stages of the algorithm are described as follows:

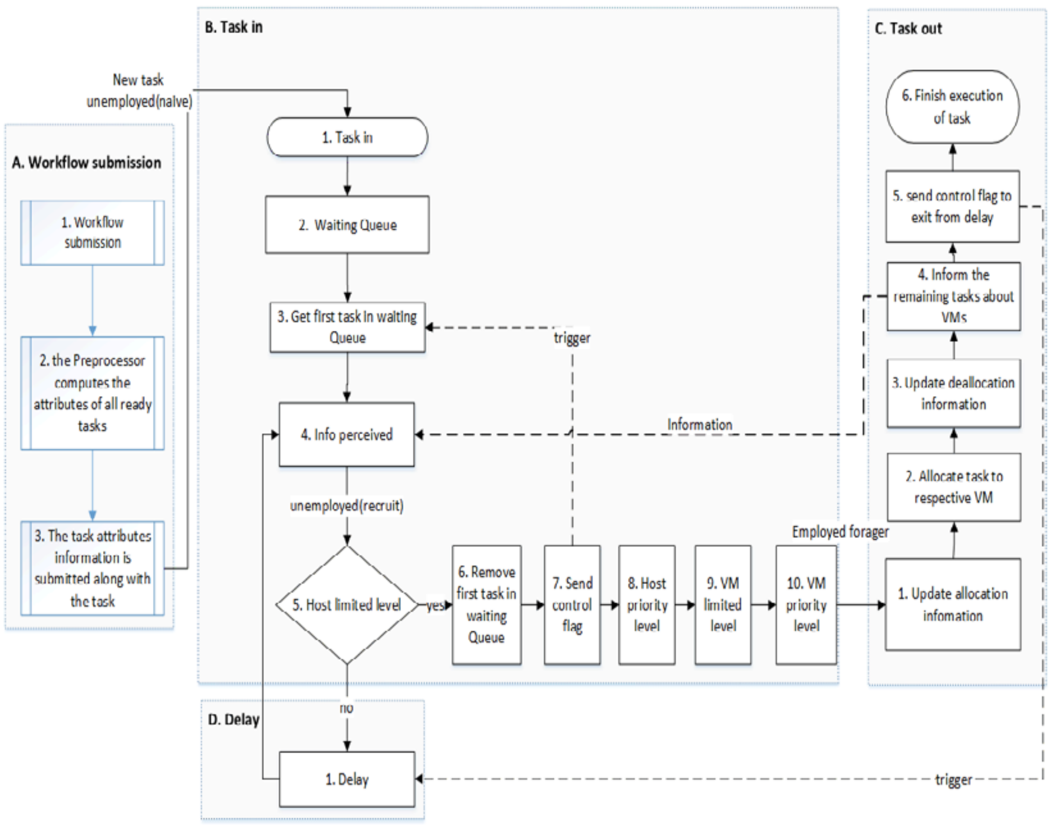
B. Task in
C. Task out
New task
unemployed(naive)
A. Workflow submission
1. Workflow submission
2. the Preprocessor computes the attributes of all ready tasks
3. The task attributes information is submitted along with the task
1. Task in
2. Waiting Queue
3. Get first task in waiting Queue
trigger
4. Info perceived
Information
unemployed(recruit)
5. Host limited level
yes
no
6. Remove first task in waiting Queue
7. Send control flag
8. Host priority level
9. VM limited level
10. VM priority level
Employed forager
6. Finish execution of task
5. send control flag to exit from delay
4. Inform the remaining tasks about VMs
3. Update deallocation information
2. Allocate task to respective VM
1. Update allocation infomation
D. Delay
1. Delay
trigger

Fig. 2. Flow diagram of the behavioral control structure for LBA_HB.

A. Workflow submission: (The input is a group of tasks)

A.1. When a new workflow arrives; it is submitted to the preprocessor.

A.2. Then, the preprocessor computes the attributes that are the number of tasks and the instruction length of each task for all ready tasks in the current workflow.

A.3. After that, the task attributes information is stamped along with the task.

