The Journal of Supercomputing (2021) 77:8787–8839
https://doi.org/10.1007/s11227-020-03601-7

CMODLB: an efficient load balancing approach in cloud computing environment

Sarita Negi, et al. ***[full author details at the end of the article]***

Accepted: 24 December 2020 / Published online: 28 January 2021

Abstract
A hybrid of supervised (artificial neural network), unsupervised (clustering) machine learning, and soft computing (interval type 2 fuzzy logic system)-based load balancing algorithm, i.e., clustering-based multiple objective dynamic load balancing technique (CMODLB), is introduced to balance the cloud load in the present work. Initially, our previously introduced artificial neural network-based dynamic load balancing (ANN-LB) technique is implemented to cluster the virtual machines (VMs) into underloaded and overloaded VMs using Bayesian optimization-based enhanced K-means (BOEK-means) algorithm. In the second stage, the user tasks are scheduled for underloading VMs to improve load balance and resource utilization. Scheduling of tasks is supported by multi-objective-based technique of order preference by similarity to ideal solution with particle swarm optimization (TOPSIS-PSO) algorithm using different cloud criteria. To realize load balancing among PMs, the VM manager makes decisions for VM migration. VM migration decision is done based on the suitable conditions, if a PM is overloaded, and if another PM is minimum loaded. The former condition balances load, while the latter condition minimizes energy consumption in PMs. VM migration is achieved through interval type 2 fuzzy logic system (IT2FS) whose decisions are based on multiple significant parameters. Experimental results show that the CMODLB method takes 31.067% and 71.6% less completion time than TaPRA and BSO, respectively. It has maintained 65.54% and 68.26% less MakeSpan than MaxMin and R.R algorithms, respectively. The proposed method has achieved around 75% of resource utilization, which is highest compared to DHCI and CESCC. The use of novel and innovative hybridization of machine learning, multi-objective, and soft computing methods in the proposed algorithm offers optimum scheduling and migration processes to balance PMs and VMs.

Keywords Machine learning · Interval type 2 fuzzy set · Load balancing · ANN · K-mean clustering · VM migration

1 Introduction

The ubiquitous nature of cloud computing has attracted tremendous users in recent years [1]. Cloud computing has the ability to handle an expanded volume of tasks by providing an adaptive online environment. The ability of the cloud is to support numerous users and tasks that provide many advantages. It introduces a new concern, known as load balancing [2]. The involvement of numerous users requires an efficient load balancing mechanism to maintain an unmanaged cloud environment system. Numerous load balancing methods are studied and examined in the previous work [3]. Here, the authors provided a clear view of the performances of different cloud load balancing techniques with their pros and cons. Figure 1a shows a balanced cloud environment in which user tasks are distributed to all virtual machines (VMs) in each physical machine (PM) to maintain load balancing. Similarly, Fig. 1b illustrates an unbalanced cloud in which user tasks are assigned to a particular PM to make it overloaded. The efficient utilization of resources makes a significant and balanced cloud.

Several researchers have shown interest in other clouds-related research such as task scheduling and VM scheduling process to achieve efficient load balancing, high utilization of resources, energy efficiency, improvement in quality of service (*QoS*), etc. [4]. In task scheduling, many optimization algorithms such as particle swarm optimization (PSO), fuzzy logic (FL), and genetic algorithm (GA) are used to achieve optimized solutions. Author [4] has analyzed various task and resource scheduling processes in different layers. Adaptive multi-objective task scheduling (AMTS) strategy was introduced for efficient resource utilization and energy efficiency [5]. The PSO algorithm is adapted for the task scheduling process. Temporal task scheduling algorithm (TTSA) deals with cost minimization problem in a hybrid cloud [6], where the solution for cost minimization problem is derived from hybrid simulated annealing PSO algorithm. VM-based task scheduling proves that

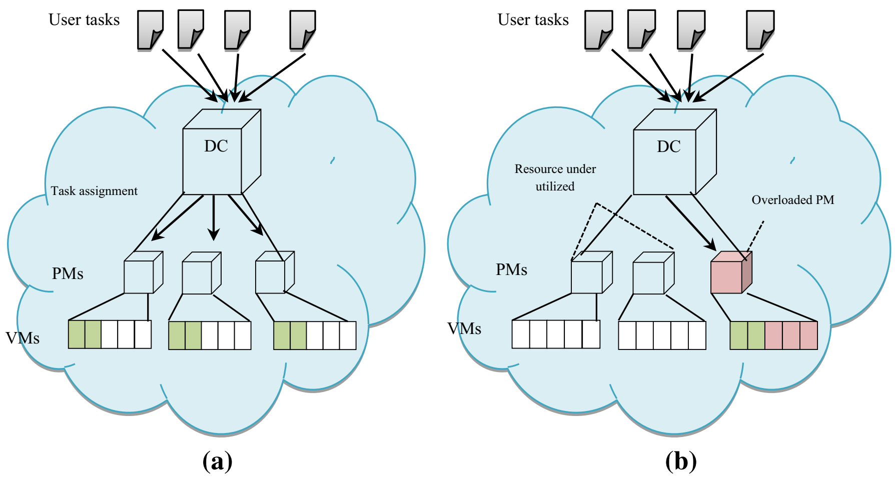


Fig. 1 **a** Balanced cloud, **b** Unbalanced cloud

load balancing and resource utilization can be improved jointly by a proper scheduling process [7]. To achieve this, greedy PSO (GPSO) algorithm in which particle initialization is better than traditional PSO is employed. In order to reduce the large consumption of energy and improve load balancing, resource pre-allocation and task scheduling were employed [8] in the process. Here, scheduling of tasks and allocation of resources are performed by the matching probabilistic and simulated annealing (SA) algorithms (CESCC).

MakeSpan is minimized by performing suitable load adjustment in a multi-data center cloud [9], in which a static load balancing strategy, namely the Multi-Rumen Anti-Grazing algorithm, is presented for load adjustment. Autonomous agent-based load balancing (A2LB) algorithm balances load among VMs in the cloud with the help of three agents as load, channel, and migration [10]. These three agents are incorporated in each data center (DC) to migrate VM from overloaded DC to underloaded DC. Another agent-based load balancing scheme is included with front-end agent (FA), server manager agent (SMA), and virtual machine agent (VMA) [11]. FA is responsible for routing VM requests, while VMA and SMA are responsible for load monitoring in VM and PM, respectively. Load balancing in a dynamic multi-service scenario in centralized hierarchical cloud-based multimedia system is considered as an integer linear programming problem [12]. The problem is solved by an efficient GA with an immigrant scheme. The load balancer is responsible for task migration, whereas load balancing is achieved using efficient task scheduling in DC [13]. Here, the tasks are scheduled by an improved weighted round robin (WRR) algorithm, which considers the current load on VM for scheduling. Dynamic Hadoop cluster on IaaS (DHCI) architecture has involved modules of VM migration, VM management, scheduling, and monitoring to achieve well-behaved load balancing for cloud system environment [14]. Parallel computing entropy is utilized for dynamic VM migration-based data locality scheme. Continuous VM migration is supported by named data networking (NDN) in order to balance load among DCs in the cloud [15]. In NDN, VMs and services are provided with a unique name to provide uninterrupted services during VM migration.

Though the above-discussed works improve load balancing, it is still a primary concern due to the increasing demand of cloud-based environment in every advanced computing field. Thus, it is necessary and demanding to design novel algorithms for load balancing in the cloud system environment. The major strength of the proposed method is that it includes the machine learning-based supervised and unsupervised technique to train the cloud systems to identify the loads on VMs and clusters them into underloaded and overloaded VMs. Once cluster formation is achieved, it would be easier for the cloud system to assign cloud tasks to underloaded VMs. Task assignments are performed using a mathematical multi-criteria-based TOPSIS-PSO method which will reduce the system complexity by introducing relative closeness for cloud performance metrics, execution time, transmission time, and CPU utilization to the PSO method. Further, a fuzzy-based technique is incorporated to identify the destination PM for the migration of overloaded VMs to achieve PM-level load balancing. The use of these soft computing based-techniques gives more realistic results than the previously existing work and enhances the proposed method strength. However, the proposed method does not focus on storage-intensive tasks and storage IOPS/transfer-based VM capacity which makes the work less advanced in a real-time-based cloud environment.

The significant contributions of this paper in cloud computing are summarized as follows:

- Efficient CMODLB load balancing method is the hybridization of artificial neural network-based load balancing (ANN-LB), technique of order preference by similarity to ideal solution with particle swarm optimization (TOPSIS-PSO), and interval type 2 fuzzy logic system (IT2FLS) methods in order to balance load among VMs and PMs in the cloud environment. The objective of CMODLB is to improve multiple objectives such as MakeSpan, resource utilization, completion time, transmission time, energy consumption, and load fairness.
- The proposed CMODLB is initiated by Stage 1 where the ANN-LB technique is applied using Bayesian optimization-based K-means (BOK-means) with ANN algorithm to create clusters of VMs into underloaded VMs and overloaded VMs. Runtime tasks are scheduled to underloaded VM to maintain the load balance.
- In Stage 2, runtime tasks are scheduled by applying the TOPSIS-PSO algorithm, which supports three objectives for scheduling. The multiple objectives taken into account are execution time, transmission time, and CPU utilization.
- In Stage 3, the VM manager monitors the load on PMs and migrates optimal VM from the overloaded PM. VM migration decision is also made if a PM has a minimum load in order to minimize energy consumption.
- In the VM migration process, optimal destination PM is selected by decision making interval type 2 fuzzy logic system (IT2FL), which achieves efficiency in uncertainties. Fuzzy logic assures that the destination PM is underloaded based on significant parameters.

The rest of this paper is prepared as follows: Sect. 2 debates previous related literature on cloud environment-based load balancing. In Sect. 3, existing problems in the previous work are highlighted. Section 4 explains the proposed CMODLB load balancing algorithm working structure, while Sect. 5 examines the experimental evaluation of implemented work. Finally, Sect. 6 gives a glimpse of the contribution of the work and concludes with future scope.

2 Related work

The adaptive cost-based task scheduling (ACTS) method was introduced to minimize the completion time of a task to maximize its *QoS* [16]. Completion time on each VM has been taken as the cost of the data access path, and tasks were given a priority based on a deadline. The minimum cost path was allocated for a low-priority task, which had a minimum deadline. Here, VMs are selected based on completion time only, which has introduced unbalanced load and resource underutilization. Hence, this approach degrades the performance of the cloud when the selection of VMs is made only by completion time.

A two-level task scheduling technique was presented in a multimedia cloud [17]. In this mechanism, first-level scheduling was performed between users to DCs and second-level scheduling was performed among DCs to servers. Both scheduling levels followed the M/M/1 queuing system. However, M/M/1queue supports only a single server and also limits waiting space for tasks. This limitation makes it less efficient.

