International Journal of Engineering, Management, Humanities and Social Sciences Paradigms (Volume 31, Special Issue of January 2019)
ISSN (Online): 2347-601X and Website: www.ijemhs.com

Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment

Mr.Gandhi Rath[1], Dr. Amaresh Sahu[2]*

*[1]*Assistant Professor,Dept. Of Computer Science and Engineering, NIT , BBSR*
[2]Associate Professor,Dept. Of Computer Science and Engineering, NIT , BBSR

gandhirath@thenalanda.com, amareshsahoo@thenalanda.com*

A B S T R A C T

Load balancing of tasks on the cloud environment is an important aspect of distributing resources from a data centre. Due to the dynamic computing through the internet; cloud computing agonizes from over- loading of requests. Load balancing has to be carried out in such a manner that all virtual machines (VM) should have balanced load to achieve optimal utilization of its capabilities. This paper proposes a novel methodology of dynamic balancing of load among the virtual machines using hybridization of modified Particle swarm optimization (MPSO) and improved Q-learning algorithm named as QMPSO. The hybridization process is carried out to adjust the velocity of the MPSO through the *gbest* and *pbest* based on the best action generated through the improved Q-learning. The aim of hybridization is to enhance the performance of the machine by balancing the load among the VMs, maximize the throughput of VMs and maintain the balance between priorities of tasks by optimizing the waiting time of tasks. The robustness of the algorithm has been validated by comparing the results of the QMPSO obtained from the simulation process with the existing load balancing and scheduling algorithm. The comparison of the simulation and real platform result shows our proposed algorithm is outperforming its competitor.

1. Introduction

Load balancing in Cloud computing is one of the most challenging and useful research for distributing the tasks among the virtual machines at the Data centers. Cloud computing is the concept of on-demand resource sharing through the internet. Cloud consists of thousands of interlinked computers in a multifarious manner, where all files and applications are hosted. Cloud computing integrates the distributed and parallel computing strategy to offer sharing of resources such as software, hardware, information and files as per demand and request of other devices or computer on the cloud. This concept offers ‘‘pay as you need” model in the distributed network. In this strategy, the customer does not require purchasing any computational platforms or software to perform a task and only require the connectivity of the internet to use the resources by paying the money for the duration it has been used. This process reduces the cost of purchasing each software package which are not required full-time and cloud computing provides the facility of dynamic use of the resources. VMs are the processing units in the cloud which compute and share resources as and when required dynamically during execution of the task. In the cloud network, a large number of VMs are connected keeping the resources in pre-emptive and non- pre-emptive manner, as a result resources are not distributed equally and some VMs do not get a chance to acquire the resources. When a task is submitted in the cloud, VMs should execute the task in a faster manner to reduce its run time complexity and in this context; all the VMs should run in parallel manner. There arises the need of scheduling the assigned tasks and completing the execution with in available resources. When multiple tasks are assigned to one or more VMs, then they run concurrently to complete the assigned tasks. When the scheduler is scheduling the tasks for VMs, it should make sure that all tasks are not loaded in one VM only keeping other VMs completely free or underutilized. Hence, it is the responsibility of the scheduler that all the customer tasks should be equally balanced among all VMs in the cloud. To avoid the problem of load balancing among all VMs, it requires an intelligent load balancing algorithm to improve the response time of execution of assigned tasks by safeguarding the maximum utilization of offered resources. Many researchers have been discussing about load balancing strategy in heterogeneous and homogeneous environments and load balancing strategy such as (1) static load balancing and (2) dynamic load balancing. The main aim of the load balancing strategy is to enhance the run time of tasks using the available resources whose capacity of tasks fluctuates during run time in irregular manner. Static load balancing methods will be concentrated properly when there is a low fluctuation of load in the VMs. Hence, the static load balancing algorithm will not properly work as the loads vary unpredictably during run time. Dynamic load balancing is more advantageous than static load balancing where loads will be varying at run time and need to consider the information associated with load information and maintenance. Due to the rapid growth of the network and necessity of resources during the run time, the dynamic techniques are highly essential and fruitful in balancing load among the heterogeneous resources. Our proposed hybrid meta-heuristic algorithm is a dynamic strategy for balancing the load as well as setting the priority of tasks in the waiting queue of VMs. When the multiple tasks have been assigned to a particular VM and other VMs are free in the cloud network, in that situation, the tasks should be removed from a heavily loaded VMs and assigned to the under loaded VMs. Hence, the multiple tasks can be distributed among all VMs with mixed priorities, as a result, the waiting time of the task will be reduced and throughput of VMs will be increase and load balancing will be carried out at VMs. The system model of load balancing is shown in Fig. 1, where user requests the task to be executed in the host. The data centre controller is in charge of the task management and assigns it to the load balancer. The load balancer runs the proposed algorithm to select and allocate the task to VM Managers. VM manager verifies the active VMs, the number of resources required by the tasks and availability of resources with host, if available VMs are not sufficient, then it creates new VMs as per the demand of the tasks. In this manner, the load balancing will be carried out based on the fitness values of the VMs. Each host has a finite number of VMs.

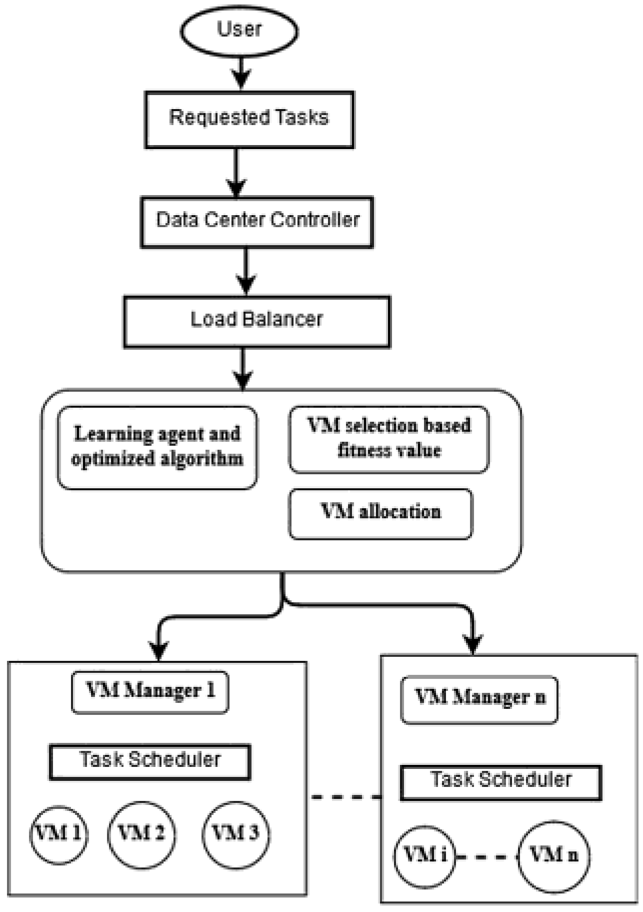


Fig. 1. System model for Load balancing.

This paper aims to achieve a long term load balancing of cloud data centers and provide efficient performance of external services. Usually, the data center is situated far away from the end-users. Distributed servers are the parts of a cloud environment which is available throughout the different internet hosting applications. In order to provide better Quality of Service (QoS) and efficient performance of external services, efficient scheduling and load balancing among the nodes in the cloud environment is required. An efficient load balancing mechanism attempts to speed up the execution time of task requested by users and also shrinks system imbalance and provides fair response time to the users. Better load balancing and scheduling mechanism evades heavy loaded and under loaded situation in data centers. When some VMs are overloaded with numerous tasks, these tasks are migrated to the under loaded VMs in the same data centers to provide better QoS metrics such as efficient response time, resource utilization, scalability and improve the migration time of the tasks. The frequent VM migrations also affect the performance of the cloud ecosystem. Hence, the hybrid meta-heuristic algorithm has been proposed to resolve this kind of dynamic problem. In this work, the objective function has been formulated based on taking a weighted sum of the difference between load on each host and average load on the cloud, total energy consumption and the number of tasks submitted to the different processing units. Each time the fitness value of each VM is calculated through the proposed algorithm (QMPSO) and the task is assigned or migrated to that VM whose fitness value is less than the threshold value.