B. Task in: (The input is an individual task)

B.1. Ready tasks for execution send request to the VM_load balancer.

B.2. The VM_load balancer inserts the ready tasks into the waiting queue.

B.3. The order of tasks is based on First Come, First Served (FCFS). Then, the VM_load balancer gets the first task in the waiting queue.

B.4. In order to decide which task should be assigned to which VM, collecting information after last allocate and de-allocate is needed. This process is similar to which honey bee should visit which food source that is based on whether honey is available at a flower patch or not. Information perceived consists of two types. First: Threshold information which measures the availability of hosts based on the variation of hosts. The same availability check is implemented in VM level. Second: Priority information which contains information about the current load. Load indicator is considered as processing time in host level and is considered as the number of tasks in VM level.

B.5. Host limit level: It limits the allocation of requests to overloaded hosts since removed task from waiting queue has to find available host that can be allocated to. It has two possibilities based on threshold information; either it finds list of available hosts then checks priority information to choose the host with the least load or it may not find any available host then it is delayed until any host become available.

B.6. If the task finds list of available hosts, then the first task is removed from the waiting queue.

B.7. A control flag is sent to get another task from waiting queue.

B.8. Host priority level: There may be more than one available host which can accept this task. Thus, the task has to find the most suitable host based on priority information. It should consider the available host with minimum load.

In this level, the task is allocated to the specified host(i) based on priority information of hosts. It has minimum processing time as shown in Eq. (7). The minimum processing time indicates the least load.

$$\text{Host}(i) \leftarrow \min\left(PT_{host}(1), PT_{host}(2), .., PT_{host}(n)\right) \quad (7)$$

where i changes from 1 to n.

B.9. VM limited level: Finding available host means that there are one or more available VMs. Then, the rule of this level is to determine the list of available VMs in a specified host.

B.10. VM priority level: There may be more than one available VM in a specified host which can accept the specified task. Thus the task has to find the most suitable VM based on priority information. The priority information in this level is considered as the number of tasks handled by VMs as shown in Eq. (8).

$$\text{VM}(j) \leftarrow \min\left(\text{Count\_}REQ_{VM}(1), \text{Count\_}REQ_{VM}(2),\ldots, \text{Count\_}REQ_{VM}(m)\right) \quad (8)$$

where j changes from 1 to m

C. Task out: After choosing suitable VM.

C.1. the task updates allocated information i.e.; how many tasks are being processed by VM, current processing time of the host and VM and check if the status of host and VM becomes overloaded.

VM(j) becomes overloaded depending on the variation of processing time, i.e. this variation value makes clear indication of load balance between this VM(j) and other VMs when the variation of this VM processing time from average processing time of all VMs is more than or equal to a threshold value as explained as:

$$SND_{vm}(j) \geq \alpha \quad (9)$$

When all VMs in a specified host become overloaded, this host is considered overloaded host.

C.2. The task is allocated to the respective VM found

C.3. The task update de-allocated information i.e; how many tasks are being processed, current processing time of the host and VM and check the availability of VM and host.

State of VM(j) becomes available when the variation of this VM processing time from average processing time of all VMs is still less than a threshold value (α). Available host which has at least one available VM.

C.4. Inform the remaining tasks about the host and VM status similar to the waggle dance performed by the honey bees to inform other honey bees in the bee hive. 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 hosts and VMs.

C.5. Send control flag to tasks to exist from delay when allocated task has finished execution.

C.6. The task will finish execution.

D. Delay: A task does not find available host.

D.1. It goes for delay until receives a control flag from task which has finished execution. Then the task starts from earlier point which has to check the information perceived. The delayed task is served first before the tasks in the waiting queue. Each task after the first round has to go to Step B.4 and then through all the remaining steps. It benefits from the knowledge collected by "bees" for selecting more suitable host and VM to allocate. In addition, in the case of all hosts become overloaded the tasks have to enter a delay until they receive information from "bees" about any host becomes available.