Li et al. attempted to reduce the heavy consumption of energy in the cloud by minimizing job completion time (JCT) [18]. For this purpose, task placement and resource allocation (TaPRA) and TaPRA-fast algorithm were developed. However, these algorithms are able to solve only a single-task placement problem and not suitable to handle multiple tasks.

Energy-aware dynamic task scheduling (EDTS) algorithm was involved with two algorithms, namely data flow graph critical path (DFGCP) and critical path assignment (CPA) [19]. Here, DFGCP was used to obtain a near-optimal solution, while the CPA algorithm was employed to find an optimal solution. This method fails to provide efficient resource utilization with lower energy consumption.

Load balancing was performed through VM scheduling in the cloud [20]. VM scheduling was carried out by combining two optimization algorithms, resulting in a new meta-heuristic algorithm named ant colony optimization with particle swarm (ACOPS). The workload for new requests was predicted based on historical information in ACOPS. A pre-rejection module was incorporated to minimize scheduling time. Maintaining historical information increases space complexity, and due to the pre-rejection module, user tasks do not experience better *QoS*.

Hybrid particle swarm optimization (HPSO) was applied for load balancing in centralized cloud-based multimedia storage [21]. In HPSO, each particle's weight was computed by multi-kernel support vector machine (MKL-SVM) and fuzzy simple additive weight (FSAW) methods. The involvement of three simultaneous and dynamic algorithms increases both load balancing time and computational complexity.

Honey bee behavior-inspired load balancing (HBB-LB) algorithm was presented to balance load across VMs and minimize the tasks' waiting time [22]. Load balancing was performed by the task transfer process in which tasks from overloaded VMs are migrated to underloaded VMs. Here, load balancing during initial task assignment is not considered and tasks are assigned to random VM, which increases MakeSpan and resource underutilization.

PSO algorithm and bacterial foraging optimization (BFO) algorithm were combined and introduced bacterial swarm optimization (BSO) to perform load balancing in DCs [24]. In a hybrid bacterial algorithm, the global solution over search space was determined by the PSO algorithm, and local search was performed by the BFO algorithm. This algorithm focused on only allocating optimal VM for incoming tasks; however, load on PMs was not considered.

Dragonfly optimization algorithm-based load balancing method was introduced to balance load among VMs [25]. Initially, each VM's load and capacity were calculated, and tasks were assigned to VMs randomly. If the load on VM was greater than the threshold, then optimal underloaded VM for each task was selected. This method increases MakeSpan for user tasks and suffers from high processing time.

Load balancing via optimal task deployment strategy was realized in [26]. Bayes theorem was introduced to find probabilities of physical host for optimal task deployment. Based on the probabilities, physical hosts were clustered to detect optimal PM for user task. In this way, the load across PMs was balanced. However, this method is not suitable to balance load across VMs.

VM migration was performed by using a distributed network of brokers [27]. Migration decision was made based on RAM and CPU state of VMs, and migration was performed by deriving policies by hypervisor. This method increases complexity and does not perform well in terms of resource utilization.

Firefly optimization–energy-aware virtual machine migration (FFO-EVMM) algorithm was presented to attain both load balancing and energy efficiency [28]. FFO-EVMM algorithm also utilizes the concept of artificial bee colony (ABC) algorithm for load monitoring on VMs. If a migration decision was made, then optimal VM for migration and optimal destination for migration were determined by FFO algorithm. This method increases migration time because time constraint is not considered in the migration process.

Energy-aware utility prediction model was introduced for VM consolidation in a cloud [29]. Here, VM consolidation model was applied periodically to optimize VM placement. VM migration was enabled if any resource of VM exceeds total capacity. Consideration of a single parameter for migration increases the number of VM migrations, which results in higher energy consumption.

Thus, the previous works seem to be insufficient to achieve efficient load balancing. The majority of the works focus on either task scheduling or load balancing in a cloud. However, introducing an effective task scheduling and load balancing (among VMs and PMs) method is a key concern in cloud computing. This work introduces an algorithm with the aim to increase load balancing performance by using effective clustering, task scheduling, and VM migration process.

3 Problem definition

An enhanced Min-Min algorithm, which involves two phases, was presented for load balancing in a cloud [30]. In the first phase, tasks were assigned to VMs based on execution time, and in the second phase, tasks were rescheduled to utilize unused resources. Since tasks are assigned to VMs, which provide minimum execution time. In the first phase, rescheduling of tasks increases execution time. Involvement of two scheduling processes increases overall MakeSpan, which degrades its performance.

In multi-objective tasks scheduling algorithm, tasks were given priority based on *QoS*, and VMs were sorted based on MIPS values [31]. Tasks and VMs were assigned as per the task and VM list. In this process, resource utilization is poor since VMs are sorted based on MIPS values. The VM list which is prepared initially is processed with all task set. After assigning a task to VM, the parameters of VM may change. This change is not considered in the presented method. Hence, the scheduling process is not much efficient.

Two-stage strategy for task scheduling includes Bayes classifier for classification of VMs in the first stage and scheduling algorithm in the second stage [32].

In this method, task queue, waiting queue, and ready queue were maintained, and tasks were put in a waiting queue until a suitable VM became idle. Maintenance of multiple queues and databases increases space complexity. This method allows waiting queue tasks to move into the ready queue only if the ready queue is free. Hence, waiting time for tasks in the waiting queue is increased significantly.

The self-adaptive learning PSO (SLPSO) algorithm was included with four update strategies to update the velocity of particles [33]. The best update strategy was detected in each iteration based on execution probability and used in the SLPSO algorithm. In SLPSO, frequent selection of update strategy leads to higher time consumption and higher complexity.

Load balancing mutation using PSO (LBMPSO) algorithm was introduced to schedule tasks and load balancing in which objective function was formulated based on expected execution time, transmission time, and round trip time [34]. All three times were computed for each task on each VM to find the optimal VM. If the number of VMs and tasks are large, then this method presents a large MakeSpan and high complexity. Hence, it motivates to introduce an effective idea to balance cloud loads without reducing execution time and resource utilization.

An efficient novel task scheduling algorithm called technique of order preference by similarity to ideal solution with particle swarm optimization (TOPSIS-PSO) algorithm has been introduced [35]. TOPSIS-PSO solves task scheduling issues using multiple objective-based technique *i.e.,* TOPSIS. The TOPSIS algorithm generates fitness value for PSO technique using three multi-criteria, viz. execution time, transmission time, and cost. The algorithm has been found suitable for task scheduling approaches with less computational complexity but lacks load balancing issues. Hence, to get an efficient outcome, we performed a multi objective-based TOPSIS-PSO algorithm (using four objectives) for scheduling of the tasks in our dynamic load balancing approach.

A new hybrid task scheduling technique named PSO with gravitational emulation local search (PSO-GELS) has been introduced for grid computing [36]. The algorithm PSO-GELS perfectly examines its role with respect to MakeSpan. We compared it with our proposed CMODLB load balancing algorithm with respect to the obtained MakeSpan values for the same configurations set because of its dynamic nature. Previously, authors [36] have compared their work with various other existing algorithms such as SA, GA, GA-SA, GA-GELS (genetic algorithm–gravitational emulation local search), PSO, and PSO-SA.

Authors of [49] have introduced a crow–penguin optimizer which is the fusion of the crow search optimization algorithm (CSA) and the penguin search optimization algorithm (PeSOA) for the execution of multi-objective task scheduling strategy (CPO-MTS). The introduced CPO-MTS algorithm performed the execution of tasks in a minimal time by a converging problem to a global optimal solution rather than the local optima. Authors have compared their work with various other existing algorithms such as CPO, CSA, PSO, ABC, GA, ACO, and PeSOA. The introduced work is suitable for load balancing in a static cloud environment which makes the algorithm less effective.

A novel task scheduling technique named threshold-based multi-objective memetic optimized round robin scheduling (T-MMORRS) is presented to offer high-quality services and maintains bursty user demands [50]. Initially, some user requests are transferred to the server where the proposed algorithm performed a quick scan for workload conditions through a burst detector. Furthermore, if the obtained result by burst detector has a normal load, then T-MMORRS chooses threshold multi-objective memetic optimization (TMMO) else T-MMORRS will choose weighted multi-objective memetic optimized round robin scheduling (WMMORRS) for burstiness load. T-MMORRS technique is compared with the multi-objective genetic algorithm (MGA) and dynamic power-saving resource allocation (DPRA). T-MMORRS achieves higher efficiency and lower time consumption. However, the algorithm offers high complexity to the cloud system.

An efficient load balancing system using an adaptive dragonfly algorithm was proposed by the authors of [51]. Completion time, processing costs, and load parameters are used to develop the multi-objective function. Authors have compared their work with dragonfly optimization algorithm (DA) and firefly algorithm (FA) and claim that the proposed method has better performance than the respective algorithms. The limitation of this work was that the author has not focused on load fairness and resource utilization cloud parameters which makes the algorithm less efficient. For more detailed survey comparison of various existing load balancing algorithms have been discussed in Table 1.

4 Proposed work

4.1 System overview

The proposed work focuses on load balancing among PMs and VMs in a cloud environment through hybrid supervised (with target attribute, *i.e.,* ANN) and unsupervised (without target attribute, i.e., BOK-means clustering) machine learning techniques for efficient load calculation and VM clustering process. The proposed cloud environment consists of M numbers of PMs as:$P = \left\{PM_1, PM_2, \ldots, PM_M\right\}$. In each PM, L numbers of VMs are included as: $VM = \left\{VM_1, VM_2, \ldots ., VM_L\right\}$. The cloud environment is involved with S number of users tasks represented as $T = \left\{T_1, T_2, \ldots, T_S\right\}$. To balance load among M PMs and L VMs, two entities, VM manager (VM_{Man}) and cloud balancer (C_{Bal}), are involved.