The main emphasis of this work may be summarized as follows: (1) load balancing for independent task in cloud computing has been studied and above problem has been formulated as a multi-objective constraints based optimization problem; (2) Three objective function has been formulated based on the constraints; first one, the difference of load between each host and average load on Cloud network; second one, in terms of the total energy consumption and third one, in terms of number of task submitted to the different number of processing unit in the cloud network. The overall fitness function is formulated by taking weighted sum of each fitness function; (3) novel method to the solution of load balancing in cloud network problem using QMPSO is suggested in this article; (4) the proposed algorithm has been investigated for load balancing in cloud environment and result obtained through the proposed algorithm are compared with other optimization algorithms such as MPSO and Q-Learning; (5) the performance of the algorithm has been validated through the simulation result and real scenario.

This paper contributed to improve the classical Q-learning algorithm for improving the load balancing in the cloud environment by integrating improved Q-learning with MPSO to improve the convergence rate and performance metrics for load balancing. In this paper, we have enhanced the classical Q-learning value-based MPSO algorithm to determine the load of each VM and balance it through the minimization of the fitness function. In this proposed scheme, the improved classical Q-learning stores the Q-value of the best action of the state and thus saves the storage space, which is used to decide the Pbest and gbest of the improved PSO in each iteration, and adjust the velocity of the MPSO. Finally, the validation of the algorithm is studied in the simulation and real scenarios.

The remaining of the paper is outlined as follows. A review of the state of arts for load balancing is provided in section 2. Problem definition and formulation of the fitness function is explained in section 3. Implementation of a hybrid meta-heuristic algorithm for solving load balancing in cloud computing is presented in Section 4. Experimental result and performance analysis are briefly described in section 5 and the validation of the algorithm has been verified by comparing the results of QMPSO with the existing algorithm in this section. Conclusions and future directions are listed in section 6.

2. Review of state of arts

Load balancing is to assign the input task among all the VMs uniformly. The main objective of the load balancing is to remove the overloaded tasks from one or more VMs and assigned it to others under loaded VMs. This process improves the efficiency and throughput of the machine. Many researchers have resolved the load balancing problem through heuristic and meta-heuristic approaches. Load balancing with security has been addressed in a distributed network (Ezumalai et al., 2010). In this work, three methodologies have been proposed to resolve the load balancing and security in a distributed network. First, it provides architecture for the mobile agent to wander all the nodes in a distributed network. Second, it offers architecture to rearrange the load among the peers to provide better performance and third, it provides security in the network. A Hierarchical load balancing scheme has been proposed to resolve the distribution of the jobs in the Grid computing environment (Malarvizhi and RhymendUthariaraj, 2009). Current node information has been taken care of while distributing the input task dynamically among the VMs and the benefits of the algorithm are to reduce the average response time for Grid application. The efficiency of the algorithm verified through the comparison with Minimal Completion Time and Perfect Information on Arrival. Honey bee behaviour based load balancing has been resolved in non-pre-emptive independent tasks on VMs (Krishna, 2013). The proposed algorithm resolves load balancing across the virtual machines for maximizing the throughput, reducing the waiting time of tasks by considering the priorities of the task and finally, the comparison has been carried out with other algorithms such as weighted Round Robin, FIFO and Dynamic Load Balancing to present the efficiency and robustness of the algorithm in terms of throughput and reducing the response time of VMs. Dynamic load balancing has been resolved in a distributed virtual environment through heat diffusion (Deng and Lau, 2014). It proposes two dynamic load balancing methods, first, it uses the local and global load balancing in the distributed virtual environment through heat diffusion and second it investigated two performance factors such as load balancing factor and convergence threshold. Improved version of particle swarm optimization has been proposed for exercising load balancing in the cloud network (Zhu et al., 2016). This strategy has improved the execution speed of tasks and efficiency. Tabu search algorithm has been projected for resource management in the cloud network (Alam et al., 2014; Larumbe and Sanso, 2013). Dynamic optimized resource allocation management algorithm has been proposed based on the three factors that are deadline constraint, cost constraint and optimum solution (Alam et al., 2014). Tabu search algorithm has been used to resolve resource allocation through prioritization and task grouping. Tabu search algorithm has been designed to optimize the locations of cloud data centers and software components such as information routing and network link capabilities (Larumbe and Sanso, 2013). Makespan and maximum utilization of resource have been reduced through a simple scheduling algorithm in the grid computing environment (Alharbi and Rabigh, 2012). It proposes a load balancing algorithm in which load is equally distributed and minimizes the flow time to complete tasks. Fuzzy logic has been exercised for efficient load balancing and reduces the cost and energy in Geo-Distributed multiple data Centers (Toosi and Buyya, 2015). It shows the optimal offline geographical load balancing through the fuzzy logic inference system by mapping input data non-linearly such as recent utilization of renewable power, availability of electric price and energy consumption to output for redirection of the requests by the data center. Load rebalancing has been carried out for distributed file systems in the cloud network and this strategy also optimizes the network traffic by maximizing the bandwidth of network (Hsiao et al., 2013). Fully distributed load balancing rebalancing algorithm has been presented to resolve load imbalance and finally, the proposed algorithm has been compared with the existing centralized approach. An online algorithm with Lyapunov optimization theory has been projected for load scheduling and eco-aware power management for cloud data centers (Deng et al., 2016). The objective of the algorithm is to minimize the time-average eco-aware power cost of cloud data centers while ensuring the quality-of-experience (QoE) constraint of user requests. Honey bee algorithm has been exercised to reduce the makespan and assign the resource to the task to improve the throughput in the cloud environment (Vasudevan et al., 2016). It is a dynamic algorithm that is used to discover the difference in nature between dependence and independence tasks and reduce the makespan for both the tasks by considering the priorities of tasks. The load of the server has been balanced through self-adaptive Randomized Optimization (Papadopoulos et al., 2016). Energy utilization has been reduced during the assignment of resources to the task in the cloud network (Moganarangan et al., 2016). Load prediction and demand of resources in the future has been explained through Enhanced Exponentially Weighted Moving Average (Lavanya and Vaithiyanathan, 2015). Soft computing techniques have been proposed to resolve the dynamic load balancing in the cloud computing environment (Mondal and Choudhury, 2015). A load balancing strategy for cloud computing has been carried out through a genetic algorithm (Dasgupta et al., 2013). Resource allocation has been carried out in homogeneous and heterogeneous cloud environment (Mishra, 2018). The author has also explained the makespan and energy utilization during allocation. A static load balancing algorithm has been proposed to solve load balancing (Wei et al., 2011; Di et al., 2012; Chen et al., 2010; Song et al., 2014). This static algorithm uses the static information to balance the load without affecting the load of the cluster node and it has the poor adaptive capability. Bayes algorithm has been implemented for long term load balancing (Zhao et al., 2016). Improved weighted round-robin algorithm has been projected to resolve load balancing for non-pre-emptive dependent tasks (Devi and RhymendUthariaraj, 2016). The performance of the load balancing has been analysed through the different load balancing algorithm and the result has been compared in terms of throughput and speed (Kanakala and Reddy, 2015). Many scheduling algorithms have been developed for the Map Reduce environment which contributes to load balancing in cloud network (Manjaly, 2013; Patel, 2015; Selvi and Aruna, 2016). Adaptive task allocation scheduler has been proposed to improve the performance of Map Reduce in a heterogeneous cloud network (Yang and Chen, 2015; Bok et al., 2016). The deadline minimizing scheduler has been proposed to minimize the deadline of the job whose deadline has been collapsed during the processing of large data such as video and image in Map Reduce frameworks (Hwang and Kim, 2012). An autonomous agent based load balancing algorithm has been presented to balance load through VMs using three agents such as Channel agent, load agent and migration agent (Singha et al., 2015).