The Pseudo code shows the main processes of the LBA_HB:

Pseudo code of LBA_HB

```
Input:   List_state_host , List_state_VM
Output: VM(j)
// return the number (i) of host which have minimum processing time
// i is the number of specified host
1:  i ← -1
2:  minPT← Integer.MAX_VALUE
3:  For each host(i) in List_state_host
4:    If host(i) available then
5:      If (PT_host(i) < minPT) then
6:          minPT = PT_host(i)
7:          i ← number of the current host
8:      End if
9:    End if
10: End for
// return the number (j) of VM which has minimum count of requests
// j is the number of specified VM
11: j ← -1
12: mincount← Integer.MAX_VALUE
13: For each VM(j) in List_VM_host(i)
14:  If VM available then
15:   If (Count_REQ_VM(j)< mincount) then
16:        mincount = Count_REQ_VM(j)
17:        j ← number of the current VM
18:   End if
19:  End if
20: End for
21: if (j=-1)     then
22: append coming task in waiting queue until one VM become available.
23: else
24:  Allocate the task to VM(i).
25: Update allocated information i.e.; how many tasks are being processed, current processing time of the host and
    VM and check the availability of VM and host
26:  De-allocate the task from this VM after end of the task execution.
27: Update allocated information i.e.; how many tasks are being processed, current processing time of the host and
    VM and check the availability of VM and host.
28: end if
```

4. Simulation Results

In this section, the performance of the proposed algorithm was analyzed based on the results of simulation done using CloudSim [9].

4.1 LBA_HB Implementation

CloudSim is used to model and simulate task scheduling in the large-scale cloud computing. The virtual nodes and computing resources were modeled to evaluate the efficiency of the LBA_HB. The experiments were implemented with 10 Data centers, 50 VMs, and 100-1000 tasks under the simulation platform. The length of the task is from 1000 Million Instructions (MI) to 20000 MI. Processing speed, available memory space, and bandwidth determine the allowable load of each VM. The parameters setting of the simulator are shown in **Table 2** .

Table 2. Simulator parameters

Type	Parameters	Value
Task (Cloudlet)	Length of task(Executable instruction length in bytes)	1000-20000
	Total number of tasks	100-1000
VM	Number of VMs	50
	Processor speed	500-2000 MIPS
	Available memory space in a single VM	256-2048 Mb
	Bandwidth	500-1000
	Cloudlet Scheduler	Time shared
	Number of Processor Elements (PEs) requirement	1-4
Data Center	Number of Data Centers	10
	Number of Hosts	2-6
	VmScheduler	Time shared

4.2 The Effect of Parameters

The effects of parameters were studied in order to adapt the simulation results. These parameters are threshold value, executable instruction length and average number of requests per user per hour.

Fig. 3 shows standard deviation of load which expresses the load balance between VMs. The threshold value changes from 0.0 to 1.0. It can be seen that the best value of threshold that corresponds to the least deviation is 0.1. It is because threshold value represents standard normal deviate of each VM processing time and therefore decreasing threshold value leads to decreasing standard deviation of VM load. However, at threshold equal to zero the deviation of load is increased since its mean that the effect of threshold condition is inactive in this case. Therefore, the value 0.1 is taken under consideration when simulating the proposed algorithm.

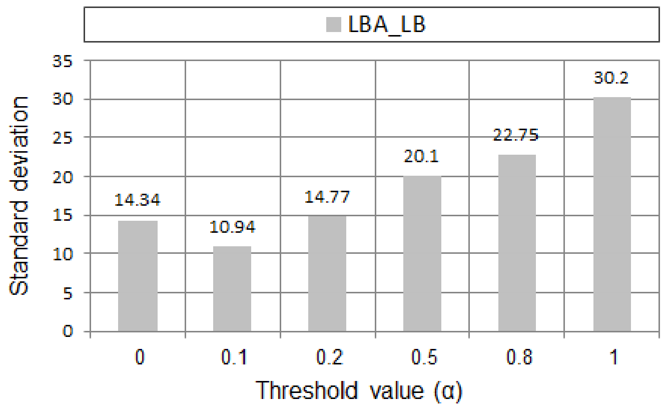

Fig. 3. Standard deviation of LBA_HB with different values of threshold.