4.1.1 4.1.1 Contribution methods

The proposed cluster-based multi-objective dynamic load balancing (CMODLB) method is introduced for an efficient load balancing without the loss of resource utilization. Figure 2 depicts the overall functioning of CMODLB method. To achieve the goal, machine learning- and soft computing-based techniques have been used for each stage to learn the behavior of the cloud to develop the effectiveness and better performance of the cloud environment. The CMODLB method comprises the following three stages:

Table 1 Comparison of previous existing load balancing algorithms

Year/references	Work highlights	Methodology	Simulator	Benefits	Tribulations
2016 [16]	Adaptive cost-based task scheduling (ACTS)	Priority is used for the tasks based on their deadline. Cost of data access path is measured from the completion time on each VM	CloudSim	Reduces execution time, computation cost, communication cost, and bandwidth. Increases CPU utilization	Introduces underutilized resources and load unbalance
2016 [17]	Distributed two-level cloud-based multimedia task scheduling (CMS)	Multimedia cloud-based task scheduling approach is proposed using cooperate game theory to enhance multimedia cloud *QoS*	SimPy	Enhances fairness to multi-class multimedia application	It limits waiting space for tasks
2017 [18]	Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in the cloud (TaPRA and TaPRA-Fast)	Introduces an analytical task placement and resource action strategy for nonlinear mixed integer programming problem	Offline simulation	Minimize energy consumption and job completion time	The algorithm is not efficient for multiple tasks
2017 [19]	Energy optimization with dynamic task scheduling mobile cloud computing (EDTS)	Introduces a dynamic voltage scaling-based DFGCP and CPA algorithm to accelerate power consumption	Android emulator	Minimize the total energy consumption for smartphones	Resource utilization needs to be increased for lower energy consumption
2015 [20]	A hybrid meta-heuristic algorithm for VM scheduling with load balancing in cloud computing (ACOPS)	Hybridization of ACO and PSO incorporates to historical information to predict the upcoming workload of requests	C compiler, Testbed@NCKUEE	The computing time of the scheduling is reduced by rejecting requests that are not suitable	Increases space complexity due to historical information maintenance

Table 1 (continued)

Year/references	Work highlights	Methodology	Simulator	Benefits	Tribulations
2017 [21]	Multi-service load balancing with hybrid particle swarm optimization in a cloud-based multimedia storage system with *QoS* provision (MKL-SVM)	The proposed algorithm rectifies the disturbance in the utilization of multiple resources. The proposed model CMSdynMLB serves the load balancing	Cloud system	Enhances resource utilization	Increases computational complexity and load balancing time
2013 [22]	Honey bee behavior-inspired load balancing of tasks in cloud computing environment (HBB-LB)	HBB-LB performed load balance using migration operation. Algorithm migrates overloaded VM to underloaded VM	CloudSim	The algorithm works well in heterogeneous cloud systems	Increases MakeSpan and underutilization of resources
2013 [23]	Artificial neural network-based load balancing in cloud environment	ANN-LB performs clustering of VMs into underloaded and overloaded so that underloaded VM is used for further task mapping process	CloudSim	Increases MakeSpan and reduces load overhead	VM migration, efficient task scheduling, and PM level load are not considered, which lead to less efficient load balancing
2017 [24]	A hybrid strategy for resource allocation and load balancing in virtualized data centers using BSO algorithms	The algorithm is the hybridization of PSO and BSO algorithm to achieve load balance in DC	CloudSim	Increases scheduling efficiency, MakeSpan and utilization	Load on PM is not considered which makes inefficient load balance
2016 [25]	Dragonfly optimization and constraint measure-based load balancing in cloud computing	A threshold is set to categorize overloaded VM and underloaded VM	CloudSim	Migration of tasks is less in numbers	Suffers from high processing time and MakeSpan

Table 1 (continued)

Year/references	Work highlights	Methodology	Simulator	Benefits	Tribulations
2017 [26]	A heuristic clustering-based task deployment approach for load balancing using Bayes theorem in cloud environment	Physical host with their probability is introduced by Bayes theorem for task mapping	CloudSim	Enhances load balance within datacenters	Not efficient to balance load across VMs
2017 [27]	Live VM migration under time constraints in shared nothing IaaS clouds	Migration decision was made based on RAM and CPU state of VMs, and migration was performed by deriving policies by hypervisor	Xen 3.21 and OpenNebula	Offers time-constrained VM migration requests and reduces SLA violation	Complexity is higher
2016 [28]	Energy-aware virtual machine migration for cloud computing—a firefly optimization approach	FFO-EVMM algorithm also utilizes the concept of artificial bee colony (ABC) algorithm for load monitoring on VMs	CloudSim	Enhances better average energy consumption	Takes more migration time
2016 [29]	Energy-aware VM consolidation in cloud data centers using utilization prediction model	VM consolidation model was applied periodically to optimize VM placement, and VM migration was enabled if any resource of VM exceeds total capacity	CloudSim	Avoids unnecessary VM migration and SLA violation. Energy consumption is reduced	Consumption of energy is higher as the number of migration is high
2015 [30]	Enhanced load-balanced min-min algorithm for static meta task scheduling in cloud computing	Tasks were assigned to VMs based on execution time, and tasks were rescheduled to utilize unused resources	CloudSim	Better MakeSpan and resource utilization	Not suitable for dynamic cloud

Table 1 (continued)

Year/references	Work highlights	Methodology	Simulator	Benefits	Tribulations
2015 [31]	Multi-objective tasks scheduling algorithm for cloud computing throughput optimization	VMs are sorted on the basis of MIPS values. The VM list which is prepared initially is processed with all task set	CloudSim	Reduces overall execution time	The scheduling process is not efficient
2017 [32]	Dynamic cloud task scheduling based on a two-stage strategy	Task queue, waiting queue, and ready queue were maintained, and the tasks were put in wait queue until a suitable VM became idle	CloudSim	Efficient task scheduling and execution	Maintaining multiple queues and databases increase space complexity
2014 [33]	Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS Cloud	The proposed framework performs resource allocation for IaaS providers with the flexibility to send tasks to external clouds to get sufficient resources	MATLAB 7.0	CPU and memory utilization are better	Time complexity and time consumption are higher
2015 [34]	Enhanced particle swarm optimization for task scheduling in cloud computing environments	Proposed load balancing algorithm LBMPSO distributes tasks to resources and finishes tasks as early as possible. Algorithms also reschedule failure tasks	CloudSim	Round trip time is best achieved by the LBMPSO	The method promises large MakeSpan and complexity for a higher number of VMs and tasks

Table 1 (continued)

Year/references	Work highlights	Methodology	Simulator	Benefits	Tribulations
2019 [35]	TOPSIS–PSO-inspired non-preemptive tasks scheduling algorithm in a cloud environment	A mathematical multi-criteria-based technique TOPSIS is used to get objective function for PSO. Fitness function is obtained from three multi-criteria, *i.e.,* execution time, transmission time and cost	CloudSim	Achieves better MakeSpan and transmission time. Reduces cost	Not efficient for load balancing
2015 [36]	An efficient meta-heuristic algorithm for grid computing	A PSO and GELS combined strategy is introduced for scheduling tasks in grid computing	JAVA Software	Better execution cost and total execution time	PSO-GELS is inefficient on load balancing issues
2013 [45]	FUGE: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method	A genetic algorithm and fuzzy-based approach called FUGE is implemented to perform job assignment to the appropriate resource	CloudSim	Better execution cost and total execution time	FUGE does not work well on load balancing issues
2020 [49]	Crow–penguin optimizer for multi-objective task scheduling strategy in cloud computing	CPO-MTS algorithm is used for multi-objective based method. It gives an improved convergence rate and converges to the global optimal solution	CloudSim	Maximizes QoS and minimizes, resource utilization cost, MakeSpan and load	Load fairness and resource utilization are not discussed which makes work less efficient

Table 1 (continued)

Year/references	Work highlights	Methodology	Simulator	Benefits	Tribulations
2019 [50]	Threshold-based multi-objective memetic optimized round robin scheduling for resource-efficient load balancing in cloud	T-MMORRS technique improves task scheduling to balance the bursty and non-bursty VM workloads	CloudSim	Reduces energy utilization	MakeSpan, resource utilization, degree of imbalance performance metrices are not concerned
2020 [51]	An efficient load balancing system using adaptive dragonfly algorithm in cloud computing	Efficient load balancing is achieved using Adaptive Dragonfly algorithm (ADA) having multi-objective function for completion time, processing costs and load	CloudSim	Better execution time and execution cost	Not efficient to balance load in dynamic cloud environment

4.1.1.1 Stage I: Grouping of VMs using BOEK-means with ANN In this stage, the load of cloud VMs under PMs is calculated using Bayesian optimization-based K-means with artificial neural network as shown in Fig. 3. The reason behind using ANN is to support multiple VMs simultaneously to get the current load with the objective to reduce the total clustering method. In our previous work [23], we investigated that the K-means clustering algorithm offers a major shortcoming in the initialization of the centroid which makes it more expensive to evaluate the functions. To overcome this problem, Bayesian optimization (BO) algorithm which builds a probabilistic model for the problem and finds posterior predictive distribution for that problem [37, 38] is introduced with the K-means algorithm.

4.1.1.2 Stage II: Task scheduling using TOPSIS-PSO method For efficient task scheduling, the technique of order preference by similarity to ideal solution with particle swarm optimization (TOPSIS-PSO) algorithm [35] is performed with different cloud objectives. A VM is allocated to a task which minimizes execution time and transmission time and maximizes CPU utilization. Here, PSO algorithm is combined with a multi-objective-based TOPSIS algorithm in order to remove the PSO's weak local search ability and to find optimal fitness function by considering three criteria. The relative closeness is formulated by the TOPSIS algorithm which is the objective function for PSO. All underutilized VMs are taken in PSO algorithm, and fitness value is calculated using multi-objective TOPSIS algorithm. The use of multi-objective TOPSIS method introduces the most efficient task scheduling outcome. PSO algorithm is set with tasks on underloaded VMs ($T_U = \{T_{U1}, T_{U2}, \dots, T_{UQ}\}$), and at each iteration, the fitness value for a particle is calculated. The fitness value gives particle local best (LBest) and global bbest (GBest) values, and both values are updated.