3. Problem definition and formulation of objective function

Cloud computing consists of set of VMs and each VM are responsible for scheduling and balancing the load by allocating VMs to servers during utilization of load balance over all servers. Consider $M = \{m_1; m_2; m_3 ::: m_m\}$ be the set of '*m*' number of virtual machines in the cloud network and each machine has its own resources. A request of VM can be represented as a d-dimensional vector and each dimension presents the loads of every resource. In this work, four types of resources have been considered such as disk, memory, CPU and bandwidth. The model of the system is considered in this work as S number of homogeneous servers and one is act as a central server which is responsible for posting the request coming from VM. All the request can be accepted by all the servers and generate equivalent VMs comprising with central server. The server can only create the VM only if it has sufficient memory which is a limitation of the system model. There is 'n' number of independent tasks being executed in 'm' number of virtual machines. Therefore, it is requiring a better scheduler to map 'n' task to 'm' VMs so that load will be balanced. The main objective of the load balancing is to maximize the balance of the utilization of resources and accept as many requests. The parameter used for single user as U (RPHPU, RS, RCPU, RM, C) where, RPHPU indicated the number of online user request coming from every time stamp on average in a user group, RS refers to request size of each user in user group, RCPU refers to amount of CPU required to execute the request, RM indicates the amount of memory needed to execute the request and C refers to the number of request sent in every minute. The establishment of load balancing in cloud computing environment is required to design a fitness function. On basis of group strategy, one host contains many VMs and each VM is able to allocate resources to many groups and each group is presented as G (RPHPU, RS, RCPU, RM, C) and after executing the group tasks by VM_i, the memory load of VM_i is calculated as follows:

$$LM^i = RM^i + \frac{Vmp_i}{Vm_i} \times 100\% \tag{1}$$

In the above equation the term Vmp_i represents the percentage of memory available in the ith VM and Vm_i represents the percentage of total memory available in the ith VM. The ratio appearing in Eq. (1) is a dimension less constant. RM^i is the remaining of memory before executing tasks. The group request next parameter is amount of CPU is required to execute the task in VM_i and it is calculated as follows:

$$LC^i = RC^i + \frac{Vmc_i}{Vm_i} \times 100\% \tag{2}$$

where Vmc_i represents the percentage of CPU available in the ith VM and Vm_i represents the percentage of total CPU available in the ith VM. The ratio appearing in Eq. (1) is a dimension less constant. RC^i is the remaining of CPU before executing. Base on the memory load and CPU load presented in Eq. (1) and Eq. (2), the overall load on VM_i can be calculated as follows:

$$OL^i = k_1 \times LM^i + k_2 \times LC^i \tag{3}$$

where k_1 and k_2 are the weight factor and $k_1 + k_2 = 1$, hence, the overall load on the host j is calculated as follows:

$$LH^j = \sum_{i=0}^{m_j} OL^{ji} = \sum_{i=0}^{m_j} k_1 \times LM^{ji} + k_2 \times LC^{ji} \tag{4}$$

where OL^{ji} represents the overall load on ith VM of jth host, LM^{ji} represents the load of memory on ith VM of jth host, LC^{ji} represents the load of CPU on ith VM of jth host and m_j represents the *m* number of VMs activated in jth host physical machine. The average load on all physical machines in cloud is calculated as follows:

$$AL = \frac{\sum_{j=0}^{p} LH^j}{p} = \frac{\sum_{j=0}^{p} \sum_{i=0}^{m_j} OL^{ji}}{p} = \frac{\sum_{j=0}^{p} \sum_{i=0}^{m_j} k_1 \times LM^{ji} + k_2 \times LC^{ji}}{p} \tag{5}$$

where *p* is the number of host in the cloud network. The difference of load between each host and average load on Cloud network is $|LH^j - AL|$. The first fitness function is defined as

$$F_1 = \sum_{j=0}^{p} |LH^j - AL| \tag{6}$$

The second fitness function is defined on basis of the energy consumption of active or idle VMs. Each VMs has two states when it is created through host or physical machine, one is active state and other is idle state. Energy consumption (EC) and makespan (MS) are two important parameters in cloud network for evaluating the performance. The execution time for completing the task in each VMs are different and makespan defined the execution time of VMs in the system. The maximum the execution time poor in load balance and minimum of the execution time provides the better load balancing. The execution time of the jth VM is T_j and it is calculated based on the decision variable U_{ij}.

$$U_{ij} = \begin{cases} 1; & \text{if } T_i \text{ is assigned to } VM_j \\ 0 & \text{if } T_i \text{ is not assigned to } VM_j \end{cases} \tag{7}$$

$$T_j = \sum_{i=1}^{n} U_{ij} \times TC_{ij} \tag{8}$$

where TC_{ij} is the ith$(1 \le i \le n)$ task completion time in jth$(1 \le j \le m)$ VM and TC_{ij} is calculated as follows:

$$TC_{ij} = \frac{L_i}{PS_j} \tag{9}$$

where L_i is the length of the ith task and length of the task is defined in terms of number of instructions (Millions of instruction) and PS_j is the processing time of jth VM in the cloud. The makespan (MS) is the maximum value of the execution time of all virtual machines and mathematically defined as follows:

$$MS = Max\ T_j\ ;\ 1 \le j \le m \tag{10}$$

The total energy consumed is defined as the sum of the energy consumed in the idle states and active states. Consider a_j joules/Millions of instruction consumed by jth VM in the active state and b_j joules/Millions of instruction consumed by jth VM in the idle state. $MS - T_j$ amount of time will be remain idle by jth VM. The mathematical representation of second fitness function in terms of the total energy consumption is defined as follows:

$$F_2 = EC = \sum_{j=1} \left(T_j \times a_j + (MS - T_j)\, b_j\right) \times PS_j \tag{11}$$

The third fitness function is defined in terms of number of task submitted to the different number of processing unit in the cloud network and it is defined as follows:

$$F_3 = w_1 \times \frac{NIPT_i}{MNIPS} + w_2 \times L_i \tag{12}$$

where $NIPT_i$ represents number of instructions for ith task, this is count of instruction in the task determined by the processor.*MNIPS* represents maximum (total) number of instruction executed byeach processor per second,. The delay cost L_i is an estimated penalty for i^{th} task, Which Cloud service provider needs to pay to customer in the event of job finishing actual time being more than the deadline advertised by the provider. w_1 and w_2 are the weight factor and its value is consider as 0.7 and 0.3 respectively. The objective or fitness function is formed by taking the weighted average of each individual fitness function which will be minimized through the proposed algorithm and it is mathematically presented as follows:

$$F = k_1 \times F_1 + k_2 \times F_2 + k_3 \times F_3 \quad (13)$$

where k_1; k_2 and k_3 are weights of the difference of load between each host and average load on Cloud network, total energy consumption and number of task submitted to the different number of processing unit in the cloud network respectively. These weights are adjusted in the simulation and preeminent values found $k_1 = 1$; $k_2 = 0.5$ and $k_3 = 0.5$. So, the optimized load balance is obtained by minimizing the fitness function in Eq. (13) with the assigned weights of each criterion.

4. Hybrid meta-heuristic algorithm for load balancing

Q-learning algorithm

Q-learning is a one of reinforcement learning algorithm in the area of machine learning which allows the agent to learn in the environment and perform an action by transition of state to get a reward or penalty based on the feedback received from the environment. The main aim of the agent is to use the control strategy to select appropriate action from set of possible action from a specified state to the destination through the transition process of state. When the processing enriched through repetitive steps, then the problem is known as a Markov decision process (MDP) with unknown probabilities of transition. Consider there are set of states $S = \{s_1; s_2; \ldots; s_n\}$ in the environment and each states have set of actions $A = \{a_1; a_2; \ldots; a_m\}$. An agent selects an action $a_t \in A$ at time t in the state $s_t \in S$ to transit to the next state $s_{t+1} \in S$ through the transition process and acquire an immediate reward r_{t+1} from the environment. It is necessary to select appropriate action that maximizes the Q-value of each state which is the main objective of finding an optimal policy in the cloud network. The Q-value function basically depends on what is the selection criterion of action in the particular state. Consider the agent in the state s_t and select an action a_t which is expected to move best next state and maximize the total expected reward in the environment, then Q-value is calculated as follows:

$$Q(s_t; a_t) = (1 - \alpha)Q(s_t; a_t) + \alpha \quad r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}; a_{t+1}) \quad (14)$$

where α is the learning rate, $\gamma(0 < \gamma < 1)$ is the discount factor and effect on the successive state by the previous action and r_t is the immediate penalty or reward received by executing the action a_t on the state s_t. The Q-value is achieved by generating a Q-table that stores the possible states and their Q-value and the appropriate actions. The Q-learning algorithm recursively attempts to generate the optimal state and system cost based on their experienced. The following greedy strategy has been used in Q-learning algorithm which yield the Q-value to convergence over time.

$$Q_{t+1}(s_t; a_t) = Q_t(s_t; a_t) + \alpha \Delta Q_t(s_t; a_t) \quad (15)$$

where α is the learning rate and it is calculated as follows:

$\alpha = 1/1 +$ total of times visited to state s_t

$$\Delta Q_t(s_t; a_t) = r_t + \alpha \max_{a'_t}\{Q_t(\delta(s_t; a_t); a'_t)\} - Q_t(s_t; a_t) \quad (16)$$

where d is the transition function. The possible action taken in the cloud computing related to load balance are resource withdrawing from VM, allocating resource in VM and swap resource between VMs. In the cloud computing the reward is calculated in the terms of cost function and the cost function is calculated in the terms of resources requested, resource allocated and rejected of request and collision of requests. The reward r_t at time t is calculated as follows:

$$r_t = r_1 n_1(s_t) + r_2 n_2(s_t) + r_3 n_3(s_t) \quad (17)$$

where $n_1(s_t)$ successive allocation of resources and it is calculated as follows:

$$n_1(s_t) = \frac{\text{resource grants} - \text{total collisions}}{\text{resource grants}} \quad (18)$$

$n_2(s_t)$ is rate of resource request collision occurs

$$n_2(s_t) = \frac{\text{resouce request collision}}{\text{resource grants}} \quad (19)$$

$n_3(s_t)$ represents the rate of resource request rejected

$$n_3(s_t) = \frac{\text{resouce request rejected}}{\text{resource requested}} \quad (20)$$

The classical Q-learning algorithm suffers with the following drawbacks such as (1) The learning agent used for updating Q-value of each state is based on the action executed and there is no model to decide which action should the agent decide so that optimal Q-value will generate. It may decide the action randomly and get an optimal Q-value; (2) Every time agent fetches the memory to get the appropriate action through the optimal Q-value of the state; (3) the convergence speed and learning rate becomes very slow. Classical Q-learning takes huge computation to calculate the Q-value for all possible actions in a particular state and takes large space to store its Q-value for all actions as a result of which its convergence rate is slow. In classical Q-learning, if there are 'n' states and 'm' action per state, then the Q-table will be of $(m \times n)$ dimension. Due to the above flaws the classical Q-learning, the modified Q-learning has been proposed to overcome the flaws of the classical Q-learning. In modified Q-learning the previous actions can be affected by the feedback of the successive states. If the action taken by the current state is false, then the preceding actions should be penalty otherwise the preceding actions should be rewarded. In the modified form classical Q-learning stores the Q-value of the best action of the state and thus saves the storage space. In the modified Q-learning, for each state 2 storages are required, respectively for storing Q-value and storing value of the lock variable of a particular state. Thus for n number of states, we require a Q-table of $(2 \times n)$ dimension. The saving in memory in the present context with respect to classical Q thus is given by $mn - 2n = n(m - 2)$..Modified Q learning algorithm is summarized below:

```
     Algorithm 1: QUpdate(S)
1.:  Initialize Q-values:
2.   For i = 1 to n          // n is the number of states
3.   For j = 1 to p        // p is number of action in each states
4.   $Q(s_i,a_j) = 0$
5.   End for
6.   End for
7.   Select an action $a_i$ from $A=\{a_1, a_2,\ldots a_m\}$ and execute it
     and go to next state $s_{t+1}$
8.   Calculate the learning factor $\alpha$
9.   Calculate the reward $r_t$ using Eq. (17)
10.  Calculate error signal $\Delta Q_t(s_t, a_t)$ using Eq. (16)
11.  Update $Q_{t+1}(s_t, a_t)$ using Eq. (15)
12.  Repeat the process for the new state until it converges
```

Modified PSO (MPSO)

The classical PSO is bio-inspired population based optimization algorithm. In PSO, each member in the population is called particle and set of particles in the population is called swarm and it updates the position and velocity of each particles based on the following principles.

$$V_i(t+1) = V_i(t) + C_1 \cdot \phi_1 \cdot (pbest_i(t) - x_i(t)) + C_2 \cdot \phi_2 \cdot (gbest(t) - x_i(t)) \quad (21)$$

$$x_i(t+1) = x_i(t) + V_i(t+1) \quad (22)$$

where $V_i(t)$ is the velocity of the $i^{th}(1 \leq i \leq N)$ agent at iteration t, $x_i(t)$ is the position of i^{th} agent at iteration t, $pbest_i(t)$ is the personal best position of the i^{th} agent at iteration t, $gbest(t)$ is the best position of the agent in the swarm at iteration t, C_1 and $C_2$are the social and cognitive factor whose values are positive, ϕ_1 and ϕ_2 are the random numbers in the range of 0 and 1.

The classical PSO needs to improvement for better convergence and maintain the good balance between exploration and exploitation. The improvement of PSO has been presented below.

$$V_i(t+1) = \left[\sum_{t=1}^{t_{\max}} cc(i,t)\, pastV_i(t)\right] + C_1 \cdot \phi_1 \cdot (pbest_i(t) - x_i(t)) + C_2 \cdot \phi_2 \cdot (gbest(t) - x_i(t)) \quad (23)$$

where $cc(i,t)$ is the correlation coefficient which is updated based on the fitness value of the i^{th} agent at iteration t as per the following expression:

$$cc(i,t) = \begin{cases} cc_{high} & \text{if } F(t+1) > F(t) \\ cc_{low} & \text{otherwise} \end{cases} \quad (24)$$

$cc(i,t)$ equal to cc_{high}, if the current fitness value is better than the previous fitness value of the i^{th} agent, otherwise its value will be cc_{low}. The value of cc_{high} and cc_{low} for the present problem are considered as 0.8 and 0.35 respectively. This helps the agent to move in the direction in which it finds the better fitness value. The correlation coefficient is multiplied with the previous velocity in each iteration to represent the weight of the previous velocity in the update equation of the present velocity. In each iteration the value of cc is corresponding to a particular previous velocity is multiplied by correlation coefficient depletion factor (CCDF) to reduce its effect in the successive velocity and CCDF value is consider as 0.5 in this problem.$pastV_i(t)$is the past velocity of the i^{th} agent at iteration t. Particle leads to a local optimum when the social component value C_2 is more as compared to the cognitive component C_1 and comparatively great value of cognitive components outcomes to roam the particles around the search space. The quality of the solution is enhanced by adapting cognitive and social coefficient term in such a manner that the cognitive component is decreased and social component is increased as iteration proceeds. The variation of the coefficients is offered (for tth generation) by means of the following Eq. (25) and Eq. (26).

$$C_1 = C_{1i} - \frac{C_{1i} - C_{1f}}{t_{\max}}\, t \quad (25)$$

$$C_2 = C_{2i} + \frac{C_{2f} - C_{2i}}{t_{\max}}\, t \quad (26)$$

where C_{1i}; C_{1f} ; C_{2i} and C_{2f} are initial and final values of cognitive and social component acceleration factors respectively and $t_{\max}$ is the maximum number of allowable iterations. To avoid the stagnation problem, it is required to replace the worst fitness value agent by new agents. Consider S number of agent are performing worst fitness and replace it by initializing the new agents with the surviving agents in the population. Each new agent 'Naget' in the swarm acts as a parent and past velocities and average position of parents are used to initializing the new agent in the population. This is possible by mutation of randomly selected agent and followed by the following mutation equation:

$$x_{ij} = x_{ij} + u_3 \frac{ub_j - lb_j}{2} \quad (27)$$

where u_3 is the random number in the range of $[-1,1]$. ub and lb are the upper and lower boundary and its value depends upon the number of task between 100 and 2000 and i is the particle in the swarm. In modified PSO, the updated particle position in Eq. (22) is defined by adding the velocity of particle obtained from Eq. 23 with the position of the particle obtained through mutation using Eq. (27).

```
Procedure QMPSO for load balancing
Input: $N$ is population size, $R = N/10$ and $Prt = N/8$
1.  Initialize Swarm size of VMs and control parameters
2.  While stopping condition not reached
3.  For $i = 1$ to $N$
4.  $oldfit_i = fit_i$
5.      Evaluate $fit_i$
6.  End for
7.  For $t = 2$ to $t_{\max}$
8.  $cc(i,t) = cc(i,t-1) \times CCDF$
9.  $pastV_i(t) = pastV_i(t-1)$
10. If $oldfit_i > fit_i$ then
11. $cc(i,1) = cc_{low}$
12. else
13. $cc(i,1) = cc_{high}$
14. $pastV_i(t) = V_i(t)$
15. End if
16. Call Qupdate($pbest$)
17. Call Qupdate($gbest$)
18. Call Qupdate($x_i$)
19. If$Q(x_i)$is better than $Q(pbest_i)$
20. $pbest_i = x_i$
21. If $Q(x_i)$is better than $Q(gbest)$
22. $gbest = x_i$
23. End for
24. Sort the swarm in decreasing order of their fitness value
25. For $i = N - R + 1$ to $N$
26. For $t = 2$ to $t_{\max}$
27. $cc(i,t) = cc_{low}$
28. $pastV_i(t) = 0$
29. $cc(i,t) = cc_{high}$
30. $x_i = \frac{1}{prt}\sum_{k=1}^{prt} x_{((i-N)+(R-1)\times prt)+k}$
31. $pastV_i(t) = \frac{1}{prt}\sum_{k=1}^{prt} pastV_{((i-N)+(R-1)\times prt)+k}(t)$
32. Mutate $x_i$ using Eq. (27)
33. End for
34. End for
35. For $i = 1$ to $N$
36. Update $V_i(t)$ and $x_i$ using Eq. (23) and Eq. (22)
    respectively
37. End while
```