Fig. 4 presents average response time of LBA_HB while executable instruction length for each task changes from 2000 to 20000 byte at different number of tasks. The average response time of tasks expresses the amount of time taken between submission of a request and the first response that produced by a task in seconds. It is observed that, the average response time starts at about 6 s at all situations, but this value is increased slightly to reach 18 s when the executable instruction length for task is equal to 2000 byte, however it is highly increased to reach 100 s when the executable instruction length is equal to 20000 byte. This is due to increasing both the number of tasks and the instruction length affects the system load. Then, the response time is also increased.

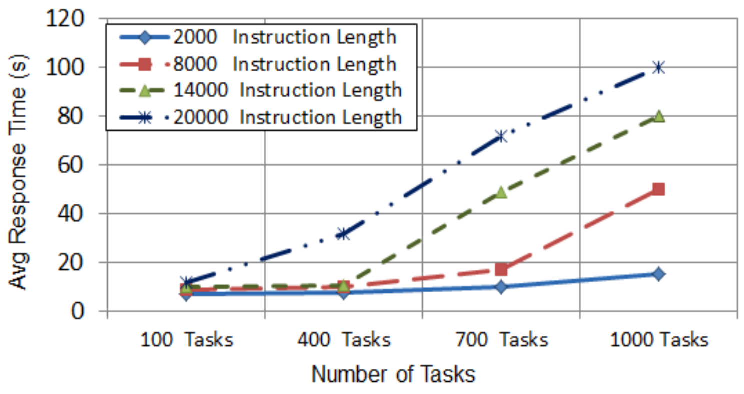

Fig. 4. Average response time of LBA_HB versus number of tasks at different values of instruction length.

Fig. 5 introduces average response time of LBA_HB versus executable instruction length for each user while number of tasks changes from 100 to 1000. It is observed that, the average response time increases slightly at low average number of tasks; however this increase in the response time becomes very fast at high average number of tasks. When the number of tasks is equal to 100 tasks, the increase of the overall average response time in LBA_HB is not reasonable with different values of instruction length. However, the increase of the instruction length is greatly increases the response time at number of requests equal to 1000. This is due to when increasing executable instruction length of tasks, the system load is increased. Therefore, it is important to consider availability and load of each VM when allocating tasks.

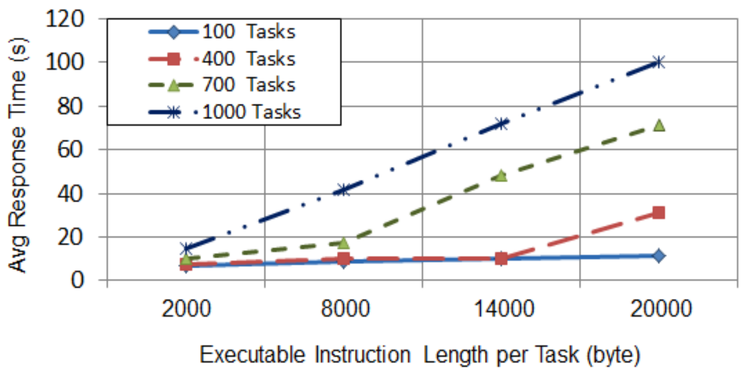

Fig. 5. Average response time of LBA_HB versus executable instruction length at different average number of requests.

4.3 Comparison with Conventional Algorithms

In this section, the results of simulation of the proposed LBA_HB were compared with two conventional static and dynamic algorithms; RR [5] and Modified throttled [6] algorithms; respectively. From the previous section, we denoted that the executable instruction length and the number of tasks are highly effect the overall performance of the system. Then, the default value of the number of tasks is chosen high to show the effect of the proposed algorithm. It is equal to 1000, while the instruction lengths of these tasks are arbitrary changes from 2000 to 20000 that lead to variation of load between different VMs in the system. The threshold value is considered 0.1 since this value guarantees the smallest standard deviation.