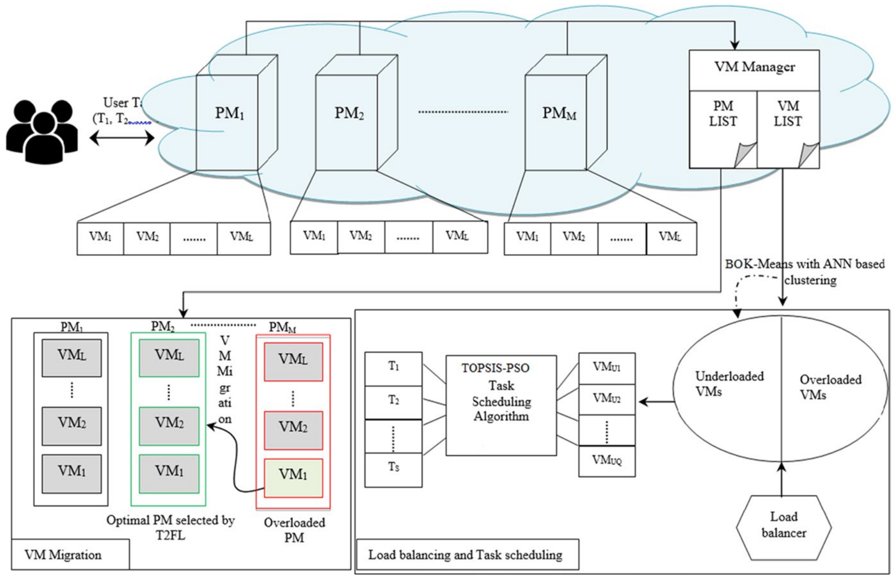


Fig. 2 CMODLB method

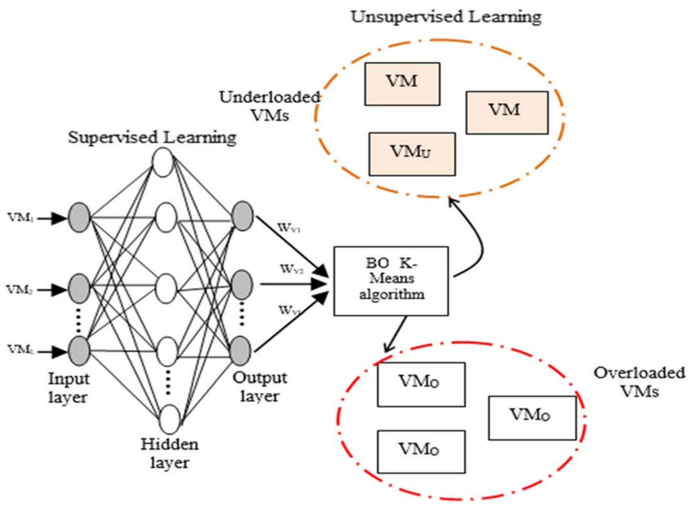


Fig. 3 BOK-Means with ANN-based clustering

4.1.1.3 Stage III: VM migration using Iterative Type 2 Fuzzy Logic (IT2FL) method For PM-level load balance, it is necessary to maintain destination PM with a balanced load even after VM migration. Available resources are also important in destination PM selection which provides better performance for user tasks. Therefore, a selection of optimal destination PM should be performed in an efficient manner. To realize this fact, interval type 2 fuzzy logic (IT2FL) is incorporated in VM migration process as shown in Fig. 4 that illustrates the flow diagram of IT2F logic for PM selection. Here, rule base follows rules to obtain fuzzy output. PM with high output is taken as PM_{des} (destination PM). Lotfi A. Zadehi introduced IT2FL set to extend the functional properties of Type1 and general fuzzy logic systems. IT2 FL gives the possibility to

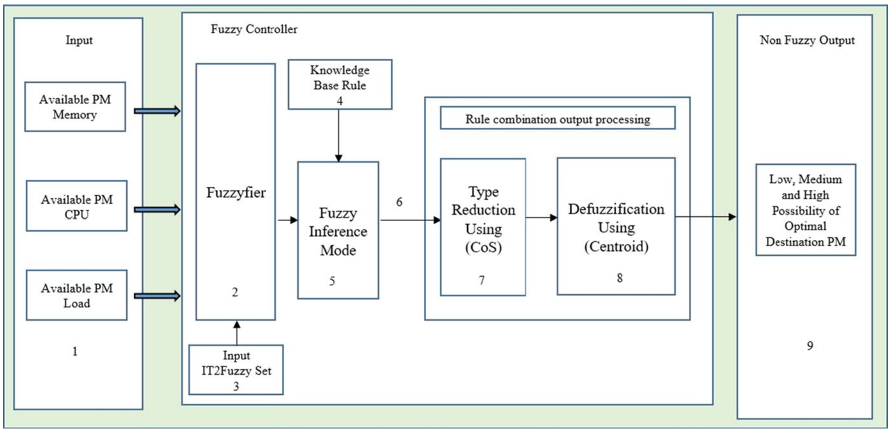


Fig. 4 IT2 FL logic for PM selection

provide more parameters to describe membership functions (MF) and handles more uncertainty [40]. It is the first-order uncertainty fuzzy set model.

The overall process of the CMODLB method includes a grouping of underloaded and overloaded VMs using supervised ANN-LB technique, scheduling of tasks by TOPSIS-PSO multi-criteria-based technique, and VM migration using iterative type 2 fuzzy logic (IT2FL) method is depicted in Fig. 2 where as Table 2 explains description of used notations in the manuscript. Each significant stage has been discussed in detail in the following sections.

4.2 BOEK-means with ANN-based clustering (Stage I)

Knowledge of the currently available load on VMs will improve task scheduling, thus reducing the unbalanced DCs and decreasing resource underutilization. To achieve this, we have introduced VM_{Man} that will perform clustering of VMs based on their current available load in each PM; VM_{Man} maintains VM list with their load information and PM list with their information. VMs are clustered as per their current load given as follows:

Table 2 Notations and their description

Notation	Description
VM	Virtual machine
PM	Physical machine
DC	Data center
T_S	Task from *sth* user
MIPS	Million instructions per second
E_L	Execution time of *sth* task on *Lth* VM
VM_{Man}	VM manager
C_{Bal}	Cloud balancer
N_P^L	Number of processors in *Lth* VM
N_{MIPS}^L	Number of MIPS in *Lth* VM
BW^L	Bandwidth of *Lth* VM
VM_U	Cluster with underloaded VMs
VM_O	Cluster with overloaded VMs
W_L, $C(V_L)$,L_L	Weight, capacity, load of *Lth* VM
E_T, $Trans_T$,, U_{CPU}	Execution time, transmission time, CPU utilization of *sth* task on *Lth* VM
C	$C=(c_1, c_2,.., c_K)$ is the set of centroid
$\gamma(c)$	Bayesian optimization for best centroid
G	$G=\{G_1, G_2\}$ Cluster groups
PM_{Over}	Overloaded physical machine for VM migration process
PM_{under}	Underloaded physical machine
PM_{des}	Destination physical machine for VM migration process
V_{op}	Optimal VM for migration process

- ***Underloaded VMs:*** Set of VMs having load lower than the target value, and it is denoted as $VM_{U.}$
- ***Overloaded VMs:*** Set of VMs having load greater than the target value, and it is denoted as $VM_{O.}$

Group formation of VMs is realized by BOK-means with ANN algorithm. In BOK-means with ANN algorithm, all VMs are fed into ANN to calculate their current load. The load on each VM is considered as weight values and denoted as $W = \{W_{V1}, W_{V2}, \ldots, W_{VL}\}$. Based on the weight value of each VM, BOK-means algorithm forms clusters such as underloaded VMs with $VM_U = \{VM_{U1}, VM_{U2}, .., VM_{UQ}\}$ and overloaded VMs with $VM_O = \{VM_{O1}, VM_{O2}, .., VM_{OR}\}$. Weight value in terms of the load is calculated using Eq. (1) as follows:

$$W_i = L_i \frac{\frac{1}{E_j} \sum E_j \forall \text{ assigned task on } V_L}{C(V_L)} \tag{1}$$

where W_L and L_L denote the weight and load on Lth VM, while $E_{j\ s}$ represents the execution time of Sth assigned tasks on $\mathrm{VM_L}$. *$C(V_L)$* provides the capacity of $\mathrm{VM_L}$ and computed as:

$$C(V_L) = \left(N_P^L \times N_{MIPS}^L + BW^L\right) \times \frac{1}{100}. \tag{2}$$

The capacity of each Lth VM ($C(V_L)$) is calculated by considering the number of processors ($\mathrm{N_P}$), the number of million instructions per second (MIPS), and bandwidth (BW) of Lth VM. Figure 3 illustrates BOK-means with ANN algorithm-based clustering process. The optimal centroid maximizes the performance of the K-means clustering algorithm. For a given function *f(c)* that represents optimization problem and *c* represents the attribute of user task belonging to the compact set of centroids *(c ∈ C)*, the probability of clustering improvement is expressed as:

$$\text{Improvement} = \emptyset(\gamma(c)) \tag{3}$$

where$\gamma(c)$ is obtained from

$$\gamma(c) = \frac{f(c_{\text{best}}) - \mu_c}{\sigma(c)}. \tag{4}$$

Here, μ_c is the predictive mean function and $\sigma(c)$ is a predictive marginal function. Improvement in an optimal solution is obtained from Eq. (3). After the initialization of the optimal centroid, all VMs are initialized with their weight values. Then BOK-means method was performed to find optimal clusters.

Algorithm:1 BOK-means with ANN algorithm (stage 1)

```
     Input: Set of VMs, V={v1, v2,..,vL}, Number of clusters k, Maximum iteration I.
     Output: Set of K groups, G={G1, G2,..,Gk}
     Begin
     //ANN
     Initialize V
1:   Fed V into ANN
2:   For each $v \in V$,
3:   Compute L by Eq.(1)
4:   Assign $W \leftarrow L$
5:   Find W={$w_{v1}, w_{v2}, w_{v3}, .., w_{vL}$}
6:   //BOEK-means
7:   for I=1, 2…..(Max iteration) do
8:   Calculate ($c_i$)=arg max f(c)
9:   
10:  Problem Function $\gamma(c) = \frac{f(c_{best})-\mu_c}{\sigma(c)}$ from Eq.4
11:  Augment new point to the $c_i$
12:  Update current location of optimum K-centroid
13:  Apply Eq.(3) to the updated optimum K-centroid
14:  end for
15:  Initialize k-centroid C=(c1, c2,.., cK) by Eq. (3)
16:  Initialize W
17:  for each $c \in C$ do
18:      $c_L \leftarrow v_L \in V$
19:  end
20:  for each $v \in V$
21:      Compute $d(v_L) \leftarrow argminDistance(v_L, c_L)$
21:  end
22:  While (iteration<I)
23:  Repeat
24:  for each $c_L \in C$ do
25:      Update cluster
26:  for each $v_L \in V$ do
27:      $minDistance \leftarrow argminDistance(v_L, c_L)$
28:  If ($minDistance \neq d(v_L)$)
29:      $d(v_L) \leftarrow minDistance$
30:      changed ←true
31:  end if
32:  end for
33:  iteration ++
34:  until changed=true
35:  end while
36:  end
37:  until the maximum of iteration reached or a solution found
38:  return cluster formation (best solution found)
```

Algorithm 1 depicts the complete process of clustering of VMs using the BOK-means method ANN algorithm. The obtained cluster includes g_1 and g_2 where

$$g_1 = VM_U = \{VM_{U1}, VM_{U2}, \dots, VM_{UQ}\}\ g_2 = VM_O = \{VM_{O1}, VM_{O2}, \dots, VM_{OR}\}.$$