The complexity of the algorithm is computed in three different phases such as making a group of the task into each VMs, scheduling for each individual group and allocating VMs for the task. When n number of the task is submitted to the data center, the task scheduler collects the information from the task manager and resource manager to create the appropriate groups. The submitted n tasks are split into p number of groups of tasks based on their fitness value. Hence, the complexity for making into different groups is O (n). The scheduling for each individual group, QMPSO takes O(n^2) to schedule n submitted tasks of each p number of groups. So QMPSO repeats the process p times to schedule n task of each group is $p \times O(n^2) \approx O(n^2)$. QMPSO balance the loads overall m number of resources. So, the algorithm migrates the task from the heaviest loaded VM to lighted loaded VM. Therefore, the complexity of balancing load is $m \times O(n) \approx O(n)$ The overall complexity of the algorithm QMPSO is $O(n + n^2 + n) \approx O(n^2)$.

Table 1
Simulation environment.

Type	Number	Parameters	Value
Data center	1	Arch	x86
		OS	Linux
		VM Monitor	Xen
		Cost	3.0
		Cost Per Memory	0.05
		Cost per Storage	0.001
VM	10 to 100	Processor speed	9726MIPS
		Memory	0.5 GB
		Bandwidth	1 GB/s
		Image size	10 GB
		Mumber of PEs	1
		VM Monitor	Xen
Host	20	MIPS	177,730
		Storage	4.0 TB
		VM Monitor	Xen
		RAM	16.0 GB
		Bandwidth	15 GB/s
		Cores	6

5. Experimental results and performance analysis

The performance of the proposed algorithm has been analysed based on the results of the simulation. The cloud computing experiment has been carried out through the CloudSim3.0.3 simulator and this simulator runs on the machine with the configuration of Intel core i7 processor, 8 GB RAM, 3.4 GHz CPU and Window 7 platform. The simulation environment of the experiments has been presented in Table 1. The performance of the algorithm has been analysed in the form of the number of tasks migrated, the response time of tasks, delayed in all tasks, idle time of tasks, makespan before load balance and after load balance through modified PSO and Improve Q-learning. Two types of scenarios have been considered for validation of the algorithm. In the first scenario, fixing the number of VMs (1 0 0) which are created from 10 real processors and varying the number of tasks from 100 to 2000 in the interval of 50 and the second scenario, fixing the number tasks (1000) and varying the number of VMs from 10 to 100 in the interval of 50. Based on the above two scenarios the performance of the algorithm has been analysed. The performance of the algorithm has been analysed in terms of Makespan, energy utilization, standard deviation which measures the effect of load balance, Throughput which measures the effective performance of the external services,

Table 2
Influence of parameters for Load balance in cloud computing.

Weights			Performance Analysis and Convergence rate				
k_1	k_2	k_3	Makespan (In ms)	Throughput (req/ms)	Standard deviation (SD)	Energy utilization (In KJ)	Load Balance (yes/No)
1	0.9	0.9	9671.36	9.18	0.389	312.52	yes
		0.8	9416.33	8.79	0.376	301.39	yes
		0.7	9281.51	8.62	0.366	297.36	yes
		0.6	9072.19	8.41	0.427	291.29	yes
		0.5	8962.91	8.13	0.391	285.71	yes
		0.4	8852.63	7.89	0.362	271.43	yes
	0.8	0.9	9428.17	8.89	0.378	253.72	yes
		0.8	9389.14	8.75	0.367	247.39	yes
		0.7	9161.31	8.39	0.342	233.87	yes
		0.6	9051.29	8.19	0.317	227.31	yes
		0.5	8942.81	7.95	0.298	219.37	yes
		0.4	8828.35	7.67	0.289	213.29	yes
	0.7	0.9	9175.37	8.56	0.359	298.15	yes
		0.8	9132.74	8.41	0.321	291.67	yes
		0.7	9194.89	8.67	0.339	292.45	yes
		0.6	9013.54	8.14	0.312	287.40	yes
		0.5	8583.71	8.04	0.298	279.19	yes
		0.4	8239.61	7.43	0.279	268.41	yes
	0.6	0.9	9087.41	8.28	0.313	267.19	yes
		0.8	8794.51	8.07	0.298	263.81	yes
		0.7	8663.95	7.89	0.287	259.31	yes
		0.6	8532.13	7.45	0.259	253.30	yes
		0.5	8374.30	7.24	0.238	247.52	yes
		04	8139.37	7.67	0.265	251.97	yes
	0.5	0.9	8894.51	7.39	0.275	243.19	yes
		0.8	8467.92	7.18	0.174	239.37	yes
		0.7	8135.29	6.74	0.096	228.36	yes
		0.6	7954.64	5.83	0.082	218.42	yes
		0.5	7791.32	5.46	0.067	173.87	yes
		0.4	8534.78	6.34	0.276	204.56	yes
	0.4	0.9	8429.49	7.98	0.373	256.72	yes
		0.8	8321.82	7.39	0.314	248.30	yes
		0.7	8148.29	7.12	0.274	229.58	yes
		0.6	8093.21	6.87	0.238	212.29	yes
		0.5	7930.62	6.45	0.208	189.43	yes

0.4	7832.31	6.23	0.136	182.64	yes

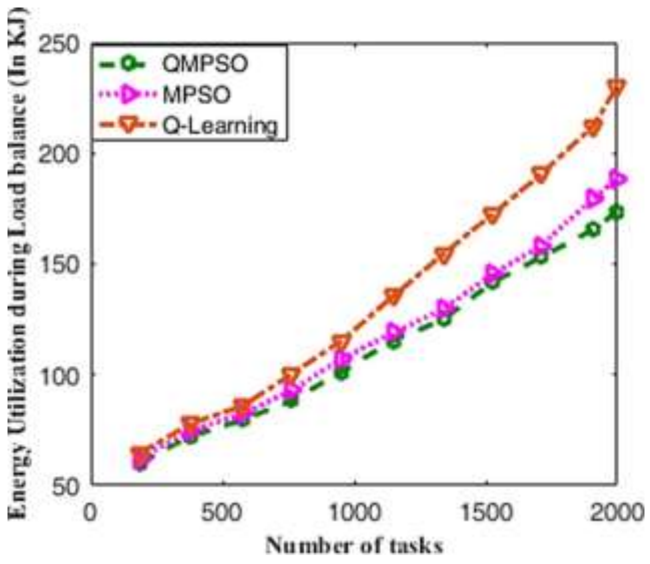


Fig. 2. Number of tasks with Energy utilization for fixed number of VMs.