Fig. 6 shows the average response time of LBA_HB, RR, and Modified throttled algorithms versus number of tasks. The number of tasks changes from 100 to 1000. The instruction lengths of these tasks are arbitrary changes from 2000 to 20000. It is shown that, LBA_HB saves up to 50% of the average response time over the other algorithms. The average response time of the LBA_HB reaches about 60 s when the number of tasks is 1000 as opposed to 120 s at the other algorithms. This is because the LBA_HB considers least load, availability of VMs, and load variation of each VM when assigning tasks to VMs.

Fig. 7 shows the average response time of LBA_HB, RR, and Modified throttled algorithms versus executable instruction length for each task. The instruction length changes from 2000 to 20000 while the number of tasks is equal to 1000. In this experiment, the tasks have the same length at each step. It is observed that, the overall average response time of each VM in LBA_HB is better than RR and Modified throttled algorithms by about more than 50% improvement. That is due to that LBA_HA avoids underutilization and over utilization of VMs.

In **Fig. 8**, the comparison of average execution time of tasks of LBA_HB, RR, and Modified throttled algorithms with different number of tasks is presented. As shown in the figure, with all values of the number of tasks, LBA_HB takes time less than RR and Modified throttled algorithm. The execution time of LBA_HB is about 8 s as opposed to about 12 s at the other algorithms. This result is due to the efficient use of resources in the proposed algorithm.

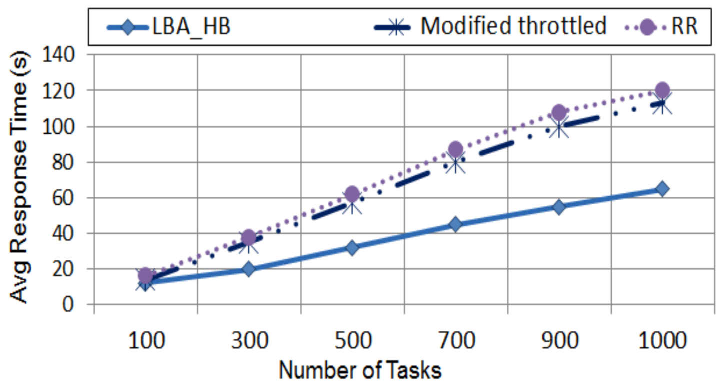

Fig. 6. Average response time versus number of tasks.

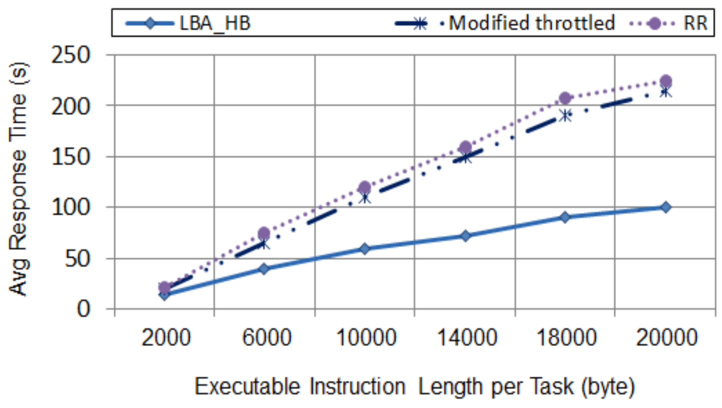

Fig. 7. Average response time with executable instruction length for each task.

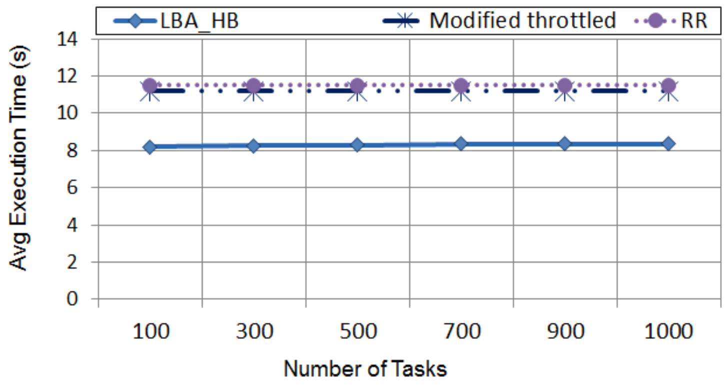

Fig. 8. Average execution time versus number of tasks.