Here,g_1 includes a cluster of underloaded VMs and g_2 includes a cluster of overloaded VMs. In the cloud, each VM is able to execute different tasks having dissimilar execution times. This process of execution of different tasks in different VMs may dynamically vary the current load of VMs. Because of this dynamic execution nature, it may happen that a VM in g_1 may achieve heavy load or a VM in g_2 may lead to achieve less load. Such factor motivates to introduce a balancer that will maintain each cluster group without losing their uniqueness. The key role of the balancer is to make a decision on load balancing by assigning its each VM to a suitable cluster location. For example, if any VM in g_1 has changed its current load, then it is the responsibility of the balancer to remove that VM from g_1 and put it into a suitable cluster location in g_2. Hence, each cluster maintains optimal VMs. To understand its concept more deeply, let us consider four VMs in the cloud system having VM_ID from 0 to 3. Each VM initially assigned some tasks to their specific IDs (0 to 3). Using ANN, the obtained VM loads are $VML_1 = 0.19$, $VML_2 = 0.19$, and $VML_3 = 0.20$. Further, assume that using ANN the obtained values are clustered into the underloaded and overloaded clusters using the BOK-means method. To classify VM_U and VM_O, a threshold is set from which $g_1 = \{VML_2 \text{ and} VML_4\}$ is obtained as an underloaded and $g_2 = \{VML_1 \text{and} VML_3\}$ is obtained as an overloaded cluster. The threshold is calculated using Eq. (6) and Eq. (7), respectively. Obtained VM loads are depicted in Table 3.

$$\text{Dist} = \sum_{j=1}^{g} \sum_{i=1}^{n} VML_i^{(j)} - m_j^2 \quad \forall j = 1, 2, \ldots, p \quad \forall k = 1, 2, \ldots, m \tag{5}$$

$$g_1 = \sum_{VM=1}^{j} VML_j \tag{6}$$

$$g_2 = \sum_{VM=1}^{i} VML_i \tag{7}$$

if $[g_1 < g_2]$ then g_1 is Underloaded. else g_1 is overloaded. if $[g_1 = g_2]$ then L_{Bal} will assign tasks to VM having minimum load among VMs of g_1 and g_2 both. The obtained underloaded VMs are further being used for task assignment execution using a multi-objective-based TOPSIS-PSO scheduling algorithm.

4.3 Multi-objective-based TOPSIS-PSO scheduling algorithm (Stage II)

The next step is task scheduling process in which each task is allocated to the optimum underloaded VM of cluster g_1. The task scheduling process aims to balance the load between VMs and to make the best use of resource utilization. In this process, to preserve load among VMs, the set of VMU determined by BOK-means with ANN algorithm is taken and other VMs have been excluded. Incoming user tasks $T = \{T_1, T_2, \ldots, T_S\}$ are assigned to optimal underloaded VM in

Table 3 Obtained VM loads

VM_Id	*VM*	*Task_Id*	*VML*
0	1	0	0.19
1	2	1	0.19
2	3	2	0.20
3	4	3	0.19

$VM_U = \{VM_{U1}, VM_{U2}, \ldots, VM_{UQ}\}$. Based on multiple parameters such as the execution time of task, the transmission time of the task, and CPU utilization, an optimal VM is selected.

The execution time (E_T) of *S*th task on *L*th VM is given by

$$\text{To be Minimized, } E_T = \frac{\text{length}_S}{\text{MIPS}_L}. \tag{8}$$

Execution time is computed by obtaining the ratio of the length of the *S*th task $length_S$ and MIPS of *L*th VM. Optimal VM is nominated for *S*th task that reduces E_T.

The transmission time (T_T) for a task T_S on VM V_L is computed as

$$\text{To be Minimized, } T_T = \frac{\text{Size}_S}{BW_L}. \tag{9}$$

The transmission time of the S^{th} task on a specific VM is obtained by taking the ratio of task size $Size_S$ and bandwidth of VM BW_L. Similarly, CPU utilization of VM is calculated by

$$\text{To be maximized, } U_{CPU} = \frac{\text{Total CPU usage of processes}}{\text{Number of processes}}. \tag{10}$$

The TOPSIS algorithm is initiated by assigning some weight values for every three criteria which are signified by $[E_T, T_T, U_{CPU}] = [H_E, H_{trans}, H_U]$ and includes the following steps:

Step1 Construction of Decision Matrix (DM). The decision matrix is constructed using multiple alternatives and multiple criteria.

Here, T_{U1}, T_{U2}, T_{U3}, T_{U4}, and T_{U5} are called alternatives (tasks) and E_T, T_T, and U_{CPU} are known as multiple criteria. E_{T1i}, T_{T1i}, and U_{CPU1i} represent the execution time, transmission time, and CPU utilization of T_i on VM_{UQ}, respectively. The obtained DM values for five tasks T_{U1}, T_{U2}, T_{U3}, T_{U4}, and T_{U5} on underloaded VM_U (vmID = 0) with E_T, T_T, and U_{CPU} are expressed in Table 4.

Step2 Construction of Standard DM. In this step, each criterion is compared with each column alternative to transform into non-dimensional attributes. In this standardization, each row of DM is divided by the root of the sum of the square of respective row as follows:

Table 4 Decision table of multiple alternatives and criteria

		Multiple alternatives						
		T_{U1}	T_{U2}		T_{U3}	T_{U4}		T_{U5}
Multiple criteria	E_T	0.256	0.282	0.307		0.333	0.359	
	T_T	1.150	1.100	1.200		1.300	1.400	
	U_{CPU}	0.800	0.800	1.000		1.000	1.000	

$$S_{ac} = \frac{DM_{ac}}{\sqrt{\sum_{i=1}^{m} DM_{ac}^{2}}} \tag{11}$$

where each element in DMrepresents in ath alternative and cth criteria for DM_{ac}.

Step3 Construction of Weighted standard DM. To get weighted standard DM, each attribute weight value is multiplied by each element in standard DM. The weight values for execution time, transmission time, and CPU utilization criteria are evaluated as

$$E_T = E_{TQ} \times H_E \tag{12}$$

$$T_T = T_{TQ} \times H_{Trans} \tag{13}$$

$$U_{CPU} = E_{TQ} \times H_U \tag{14}$$

where $H_E, H_{trans}$and H_U are the weight values for E_T, T_TandU_{CPU}, respectively, and E_{TQ}, T_{TQ}andU_{CPUQ} on Qth VM. The weight values should be a positive integer.

Step4 Evaluation of ideal and negative solution. In this step, an extremely positive ideal solution that maximizes benefit criteria and a negative ideal solution that minimizes benefit criteria are determined. $\mathrm{E_T}$ and $\mathrm{T_T}$ are considered as performance criteria which are to be minimized, and $\mathrm{U_{CPU}}$ is taken as benefit criteria which are to be maximized. Positive ($\mathrm{A^+}$) and negative ($\mathrm{A^-}$) ideal solutions are given by

$$A^{+} = \left(V_{U1}^{+}, V_{U2}^{+}, .., V_{Ua}^{+}\right) \tag{15}$$

$$A^{-} = \left(V_{U1}^{-}, V_{U2}^{-}, .., V_{Ua}^{-}\right) \tag{16}$$

where V_{Ua}^{+} and V_{Ua}^{-} represent the positive and negative solutions for ath alternative, respectively.

Step5 Determination of separation measures. In this step, each alternative is separated from A^+ and A^- which is measured as

$$f_Q^{+} = \sqrt{\sum_{c=1}^{3} \left(V_{ac} - V_c^{+}\right)^2} \tag{17}$$

$$f_Q^{-} = \sqrt{\sum_{c=1}^{3} \left(V_{ac} - V_c^{-}\right)^2} \quad a \in alternatives\ c \in criteria. \tag{18}$$

Equations (17) and (18) calculate the distance between each alternative positive and negative solution, respectively. Here, the number of criteria is 3.

Step6 Calculation of relative closeness. The value of relative closeness (RC) is achieved from the positive and negative separation measures (f_Q^+ and f_Q^-) of tasks on Qth VM with respect to positive ideal solution $\mathrm{A^+}$ and is defined as

$$FV_Q = RC_Q = \frac{f_Q^-}{f_Q^- + f_Q^+}. \tag{19}$$

The role of RC is to perform as a fitness value for the PSO algorithm. The ability of particle swarm reaction introduces solutions to optimization problems [36]. In PSO, a swarm comprises individual particles that fly on search space, and these particles handle solutions to the problems. By the experience of a particle and its neighboring particles, the position may be influenced by the best local position (LBest). However, if the neighborhood of a particle has the best position in the swarm, it would be the best global position (GBest). Once particle has been initialized, in each repetition particle is placed near to GBest and velocity using Eq. (20) and FV is calculated. The velocity of a particle is expressed as

$$ {}_{Q}^{i+1}Vel = \omega * {}_{Q}^{i}Vel + L^1 * ran^1 * \left(LBest - {}_{Q}^{i}Par \right) + L^2 * ran^2 * \left(GBest - {}_{Q}^{i}Par \right) \tag{20}$$

$$ {}_{Q}^{i+1}Par = {}_{Q}^{i}Par + {}_{Q}^{i+1}Vel \tag{21}$$

where ${}_{Q}^{i+1}Vel$ signifies existing velocity and ${}_{Q}^{i}Vel$ denotes the earlier velocity of Q. ${}_{Q}^{i+1}Par$and${}_{Q}^{i}Par$ are the current and earlier position of Q. Two learning factors L^1, L^2 and the random numbers ran^1, ran^2 are involved in velocity computation with respect to the values depicted in Table 4. Each user task performs this process and is then scheduled to an optimal underloaded VM. The obtained relative closeness for each task T from TOPSIS is initialized to compute the fitness value for each particle. Initially, for each task, TOPSIS calculates a decision matrix for the given multiple alternatives (tasks) and three criteria (execution time, transmission time, and CPU utilization) using Eqs. (8), (9), and (10), respectively. The algorithm recalculates the standard decision matrix by a root of the sum of squares of respective rows using Eq. (11). Weighted standard DM is calculated using Eq. (12), Eq. (13), and Eq. (14) for E_T, $Trans_T$, andU_{CPU}, respectively. Next, positive and negative ideal solutions are performed to check maximum and minimum benefit from the criteria using Eq. (15) and Eq. (16). Separation of alternatives from f_Q^+ and f_Q^- using Eq. (17) and Eq. (18) is performed. Relative closeness (fitness value) is obtained for each task with respect to each alternative as a particle using Eq. (19). The obtained FV value is further compared to the current local best particle value, which is calculated in the initial steps of PSO. If the current FV is larger than the current local best particle, then update the value of LBest with the current FV value updated, else the value of LBest will remain the same. Once the algorithm gets the value of all LBest, the largest LBest is found by comparing them with each other. The largest LBest will be considered as the global best particle GBest. Similarly, if the current GBest is larger than the previous GBest, then the value of GBest is updated with the current GBest value; else, the value of LBest will remain. The process is iteratively performed until the maximum number of iterations is reached. During each iteration, particle velocity and position are updated using Eq. (20) and Eq. (21). The final particle solution having the largest GBest will be assigned to the most optimal underloaded VM $V_{Uoptimal}$ for mapping.