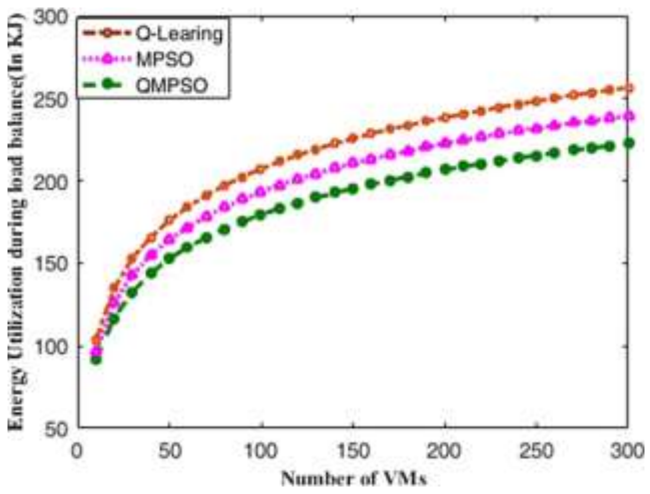
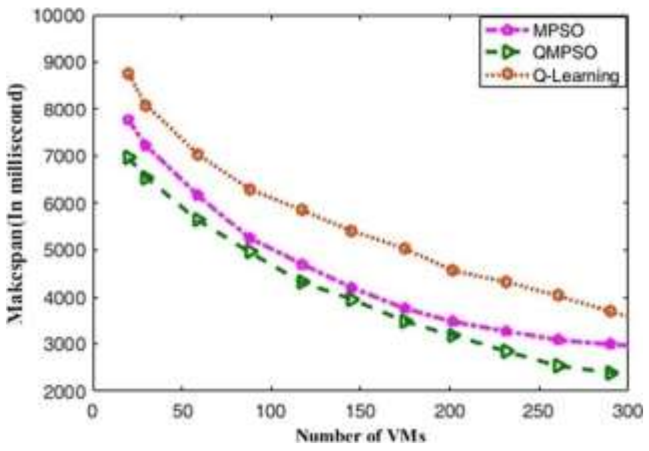


Fig. 3. Number of VMs with Energy utilization for fixed number of Tasks.

Tasks migrated measures the performance of cloud and Quality of services (QoS), Degree of imbalance, Idle time which measures the waiting time of tasks and Processing time of tasks based on the computing power of VMS. The experiment has been conducted through the fitness function which has been presented in Eq. (13) and the weight of the fitness value has been decided through the exercise of different values which has been presented in Table 2. The optimal value found for the weight of the fitness function after the exercise of the simulation as k_1 ¼ 1:0; k_2 ¼ 0:5; and k_3 ¼ 0:5. The performance has been found out by the varying the number of tasks and VMs. The energy utilization during load balancing by varying tasks and VMs has been presented in Fig. 2 and Fig. 3 respectively for three different algorithms such as QMPSO, MPSO, and Q-Learning. The conclusion is drawn from Fig. 2 and Fig. 3 that QMPSO takes the least amount of energy during load balance as compared to its competitors. The performance has been exercised in terms of makespan. Makespan evaluates the response time of the user for a particular task and respond time affect the QoS in cloud computing, as a result, the service provider can promise the QoS to the client. Makespan

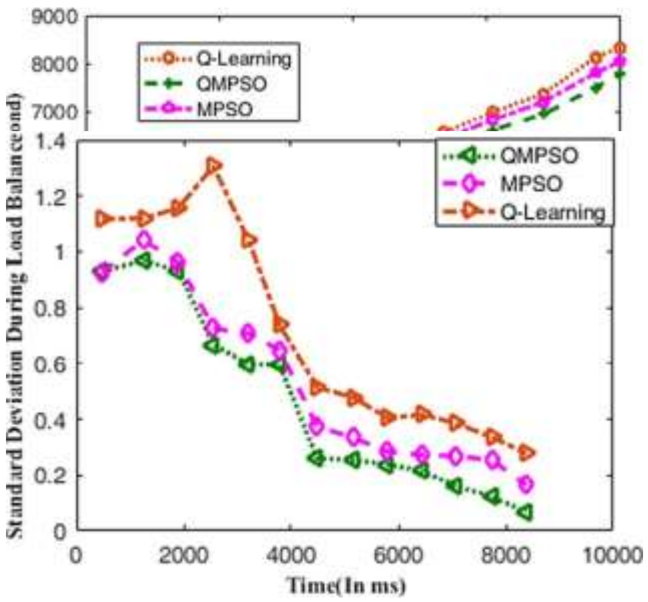


has presented in Fig. 4 and Fig. 5 by varying tasks and VMs respec-

Fig. 4. Number of VMs with Makespan fixed number of Tasks.

Fig. 5. Number of Tasks with Makespan fixed number of VMs.

Fig. 6. Time with standard Deviation for QMPSO, MPSO and Q-Learning.



tively. The conclusion is drawn from Fig. 4 and Fig. 5 that it shrinks into a noteworthy amount in QMPSO as compared to MPSO and Q-Learning. Hence, QMPSO achieved the stability and load balance efficiently through the comparison of Energy utilization and make-span with its counterpart algorithm. The load balance has been evaluated through the standard deviation and standard deviation

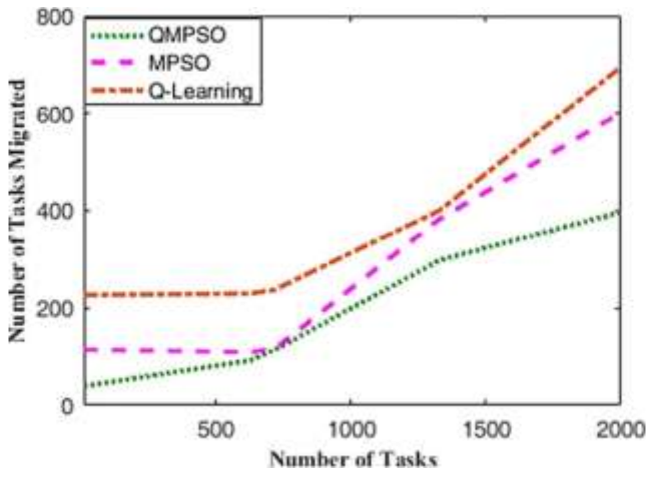


Fig. 7. Number of Tasks Migrated with Number of Tasks for QMPSO, MPSO and Q-Learning.

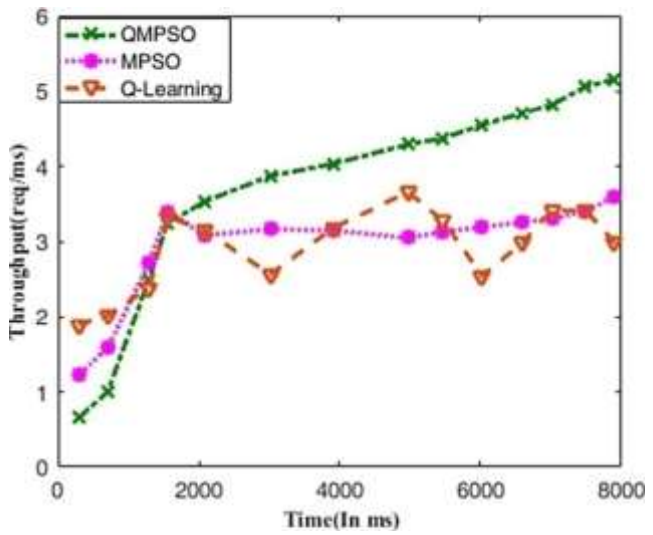


Fig. 8. Time with Throughput for QMPSO, MPSO and Q-Learning.

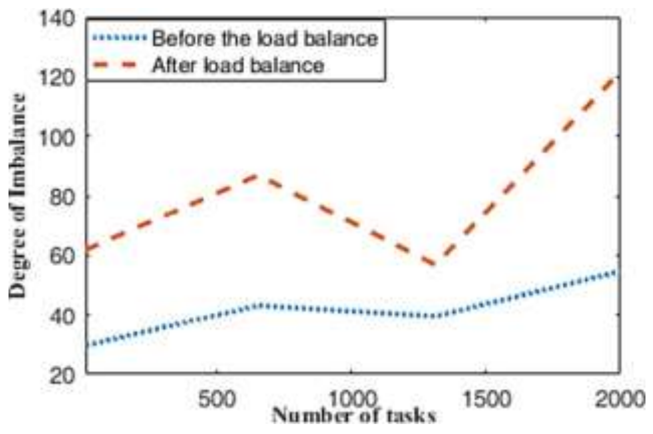


Fig. 9. Number of tasks with Degree of Imbalance before and after load balance.

presents the degree of load balance in the could network. The smaller the value of standard deviation indicated better in balancing of the load. The performance of load balancing has been exercised in terms of standard deviation and it has been presented in

Fig. 6. Fig. 6 shows that initially, the standard deviation value of QMPSO is small as compared to MPSO and Q-Learning and gradu- ally it decreases. At time t = 3700 ms standard deviation value is the same for QMPSO and MPSO. However, when t 3700, it decli- nes the degree of Standard deviation of MPSO is greater than QMPSO. This is because the proposed QMPSO gets optimal resources rapidly permitting the residual computing power. The conclusion is drawn from the analysis that the QMPSO approach has better resource utilization capability and efficient load balanc- ing as compared to its competitors. The performance has been evaluated in terms of task migrated. The task is migrated because of not getting the requested resource on the physical host. Task migrated with the number of tasks has been evaluated and pre- sented in Fig. 7. Fig. 7 shows that the number of tasks migrated for QMPSO is less as compared to MPSO and Q-Learning. The load balance has been evaluated through the throughput, throughput is used to evaluate the performance of the external service such as availability of resources to deal with the requested task, ability of transmitting data and ability of responding requested tasks, etc.the result of the experiment has been presented in Fig. 8 for throughput analysis. QMPSO.