4.4 Comparison with SI Algorithms

SI based schedulers deal with the optimization of one or more balancing metrics such as makespan, degree of imbalance (DI), and standard deviation. Therefore, it is important to compare the proposed LBA_HB with the existing SI algorithms in terms of these metrics. Since most of the existing SI algorithms do not take into consideration the effect of response time and execution time, the previous section presents the effect of these metrics at the proposed algorithm as opposed to the conventional scheduling algorithms only. In this section, LBA_HB was compared with ACO algorithm [11] which is based on ant colony and two honey bee algorithms, ABC algorithm [14] and HBB-LB algorithm [18]. In addition, the two conventional algorithms, RR and Modified throttled are considered in this comparison.

DI measures the imbalance among VMs, which is defined as follows：

$$\mathrm{DI} = \frac{PT_{\max\_vm} - PT_{\min\_vm}}{PT_{Avg\_vm}} \tag{10}$$

The proposed LBA_HB aims to minimize DI. The consideration of DI during the allocation helps to avoid unbalanced workload of VMs.

Fig. 9 shows the comparison of DI between LBA_HB, HBB-LB, ABC, ACO, RR, and Modified throttled algorithms versus different number of tasks. It is clear from the figure that, the proposed LBA_HB is highly preserves the DI since LBA_HB limits allocation of requests to VM when the variation of this VM processing time from average processing time of all VMs becomes more than or equal to a threshold that helps to maintain this DI. The DI in HBB-LB algorithm is greater than the proposed algorithm since this algorithm makes the balance when the overall system becomes unbalanced which increases the imbalance degree.

In **Fig. 10**, the performance evaluation is compared in terms of the average makespan with different number of tasks. Makespan can be defined as the overall task completion time. It is seen that, with the increase of the number of tasks, LBA_HB takes an overall time that is less than the other algorithms. This is due to that LBA_HB assigns tasks to VMs according to least load, availability, and load variation of each VM.

Fig. 11 shows the comparison of standard deviation between LBA_HB, HBB-LB, ACO, ABC, RR, and Modified throttled algorithms with different number of tasks. The standard normal deviation of each VM processing time is the variation of the processing time of this VM from average processing time of all VMs. It is clear from the figure that, the standard deviation of the proposed LBA_HB is not reasonable when compared to the other algorithms since it adapts a threshold value which preserves this deviation.

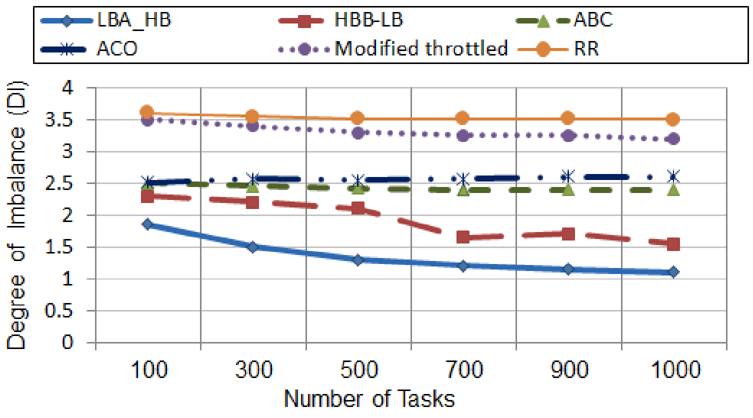

Fig. 9. DI versus number of tasks.