By assigning each task to optimal VM, MakeSpan, completion time, and transmission time of tasks are reduced significantly. TOPSIS-PSO method assigns user tasks only to VM_U. Therefore, TOPSIS-PSO-based task scheduling process with ANN-LB technique introduces load balancing and efficient tasks scheduling among VM_S. Further, it reduces MakeSpan, completion time, and transmission time of tasks without reducing resource utilization. The overall procedure of the TOPSIS-PSO algorithm is shown in *Algorithm 2*. Finally, each task in the user task set is scheduled to optimal VM in the underloaded VM set, which can be expressed as

$$\left[T_1, T_2, T_3, .., T_S\right] = \left[VM_{Ut1}, VM_{Ut2}, VM_{Ut3}, \dots, VM_{Uts}\right]. \tag{22}$$

Tasks which are assigned to $VM_{U2}$and VM_{U4} are denoted in Table 5 where tasks $\boldsymbol{T}_{0,}$ $\boldsymbol{T}_3$, and $\boldsymbol{T}_4$ are mapped to $\boldsymbol{VM}_{U2}$ and Tasks $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$ are mapped to $\boldsymbol{VM}_{U4}$. The given task assignment matrix table represents task mapping according to the increased number of tasks and VMs. The calculation is executed for both less and a considerable number of tasks, and related effects are observed for both cases. In this observation, a small number of VMs and runtime tasks are used.

The assignment of the best VM for user task results in better performance and better resource utilization. Therefore, MakeSpan, which is the overall completion time for all assigned tasks for a particular VM, is minimized significantly. Involvement of LBal efficiently monitors the load on grouped VMs and reassigns VM after task assignment to prevent VMs from overloading. Hence, balanced load among all VMs is maintained by efficient clustering and task scheduling processes. It is also necessary to balance load among PMs. PSO parameters are depicted in Table 6. Load balancing among PMs is realized by the VM migration process, which is elaborated in the next section.

Table 5 Tasks assignment matrix to VMs

Tasks	Resources	
	VM_2	VM_4
$\boldsymbol{T}_0$	1	0
$\boldsymbol{T}_1$	0	1
$\boldsymbol{T}_2$	0	1
$\boldsymbol{T}_3$	1	0
$\boldsymbol{T}_4$	1	0

Table 6 PSO parameters

Parameters	Values	Description
Swarm size	No. of particles	Problem space in terms of swarm size
Particle(Q)	5	Particles or population in swarm
Iterations	50–1000	Number of swarm iterations
Constant factor (L1)	1.49445	Self-consciousness study factor [35] or cognitive parameter [36]
Constant factor (L2)	1.49445	Swarm consciousness study factor [35] or social parameter [36]
Max(V)	(No. of resources or 1)	Maximum velocity of a particle
Min(V)	(0–1)	Minimum velocity of a particle
rand	(0–1)	Uniformly distributed random number

Algorithm:2 TOPSIS-PSO Scheduling Algorithm (Stage II)

```
    Input: User tasks, T={$t, t_2, t_3, .., t_S$},VMU={$VM_{U1}, VM_{U2},.., VM_{UQ}$}
    Output: Optimal VM for user Tasks
    Begin
    Obtain T
1:  Initialize particles with LBest & GBest
2:  for each $t \in T$
3:      Construct DM
4:      Standardize DM by Eq. (11)
5:      Construct DM using Eq.(8),(9),(10)
6:      Construct $S_{ij}$ standard DM using Eq.(11)
7:      Evaluate Weighted standard DM by Eq.(12),(13),(14)
8:      Calculate ($A^+$) and ($A^-$) ideal solutions using Eq.(15),(16)
9:      Evaluate separation Measures $f_Q^+$ and $f_Q^-$ using Eq. (17) (18)
10:     while (iteration>t) do
11:         for each particle //Compute FV
12:             Calculate relative closeness as FV using Eq. (19)
13:         end for
14:         if (FV>LBest)
15:             Update LBest ← FV
16:         else
17:             Keep LBest
18:         end if
19:         Compare all LBest
20:         Assign high LBbest ← GBest
21:         if(current GBest>previous GBest)
22:             Update current GBest ← GBest
23:         else
24:             Keep previous GBest
25          end if
26:         Particle with high GBest ← $V_{Uoptimal}$
27:         Assign $t$ ← $V_{Uoptimal}$
28:         Compute particle velocity and position using Eq. (20) and (21 )
29:         Iteration ++
30:     end while
31: end for
32: end
33: until the maximum number of iterations reached or a solution found
34: return efficient scheduling of tasks to VMs (best solution)
35:
```

4.4 VM migration based on fuzzy Logic (Stage III)

Both VM clustering and task scheduling are involved in load balancing across VMs, whereas VM migration is involved in load balancing among PMs. Load on each PM is monitored by the VM manager, which maintains PM information. PM load is calculated in terms of load on VMs that are presented in that PM as follows:

$$L(PM_m) = \sum_{i=1}^{L} L_i. \tag{23}$$

Equation (23) computes load on mth PM in terms of load L of all VMs (L_i) presented in that PM. VM migration also enables energy efficiency along with load balancing. VM manager takes VM migration decision in the following conditions:

1. If a PM becomes overloaded (for load balancing)
2. If a PM has less load (for energy efficiency).

When a PM attains one of the above conditions, optimal VM is migrated from that PM to optimal destination PM. Consider PM_m with $\{VM_1, VM_2, .., VM_i, \dots, VM_L\}$ and load on VMs as $\{L_1, L_2, .., L_i, \dots, L_L\}$. If PM_m is overloaded, then the optimal VM is selected and migrated from that PM to another PM. The optimal VM for migration is selected based on migration time, and it is computed for Lth VM as

$$MT(VM_L) = \frac{RAM(VM_L)}{BW_L}. \tag{24}$$

Here, $RAM(VM_L)$ represents RAM of VM_L and BW_L denotes the bandwidth of $\mathrm{VM_L}$. Condition for a VM to be selected as an optimal VM for migration is formulated as

$$optimalVM = \min\{MT(VM)\}. \tag{25}$$

A VM that satisfies the above condition is selected for migration and denoted as V_{op}. The next important step is to find an optimal destination PM for migration. Important mathematical definitions, terminology, and well-explained concepts that explain how IT2FL is set and the system is evaluated in the proposed research work are discussed in the following subsections. Detailed IT2FL set and system process are explained as follows (Fig. 4).

4.4.1 Interval type 2 fuzzy logic set (IT2FL)

Definition 1 A T2 FS is denoted as S' and expressed by its T2 degree of membership function (DoMF) and $\mu_{S'}(x,f)$ is defined as follows:

$$S' = \int x \in X \int f \in D_x \frac{\mu_{S'}(x,f)}{(x,f)} \quad \mu_{S'}(x,f) \leq 1 \tag{26}$$

where expression $x \in X, f \in D_x$ is the domain of x in $\subseteq [0,1]$.The primary variable x of S' can be denoted as X. The use of $\iint$ is to represent the union over, all correct values of x and f. If the universe of discourse (UoD) is discrete in nature, then it will be expressed as

$$S' = \sum_{x \in X} \sum_{f \in D_x} \frac{\mu_{S'}(x,f)}{(x,f)}. \tag{27}$$

In discrete UoD, ∫∫ is simply replaced with $\sum\sum$to represent the values of variables x and f. In this research, the continuous UoD approach is being focused on MF. The value of the fuzzy set S' is expressed as:

$$S' = \left\{ \left((x,f), \mu_{S'}(x,f) \right) \middle| x \in X, f \in D_x \subseteq [0,1] \right\}. \tag{28}$$

where $S'$is the fuzzy set defined over a UoD of primary variable x with T2 DoMF $\mu_{S'}(x,f)$.

Definition 2 If the value of $\mu_{S'}(x,f) = 1$ in Eq. (24), then S' is said to be IT2 FLS.

$$S' = \int x \in X \int f \in D_x \frac{1}{(x,f)},\ D_x \subseteq [0,1] \tag{29}$$

Definition 3 The vertical slice of $\mu_{S'}(x,f)$ is the representation of value x' in x having 2d-plane with axes f and $\mu_{S'}\left(x',f\right)$. Secondary MF is the vertical slice of $\mu_{S'}(x,f)$ [40], *i.e.,*

$$\mu_{S'}\left(x = x',f\right) \equiv \mu_{S'}\left(x'\right) = \int f \in D_{x'} \frac{1}{f},\ D_x \subseteq [0,1] \tag{30}$$

where$\mu_{S'}\left(x'\right)$ is the secondary MF of S′. Equation (30) is used when all secondary MF values of IT2 FLS are 1, otherwise $f_{x'}(f)$ will be replaced with value 1. The value of $f_{x'}(f)$ lies between $0 \le f_{x'}(f) \le 1$.

Definition 4 The representation of $\mu_{S'}(x,f)$ in IT2 FLS is achieved by footprint of uncertainty (FoU). The union of primary membership *(D_x)* is the FoU of S′ as

$$FoU\left(S'\right) = \bigcup_{x \in X} D_x = \{(x,f) | f \in D_x \subseteq [0,1]\} \tag{31}$$

where FoU for lower member function L(MF) $L\mu_{S'}(x)$ and upper member function U(MF) $U\mu_{S'}(x)$ are expressed as follows:

$$L\mu_{S'}(x) = infi(f\ | f \in [0,1], \mu_{S'}(x,f) > 0) \tag{32}$$

and

$$U\mu_{S'}(x) = sup(f\ | f \in [0,1], \mu_{S'}(x,f) > 0) \tag{33}$$

where *infi* and *Sup* are the infimum and supermum of the support of $\mu_{S'}(x)$.