≥

Fig. 8 shows that the performance of external service for Q-learning is better than its counterpart at the initial stages, as time increases the performance of the external service is stable, it is same for three algorithms when t = 1700 and gradually, the

Fig. 10. Number of VMs with Idle Time for all tasks for QMPSO, MPSO and Q-Learning.

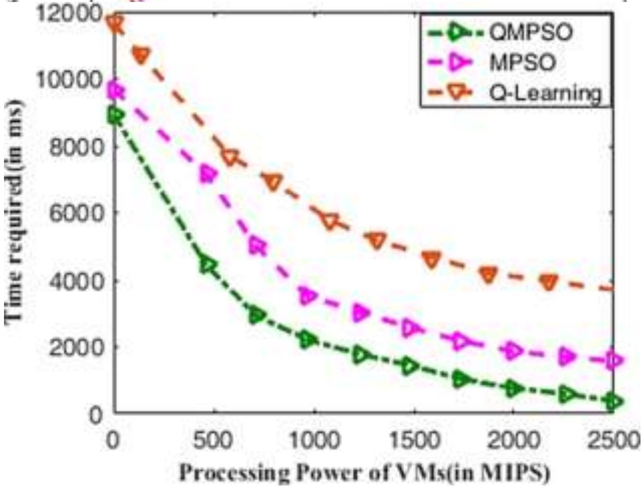


Fig. 11. Processing Power of VMs with Time required for all tasks for QMPSO, MPSO and Q-Learning.

Table 3
Analysis of performance for QMPSO, MPSO and Q-Learning.

Algorithm	Makespan (In ms)		Throughput (req/ms)	Standard deviation (SD)	Energy utilization (In KJ)		Task Migrated
	for fixed VMs (300)	for fixed Tasks (2000)			for fixed VMs	for fixed Tasks	
QMPSO	7791.30	2389.00	5.46	0.067	173.80	222.45	397
MPSO	8051.70	2966.60	3.81	0.165	188.28	239.56	601
Q-Learning	8335.50	3588.00	3.15	0.278	230.21	256.67	695

performance of the external service for QMPSO is better than MPSO and Q-Learning. This is because of the physical hosts chosen in the set by QMPSO has achieved the demand of the requested tasks. The conclusion is drawn from this analysis that QMPSO has acquired better efficiency and stability as compared to its counterpart algorithm. Load balancing reduces the degree of imbalance (the better the load balance less the degree of imbalance). Load balance basically depends on the number of the task requested for accruing resources available in the physical host. Exercise has been carried to present the degree of Imbalance before the execution of the proposed algorithm and after the execution of the proposed algorithm and it has been presented in Fig. 9. The conclusion is drawn from Fig. 9 that the degree of Imbalance is least after exercise of idle time of the tasks increases due to the allocation of more tasks in single high processing power VMs. The waiting time tasks are more if only one task is allowed to execute at the same time other tasks allow us to wait in the ready queue for execution. If the processing power of VMs is more and number VMs are allocated, then the waiting time of all tasks will be less. The Idle time of the tasks and processing time of VMs has been presented in Fig. 10 and Fig. 11 respectively. Fig. 10 shows that idle time for all tasks is less for QMPSO as compared to MPSO and Q-Learning. The processing power of VMs has been presented in Fig. 11. The processing power has been increased from 200MIPS to 2500 MIPS, as a result, the processing time of the tasks has been deduced to 9500 ms to 102 ms. The conclusion is drawn from the analysis that QMPSO effectively and efficiently balances the load in the cloud network.

The comparison has been carried out with its counterpart algorithms and the result has been presented in Table 3. The conclusion shows that the QMPSO is outperformed MPSO and Q-learning in load balancing of tasks in the cloud network.

The robustness of the algorithm has been verified in the real platform. In the real platform, the cloud data center consists of 4 hosts, each host capable of supporting virtualization technology. The detail of the host specification used in the experiment has been presented in Table 5. In this experiment, we have considered 16 VMs for executing the different groups of tasks ranges from 20 to 80 in the 4 hosts. Each VM consists of 9226 MIPS of Processing speed, 512 MB of RAM, 10240 MB of storage space and 1024MIPS of bandwidth. The experiment and simulation have been conducted through 4 hosts and 16 VMs with the specification as provided in Table 4. Performance evaluated through the experiment and simulation has been presented in Table 5. It shows that the proposed QMPSO algorithm outperforms than its competitors and the percentage of error in QMPSO is less as compared with other algorithms.

The effectiveness and robustness of the proposed algorithm QMPSO have been verified through the exiting algorithm IPSO (Saleh et al., 2018). The author (Saleh et al., 2018) proposed task scheduling in the cloud computing environment through an improved version of Particle swarm optimization called IPSO. Improve version of PSO(IPSO) is projected to afford the optimal allocation for a large number of tasks and it is accomplished by splitting the submitted tasks into several batches through a

Table 4
Host technical details.

Host ID	Processing Cores	Speed, MIPS	RAM, MB	Storage, MB	BW, MIPS
1	4	6000	204,800	1,048,576	102,400
2	3	4500	152,400	1,048,576	102,400
3	2	3500	102,400	1,048,576	102,400
4	1	2000	51,200	1,048,576	102,400

Table 5
Comparison of simulation and real platform result.

In Real Platform								In Simulation					% of Err r
VM	TASK	Algorithm	Makespan (in Sec.)	No. of Task Migrated	Throughput (r_{eq}/ms)	Energy utilization	Completion Time	Makespan (in Sec.)	No. of Task Migrated	Throughput (r_{eq}/ms)	Energy utilization	Completion Time	
16	20	QMPSO	2.56	3	1.7	87.20	3.25	2.30	3	1.53	78.4	2.92	10
	30		2.87	5	3.08	100.02	3.95	2.58	5	2.77	89.9	3.55	
	40		3.15	6	4.13	147.2	4.51	2.83	6	3.71	132.4	4.05	
	60		3.47	8	5.54	185.9	5.95	3.12	8	4.98	167.3	5.35	
	80		4.02	10	6.47	236.7	7.02	3.61	10	5.82	213.0	6.31	
	20	MPSO	2.71	5	2.34	90.12	3.45	2.57	5	2.22	85.61	3.2	20
	30		3.9	7	3.5	105.2	4.02	3.70	7	3.32	99.94	3.8	
	40		5.1	9	4.9	135.3	4.91	4.84	9	4.65	128.5	4.66	
	60		7.6	12	6.02	194.6	6.02	7.22	12	5.71	184.7	5.71	
	80		12.6	14	8.30	242.7	7.95	11.9	14	7.8	230.5	7.55	
	20	Q-Learning	4.21	4	2.14	101.2	3.97	4.05	4	2.06	97.45	3.82	27
	30		5.5	6	3.36	119.3	4.82	5.29	6	3.23	114.8	4.64	
	40		6.4	8	4.91	155.6	5.96	6.16	8	4.72	149.8	5.73	
	60		8.1	11	6.25	193.24	7.98	7.8	11	6.01	186.0	7.68	
	80		12.27	15	8.45	235.6	9.03	11.81	15	8.13	226.8	8.67	

Table 6
Comparison of the simulation result of QMPSO with (Saleh et al., 2018) by varying number of task up to 1000 with fixed number VM.