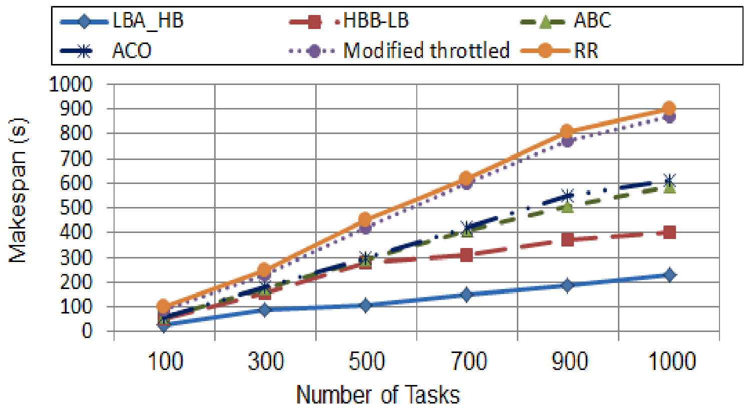

Fig. 10. Average makespan versus number of tasks.

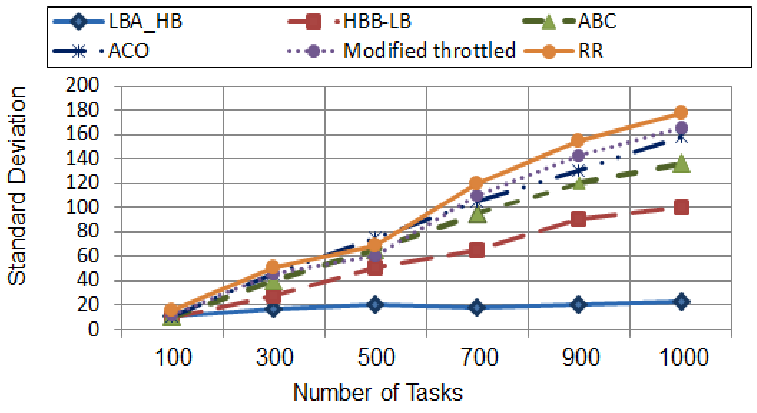

Fig. 11. Standard deviation versus number of tasks.

5. Conclusion

In this paper, a load balancing algorithm in cloud computing environment based on behavior of honey bee foraging strategy is proposed. The proposed LBA_HB aims to minimize overall response time and data center processing time since it distributes workload between different VMs with considering availability and load of each VM. It limits allocation of requests to VM when the variation of this VM processing time from average processing time of all VMs becomes more than or equal to a predefined threshold. Simulation results show that the proposed algorithm improves the average response time and execution time over the well-known algorithms; RR and Modified throttled. In addition, it preserves the deviation and balancing better than the existing SI algorithms; ACO, and ABC. Although, the migration process is not efficient in the proposed LBA_HB since it checks the variation value of VM during task allocation, the migration can be implemented in the case of serving a group of dependent tasks. This enhancement can be implemented in Future work.

References

[1] P. T. Endo, M. Rodrigues, G. E. Gonçalves, J. Kelner, D. H. Sadok, and C. Curescu, "High availability in clouds: systematic review and research challenges," *Journal of Cloud Computing: Advances, Systems and Applications*, pp. 5-16, 2016. Article (CrossRef Link).

[2] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," *Future Generation Computer System*, vol.70, pp.138–147, May 2017. Article (CrossRef Link).

[3] S. Aslam, and M. A. Shah, "Load balancing algorithms in cloud computing: A survey of modern techniques," *National Software Engineering Conference (NSEC)*, Pakistan, December 2015. Article (CrossRef Link).

[4] W. Saber, R. Rizk, W. Moussa, and A. Ghonem, "LBSR: Load balance over slow resources," in *Proc. of International Conference on Computer Applications & Technology (ICCAT)*, Cairo, Egypt, January 28-29, 2017.

[5] P. Samal, and M. Pranati, "Analysis of variants in round robin algorithms for load balancing in cloud computing," *International Journal of Computer Science and Information Technologies*, vol. 4, no. 3, pp. 416-419, 2013. Article (CrossRef Link).

[6] S. G. Domanal, and G. R. M Reddy, "Load balancing in cloud computing using modified throttled algorithm," in *Proc. of International Conference on Cloud Computing in Emerging Markets (CCEM)*, Bangalore, India, October 2013. Article (CrossRef Link).