Figure 5 shows the presentable image of IT2 FLS triangular MF for S′ with its two bounded T1 FSs. $L\mu_{S'}(x)$ and $U\mu_{S'}(x)$ are the L(MF) and U(MF), respectively. The region between L(MF) and U(MF) is the footprint of uncertainty (FoU) [40] which is the primary membership that consists of bounded region for IT2 FLS. Any

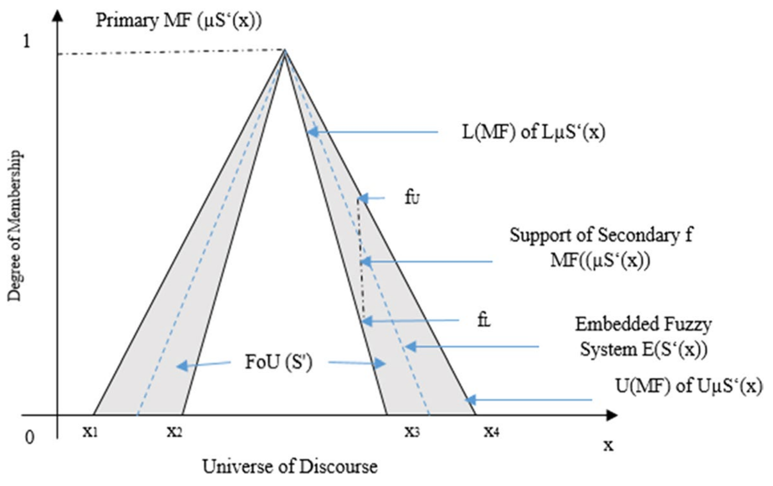


Fig. 5 IT2 FL set MF and FoU

T1 FS within the FoU is the embedded T1 FS, and such sets are represented as L (S') and U $\left(S'\right)$. Each input value in continuous UoD maintains DoMF that ranges the value between L(MF) and U(MF). In this work, triangular MF is used to characterize the fuzzy set. The expression of triangular MF is as follows:

$$\Delta(x : a, b, c) = \begin{cases} 0, & x \le a \\ \dfrac{x-a}{b-a}, & a \le x \le b \\ 1, & x = b \\ \dfrac{c-x}{c-b}, & b \le x \le c \\ 0, & x \ge c \end{cases} \tag{34}$$

A triangular MF $\Delta(a, b, c)$ is the collection of optimistic boundary or lower limit (a), an expected value or middle value (b), and pessimistic boundary or upper limit (c).

4.4.2 Interval type 2 fuzzy logic system (IT2FLS)

IT2FLS is comprised of five stages, *i.e.,* fuzzification, creation of knowledge base rule, fuzzy inference mechanism, type reduction, and defuzzification. The only difference between IT2 FLS and T1 FLS is that T1 FLS does not perform the type reduction stage. Type reduction is the central most part in IT2 FLS in which it transfers the fuzzy output set to T1 fuzzy set. The computational process includes input vectors $x_1 \in S_1', \ldots .., x_m \in S_m'$ and single output $y \in Y$. Knowledge base rules are characterized as Q fuzzy rules (FR) and expressed as [46]:

$$FR_i^{'} : if\ x_1\ is\ L_i^1\ and\ x_2\ is\ L_i^2\ and\ \dots\ x_m\ is\ L_i^m\ then\ y\ is\ O^i\ for\ i \in FR, m \in input \tag{35}$$

where $\mathrm{FR}_\mathrm{i}^{'}$ is the i^th fuzzy rule that consists of a set of universe of discourses $(x_1, x_2 \dots x_m \text{and} y)$ and linguistic variables $(L_i^1, L_i^2..L_i^m \text{and} O^i)$. The part of FR *($x_1$ is L_i^1 and x_2 is L_i^2 and .....x_m is L_i^m)* is said to be the *antecedent* or *premise*, whereas rule statement *(y is O^i)* is the *consequence* or *conclusion*. The execution process of an IT2 FLS and its mathematical terminology is discussed in the following steps:

Step1 Obtain the membership function of x_m, on each $S_m^{'}$ using Eq. (26) which is represented as:

$$\left[L\mu_{S_m^{'}}\left(x_m\right), U\mu_{S_m^{'}}\left(x_m\right)\right] \qquad form = 1, 2 \dots .M \tag{36}$$

where $L\mu_{S_m^{'}}\left(x_m\right)$ and $U\mu_{S_m^{'}}\left(x_m\right)$ are the L(MF) and U(MF) functions of UoD x_m on fuzzy set $S_m^{'}$. The value of L(MF) always must be less than or equal to the value of U(MF).

Step2 Compute the firing interval of the mth rule, $FoU_Q(x)$,

$$FoU_Q(x) = \left[f_L^m, f_U^m\right] \tag{37}$$

wheref_L^m andf_U^m are the upper bound and lower bound firing interval of mth fuzzy rule, respectively. Symbol × denotes the product t-norm operation and expressed as:

$$f_L^m = \left\{L\mu_{S_1^{'}}\left(x_1\right) \times L\mu_{S_2^{'}}\left(x_2\right) \dots \times L\mu_{S_m^{'}}\left(x_m\right)\right\} \tag{38}$$

$$f_U^m = \left\{U\mu_{S_1^{'}}\left(x_1\right) \times U\mu_{S_2^{'}}\left(x_2\right) \dots \times U\mu_{S_m^{'}}\left(x_m\right)\right\} \tag{39}$$

Step3 Compute the type reduction step with the defined rule and firing interval $FoU_Q(x)$ using the center of set (CoS) reducer method.

$$Y_{CoS}(x) = \bigcup_{\substack{f^m \in f^m(x) \\ y^m \in O^m}} \sum_{m=1}^{M} f^m y^m \Big/ \sum_{m=1}^{M} f^m = \left[y^L(x), y^R(x)\right] \tag{40}$$

Perform Karnik–Mendel (KM) algorithm [47] to compute the left- and rightmost points $y^L(x)$and$y^R(x)$, respectively. The KM algorithm is used to obtain the switch points for $y^L(x)$ and$y^R(x)$. The function of $y^L(x)$ is to perform the firing interval switching from upper bound to lower bound. It is the minimum of $Y_{CoS}(x)$ and can be computed as

$$y^L(x) = \frac{\sum_{m=1}^{L} f_U^m y_L^m + \sum_{m=L+1}^{M} f_L^m y_L^m}{\sum_{m=1}^{L} f_U^m + \sum_{m=L+1}^{M} f_L^m}. \tag{41}$$

The function of $y^R(x)$ is to perform the switching firing interval from lower bound to upper bound, and it gets maximum of $Y_{CoS}(x)$. It is computed as

$$y^R(x) = \frac{\sum_{m=1}^{R} f_L^m y_U^m + \sum_{m=R+1}^{M} f_U^m y_U^m}{\sum_{m=1}^{R} f_L^m + \sum_{m=R+1}^{M} f_U^m} \tag{42}$$

where $Ly^L \leq y^L(x) \leq Ly^{L+1}$ $Uy^R \leq y^R(x) \leq Uy^{R+1}$.

Step4 Calculate the defuzzified output from [48],

$$Y = \left[y^L(x) + y^R(x)\right]/2. \tag{43}$$

The above equations will be used, in finding the best possible optimal destination PM. IT2 FLS has unique characteristics that provide better performance even in uncertain abound conditions and work well with missing components. The Java-based toolkit Juzzy [39] library files have been imported to input the IT2 FLS-based VM migration results in CloudSim. Configuration parameters for the fuzzy system are shown in Table 7. To understand the working approach of IT2 FLS in the proposed CMODLB method, a descriptive example is discussed. In our proposed method, we present a model for the *possibility* of selection of the optimal destination PM for VM migration, which depends on available memory (*Avl_Mem*), available CPU (*Avl_CPU*), and available load (*Avl_Load*) of destination PM. It is assumed that the selection of optimal destination PM takes input variable *Avl_Mem*, *Avl_CPU*, and *Avl_Load* having linguistic terms low and high and output variable *Possibility* with linguistic terms low, medium, and high. The input and output linguistic variables are defined by their fuzzy values. The *Avl_Mem* of m^th^ PM is said to be low and high as defined in Eq. (44). Similarly, *Avl_CPU* and *Avl_Load* are defined in Eq. (45) and Eq. (46), respectively (Table 6).

$$if\ Avl_{Mem\,j}\ is \begin{cases} 0 \leq j \leq 5, & \text{Low} \\ 5.1 \leq j \leq 10, & \text{High} \end{cases} \tag{44}$$

$$if\ Avl\_CPU_j\ is \begin{cases} 0.5 \leq j \leq 5.2, & \text{Low} \\ 5.3 \leq j \leq 10, & \text{High} \end{cases} \tag{45}$$

Table 7 Simulation parameters for type 2 fuzzy system

Parameter	Value
Fuzzy type	Interval type 2 fuzzy
Input	Available memory in PM, available CPU in PM, and load in PM
Membership function type	Triangular with FoU
Linguistic term	Low, medium, and high
Defuzzification method	Centroid of sum
Type reduction method	Center of sets (using KM algorithms)
Representation	Vertical slices
Output	Possibility

$$if\ Avl\_Load_j\ is \begin{cases} 0.0 \le j \le 3.5, & \text{Low} \\ 3.6 \le j \le 10, & \text{High} \end{cases} \tag{46}$$

The number of rules in the rule base is set by the product of the maximum number of linguistic terms in each input fuzzy set, *i.e.,* $2 \times 2 \times 2 = 8$. The corresponding eight fuzzy rules using Eq. (36) are elaborated in the following section**.** R^1 is rule#1 which states: If available memory, available CPU, and available current load of jth PM are low, then there is a low possibility for this PM to be an optimal destination PM for VM migration. Similarly, other rules are defined in the same manner. It is shown that the highest possibility of being an optimal destination PM is achieved when the input set matches Rule#7.