Performance matrices	QMPSO	IPSO
Makespan (In Sec)	5.8	167
Standar deviation	0.9	16
Degree of Inbalance	35	17
Idle time of the tasks (In Sec)	8.2	Not evaluated
Throughput (req/ms)	5.3	Not evaluated
Tasks Migrated	165	Not evaluated
Time required for all tasks (In Sec.)	8.5	Not evaluated
Energy utilization (In KJ)	165	Not evaluated

dynamic strategy. The utilization of resources is based on each formation batches and after getting the sub-optimal solution for each batch, the algorithm combines all sub-optimal solutions of batches into the final allocation map. Lastly, IPSO attempts to balance load based on the final allocation map. The efficiency of the algorithm has been evaluated in the terms of the degree of imbalance, the standard deviation of load and makespan. The comparison of the QMPSO with the existing algorithm IPSO has been presented in Table 6. It is noted from Table 6 that the proposed QMPSO algorithm is outperformed (Saleh et al., 2018).

6. Conclusions and future direction

In this paper, the hybrid meta-heuristic algorithm such as QMPSO has been proposed for load balancing for independent tasks in the cloud computing network. The proposed algorithm balances the load by reassigning the load to the appropriate VMs by considering the fitness value of each VMs. The proposed algorithm also improves the makespan, throughput, energy utilization during load balancing and reduces the waiting time of the tasks effectively as compared to separated algorithms such as MPSO and Q-learning. At last, we have also compared our proposed algorithm with the existing algorithm and found that our proposed algorithm outperforms the existing algorithm. In the future, the load balancing will be carried out among the dependent tasks dynamically.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Ezumalai, R., Aghila, G., Rajalakshmi, R., 2010. Design and architecture for efficient load balancing with security using mobile agents. Int. J. Eng. Technol. (IACSIT) 2 (1), 149–160 [Online].
Malarvizhi, N., RhymendUthariaraj, V., 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.
Krishna, P. Venkata, 2013. Honey bee behavior inspired load balancing of tasks in cloud computing environments. Appl. Soft Comput. 13 (5), 2292–2303.
Deng, Y., Lau, R.W., 2014. Dynamic load balancing in distributed virtual environments using heat diffusion. ACM Trans. Multimedia Comput. Commun. Appl. (TOMM) 10 (2), 16.
Zhu, Y., Zhao, D., Wang, W., and He, H. (2016, January) 'A Novel Load 460 Balancing Algorithm Based on Improved Particle Swarm Optimization in Cloud Computing Environment', In International Conference on Human Centered Computing, Springer, pp. 634–645.
Alam, M.I., Pandey, M., Rautaray, S.S., 2014. A proposal of re-source allocation management for cloud computing. Int. J. Cloud Comput. Services Sci. 3 (2), 79–86.
Alharbi, F., Rabigh, K.S.A., 2012. Simple scheduling algorithm with load balancing for grid computing. Asian Trans. Comput. 2 (2), 8–15.
Toosi, A.N., Buyya, R., 2015, December. A Fuzzy Logic-based Controller for Cost and Energy Efficient Load Balancing in Geo-Distributed Data Centers. In: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pp. 186–194.
Hsiao, H.C., Chung, H.Y., Shen, H., Chao, Y.C., 2013. Load rebalancing for distributed file systems in clouds. IEEE Trans. Parallel Distrib. Syst. 24 (5), 951–962.
Deng, X., Wu, D., Shen, J., He, J., 2016. Eco-aware online power management and load scheduling for green cloud datacenters. IEEE Syst. J. 10 (1), 78–87.
Vasudevan, S.K., Anandaram, S., Menon, A.J., Aravinth, A., 2016. A novel improved honey bee based load balancing technique in cloud computing environment. Asian J. Information Technol. 15 (9), 1425–1430.
Papadopoulos, A.V., Klein, C., Maggio, M., Drango, J., Dellkrantz, M., Hernndez-Rodriguez, F., et al., 2016. Control-based load balancing techniques: analysis and performance evaluation via a randomized optimization approach. Control Eng. Pract. 52, 24–34.
Moganarangan, N., Babukarthik, R.G., Bhuvaneswari, S., Basha, M.S., Dhavachelvan, P., 2016. A novel algorithm for reducing energy consumption in cloud computing environment: Web service computing approach. J. King Saud Univ.-Comput. Information Sci. 28 (1), 55–67.
Lavanya, M., Vaithiyanathan, V., 2015. Load prediction algorithm for dynamic resource allocation. Indian J. Sci. Technol. 8 (35), 1–4.
Mondal, B., Choudhury, A., 2015. Simulated annealing (SA) based load balancing strategy for cloud computing. (IJCSIT) Int. J. Comput. Sci. Information Technol. 6 (4), 3307–3312.
Larumbe, F., Sanso, B., 2013. A tabu search algorithm for the location of data centers and software components in green cloud computing networks. IEEE Trans. Cloud Comput. 1 (1), 22–35.
Dasgupta, K., Mandal, B., Dutta, P., Mandal, J.K., Dam, S., 2013. A genetic algorithm (ga) based load balancing strategy for cloud computing. Procedia Technol. 10, 340–347.
Mishra, Sambit Kumar, Sahoo, Bibhudatta, Parida, PritiParamita, 2018. Load balancing in cloud computing: a big picture. J. King Saud Univ.-Comput. Information Sci.
Wei, Q., Xu, G., Li, Y., Research on cluster and load balance based on Linux virtual server. In: Proc. Inf. Comput. Appl., 2011, pp. 169–176.
Di, Y., Wang, S., Sun, X., A dynamic load balancing model based on negative feedback and exponential smoothing estimation. In: Proc. 8th Int. Conf. Autonomic Auton. Syst., Mar. 2012, pp. 32–37.
Chen, W., Zhang, Y., Xiong, Z., 2010. Research and realization of the load balancing algorithm for heterogeneous cluster with dynamic feedback. J. Chongqing Univ. 33 (2), 2–14.
Song, S., Lv, T., Chen, X., Feb. 2014. Load balancing for future internet: an approach based on game theory. J. Appl. Math. 2014, (2014) 959782.
Zhao, J., Yang, K., Wei, X., Ding, Y., Hu, L., Xu, G., 2016. A heuristic clustering-based task deployment approach for load balancing using Bayes theorem in cloud environment. IEEE Trans. Parallel Distrib. Syst. 27 (2), 305–316.
Devi, D. Chitra, RhymendUthariaraj, V., 2016. Load balancing in cloud computing environment using improved weighted round robin algorithm for nonpreemptive dependent tasks. Scientific World J. 2016.
Kanakala, V.R.T., Reddy, V.K., 2015. Performance analysis of load balancing techniques in cloud computing environment. TELKOMNIKA Indones. J. Electr. Eng. 13 (3), 568–573.
Manjaly, J.S., 2013. Relative study on task schedulers in HadoopMapReduce. Int. J. Adv. Res. Comput. Sci. Softw. Eng. 3 (5).
Patel, H.M., 2015. A comparative analysis of MapReduce scheduling algorithms for Hadoop. Int. J. Innov. Emerg. Res. Eng. 2 (2).
Selvi, R.T., Aruna, R., 2016. Longest approximate time to end scheduling algorithm in Hadoop environment. Int. J. Adv. Res. Manag. Archit. Technol. Eng. 2 (6).
Yang, S.J., Chen, Y.R., 2015. Design adaptive task allocation scheduler to improve MapReduce performance in heterogeneous clouds. J. Netw. Comput. Appl. 57, 61–70.
Bok, K., Hwang, J., Jongtae Lim, J., Kim, Y., Yoo, J., 2016. An efficient MapReduce scheduling scheme for processing large multimedia data. Multimed. Tools Appl., 1–24
Hwang, Eunji, KyongHoon Kim, 2012. Minimizing cost of virtual machines for deadline-constrained mapreduce applications in the cloud. In: Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing. IEEE Computer Society.
Singha, A., Juneja, D., Malhotra, M., 2015. Autonomous Agent Based Load-balancing algorithm in Cloud Computing. International Conference on Advanced ComputingTechnologies and Applications (ICACTA), 45, 832–841.
Saleh, H., Nashaat, H., Saber, W., Harb, H.M., 2018. IPSO task scheduling algorithm for large scale data in cloud computing environment. IEEE Access 7, 5412–5420.