[7] M. Gamal, R. Rizk, H. Mahdi, and B. Elhady, "Bio-inspired load balancing algorithm in cloud computing," in *Proc. of The International conference on Advanced Intelligent systems and Informatics (AISI)*, Cairo, Egypt, pp. 579-589, September 2017. Article (CrossRef Link).

[8] H. de Vries, and J. C. Biesmeijer, "Modelling collective foraging by means of individual behavior rules in honey-bees," *Behavioral Ecology and Sociobiology, Springer-Verlag*, vol. 44, no.2, pp. 109 – 124, 1998. Article (CrossRef Link).

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, January 2011. Article (CrossRef Link).

[10] E. Pacini, C. Mateos, and C. G. Garino, "Distributed job scheduling based on Swarm Intelligence: A survey," *Computers & Electrical Engineering*, vol. 40, no. 1, pp 252-269, January 2014. Article (CrossRef Link).

[11] M. Tawfeek, A. El-Sisi, A. Keshk, and F. Torkey, "Cloud task scheduling based on ant colony optimization," *The International Arab Journal of Information Technology*, vol. 12, no. 2, pp. 129-137, 2015.

[12] K. Nishant, P. Sharma, V. Krishna, C. Gupta, et al, "Load balancing of nodes in cloud using ant colony optimization," in *Proc. of 14th International Conference on Computer Modelling and Simulation (UKSim)*, Cambridge, March 2012. Article (CrossRef Link).

[13] S. Dam, G Mandal, K. Dasgupta and P. Dutta, "An ant colony based load balancing strategy in cloud computing," *Advanced Computing, Networking and Informatics*, vol. 2, Smart Innovation, Systems and Technologies, Springer, vol. 28, pp. 403-413, 2014. Article (CrossRef Link).

[14] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," *Technical Report TR06,* Computer Engineering Department, Erciyes University, Turkey, 2005.

[15] K. R. Babu, A. A. Joy, and P. Samuel, "Load balancing of tasks in cloud computing environment based on bee colony algorithm," in *Proc. of Fifth International Conference on Advances in Computing and Communications (ICACC)*, Kochi, September 2015. Article (CrossRef Link).

[16] Y. S. Sheeja, and S. Jayalekshmi,"Cost effective load balancing based on honey bee behavior in cloud environment," in *Proc. of First International Conference on Computational Systems and Communications (ICCSC)*, Trivandrum, December 2014. Article (CrossRef Link).

[17] K. R. Babu, and P. Samuel, "Enhanced bee colony algorithm for efficient load balancing and scheduling in cloud," *Innovations in Bio-Inspired Computing and Applications, Advances in Intelligent Systems and Computing, Springer*, vol. 424, pp. 67-78, December 2015. Article (CrossRef Link).

[18] D. Babu L. D., and P. Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Applied Soft Computing, Elsevier*, vol. 13, no. 5, pp. 2292-2303, May 2013. Article (CrossRef Link).

Walaa Hashem received B.Sc. degree in Computer and Control Engineering from Suez Canal University in 2006. Her research interests are in the area of cloud computing.

Heba Nashaat is an assistant professor in the Electrical Engineering Department, Port Said University, Egypt. She received her B.Sc. and M.Sc. degrees in Computer and Control Engineering, Suez Canal University, in 2001 and 2006, respectively. Her Ph.D. degree in computer and control engineering is received from Port Said University in 2011. Her research interests are in the area of computer networking, including mobile networks, handovers, and cloud computing.

Rawya Rizk is professor and head of the Electrical Engineering Department, Port Said University, Egypt. She received her B.Sc., M.Sc. and Ph.D. in Computer and Control Engineering from Suez Canal University in 1991, 1996 and 2001, respectively. Her research interests are in computer networking, including mobile networking, wireless networks, sensor networks, ad hoc networks, QoS, routing, traffic and congestion control, handovers and cloud computing. She is the Chief Information Officer (CIO), Port Said University.