The rules are stated as

R^1 : If *Avl_Mem* is Low and *Avl_CPU* is Low and *Avl_Load* is Low THEN *Possibility* is Low

R^2 : If *Avl_Mem* is Low and *Avl_CPU* is Low and *Avl_Load* is High THEN *Possibility* is Low

R^3 : If *Avl_Mem* is Low and *Avl_CPU* is High and *Avl_Load* is Low THEN *Possibility* is Medium

R^4 : If *Avl_Mem* is Low and *Avl_CPU* is High and *Avl_Load* is High THEN *Possibility* is Low

R^5 : If *Avl_Mem* is High and *Avl_CPU* is Low and *Avl_Load* is Low THEN *Possibility* is Medium

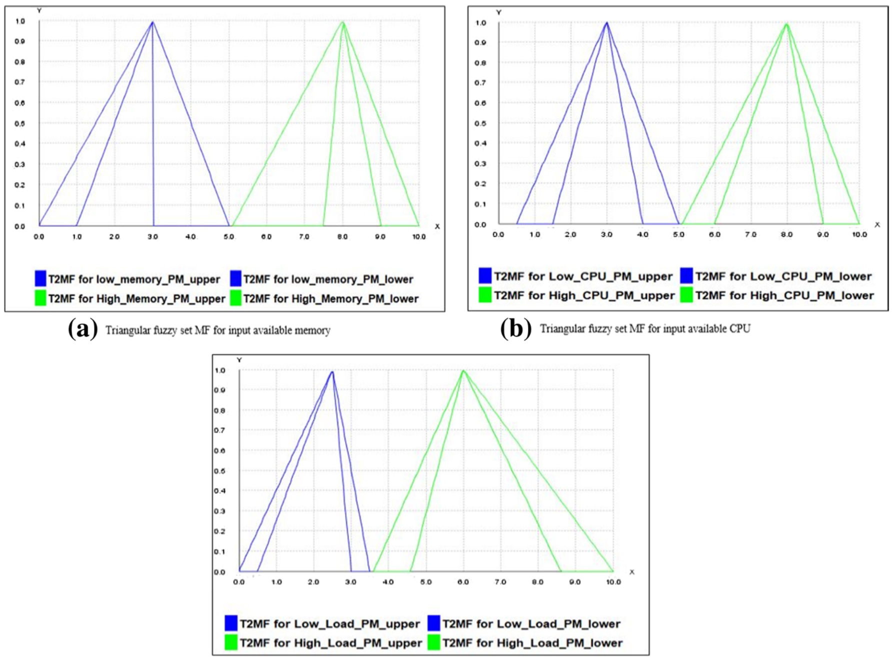
R^6 : If *Avl_Mem* is High and *Avl_CPU* is Low and *Avl_Load* is High THEN *Possibility* is Medium

R^7 : If *Avl_Mem* is High and *Avl_CPU* is High and *Avl_Load* is Low THEN *Possibility* is High

R^8 : If *Avl_Mem* is High and *Avl_CPU* is High and *Avl_Load* is High THEN *Possibility* is Medium

Each input domain consists of two IT2 FS. Figure 6 a–c shows the UMF and LMF of input fuzzy set *Avl_Mem, Avl_CPU*, and *Avl_Load* using vertical slice representation by Eq. (26). Triangular MF has been used for the input crisp set given in Eq. (34). The obtained MF graphs represent universe of disclosure on the x-axis that ranges from 0 to 10, and the degree of MF on the y-axis which ranges between 0 and 1. Table 8 shows the FoU for upper and lower MF for each input and output linguistic variable. Representation of Type 2 set can be achieved through vertical slice, horizontal slice, wavy slice, zSliced, etc. However, in this experiment, vertical slice representation is performed due to its simplicity and ease. The FoU L(MF) and FoU U(MF) are the footprints of uncertainty for lower and upper membership functions for each input and output linguistic variable. Table 8 depicts FoU L(MF) and FoU U(MF) for low PM memory having triangular MF values $\Delta(x : 1.0, 3.0, 3.0)$ and $\Delta(x : 0.0, 1.0, 5.0)$, respectively. Similarly, other values are obtained. A fuzzy set is fed into the inference model which combines fuzzy set and rule base. Based on rules provided in the rule base, IT2 FLS provides single output for multiple inputs.

To understand its working nature, let the value of the input vector $(x_1 = Avl_{Mem}, x_2 = Avl_{CPU}, x_3 = Avl\_Load)$ be (1.5, 3.8, 7.5). The input value of Avl_{Mem} is 1.5 which is low as given in Eq. (41), Avl_{CPU}is3.8 which represents low from Eq. (42) and $Avl_{Load}$is 7.5 which is high in range as per Eq. (43). The input value matches with rule number two (R^2 : If *Avl_Mem* is Low and *Avl_CPU* is Low and *Avl_Load* is High THEN *Possibility* is Low). Using Eq. (36), the obtained lower



(a) Triangular fuzzy set MF for input available memory

(b) Triangular fuzzy set MF for input available CPU

(c) Triangular fuzzy set MF for input PM load

Fig. 6 Member function (MF) for T1 FLS is shown in **a**, **b** and **c**

Table 8 FoU MF upper and lower bounds

S. No.	Low-memory PM	High-memory PM
FoU L(MF)	$\Delta(x : 1.0,3.0,3.0)$	$\Delta(x : 7.5,8.0,10.0)$
FoU U(MF)	$\Delta(x : 0.0,1.0,5.0)$	$\Delta(x : 5.1,8.0,10.0)$
S. No.	Low available CPU PM	High-memory PM
FoU L(MF)	$\Delta(x : 1.5,3.0,4.0)$	$\Delta(x : 6.0,8.0,9.0)$
FoU U(MF)	$\Delta(x : 0.5,3.0,5.0)$	$\Delta(x : 5.1,8.0,10.0)$
S. No.	Low available load PM	High-memory PM
FoU L(MF)	$\Delta(x : 0.5,2.5,3.0)$	$\Delta(x : 4.5,6.0,10)$
FoU U(MF)	$\Delta(x : 0.0,2.5,3.5)$	$\Delta(x : 3.6,6.0,10.0)$

and upper membership functions for Avl_{Mem},Avl_{CPU}andAvl_{Load} on each low and high linguistic variables are represented in Table 9. These bounds will be used in fuzzy inference rules for firing intervals IT2 FSs.

Table 9 FoU MF upper and lower bounds

Lower MF	Upper MF
$[\mu Low_L\left(Avl_{Mem} = 0.30\right)$	$\mu Low_U(Avl_{Mem} = 0.52)]$
$[\mu High_L\left(Avl_{Mem} = 0.70\right)$	$\mu High_U(Avl_{Mem} = 0.96)]$
$[\mu Low_L\left(Avl_{CPU} = 0.28\right)$	$\mu Low_U\left(Avl_{CPU} = 0.60\right)]$
$[\mu High_L\left(Avl_{CPU} = 0.84\right)$	$\mu High_U\left(Avl_{CPU} = 0.93\right)]$
$[\mu Low_L\left(Avl_{Load} = 0.50\right)$	$\mu Low_U\left(Avl_{Load} = 0.60\right)]$
$[\mu High_L\left(Avl_{Load} = 0.31\right)$	$\mu High_U\left(Avl_{Load} = 0.59\right)]$

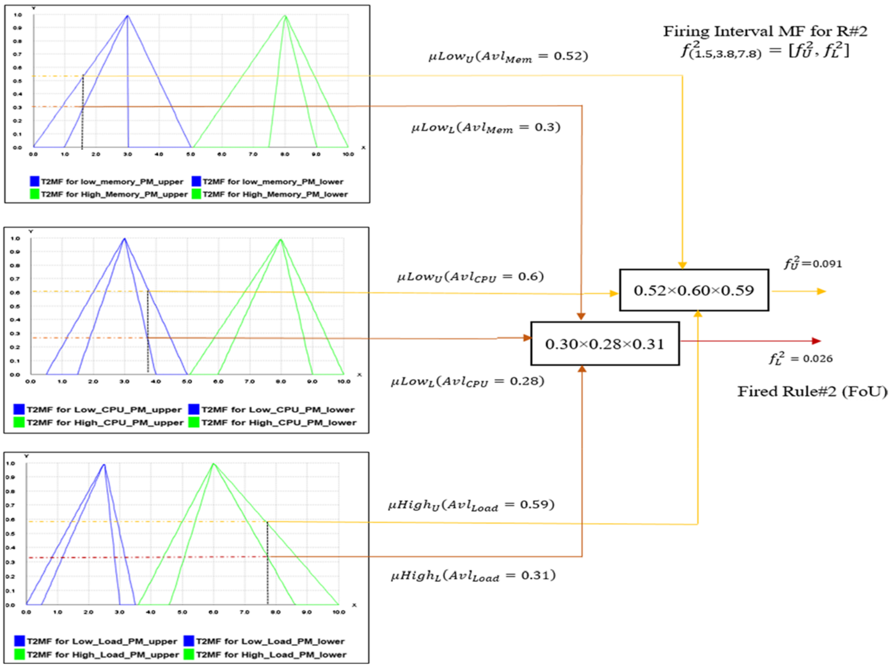

T2MF for low_memory_PM_upper
T2MF for low_memory_PM_lower
T2MF for High_Memory_PM_upper
T2MF for High_Memory_PM_lower
T2MF for Low_CPU_PM_upper
T2MF for Low_CPU_PM_lower
T2MF for High_CPU_PM_upper
T2MF for High_CPU_PM_lower
T2MF for Low_Load_PM_upper
T2MF for Low_Load_PM_lower
T2MF for High_Load_PM_upper
T2MF for High_Load_PM_lower
Firing Interval MF for R#2
$f^2_{(1.5,3.8,7.8)} = [f^2_U, f^2_L]$
$\mu Low_U(Avl_{Mem} = 0.52)$
$\mu Low_L(Avl_{Mem} = 0.3)$
$\mu Low_U(Avl_{CPU} = 0.6)$
$\mu Low_L(Avl_{CPU} = 0.28)$
$\mu High_U(Avl_{Load} = 0.59)$
$\mu High_L(Avl_{Load} = 0.31)$
0.52×0.60×0.59
0.30×0.28×0.31
f^2_U=0.091
f^2_L = 0.026
Fired Rule#2 (FoU)

Fig. 7 Calculation of firing interval MF for rule#2

Figure 7 explains the comparable calculation for firing intervαal to a given input. When x = 1.5, the vertical line at 1.5 intersects FoU (for low *Avl_Mem)* in the interval $\left[\mu Low_L\left(Avl_{Mem} = 0.30\right), \mu Low_U\left(Avl_{Mem} = 0.52\right)\right]$. For x = 3.8, the vertical line at 3.8 intersects FoU (for low *Avl_CPU)* in the interval $[\mu Low_L\left(Avl_{CPU} = 0.28\right), \mu Low_U\left(Avl_{CPU} = 0.60\right)]$, and for input x = 7.8, the vertical line at 7.8 intersects FoU (for low *Avl_Load)* in the interval $[\mu High_L\left(Avl_{Load} = 0.31\right), \mu High_U\left(Avl_{Load} = 0.59\right)$. From these inputs, two firing levels are then computed, *i.e.,* firing interval lower membership (f^m_L) and upper membership (f^m_U) for each rule using Eq. (35) and Eq. (36), respectively. The lower firing intervals for rule#1 are obtained by multiplying each L(MF) of three inputs

$(LowAvl_{Mem}, LowAvl_{CPU} and LowAvl_{Load})$. Similarly, upper firing intervals for rule#1 are obtained by multiplying each U(MF) of three inputs $(LowAvl_{Mem}, LowAvl_{CPU} and LowAvl_{Load})$ and hence $f^1_{(1.5,3.8,7.8)} = [f^1_L = 0.042, f^1_U = 0.187]$. The fired rule interval FoU for rule#2 is $f^2_{(1.5,3.8,7.8)} = [f^1_L = 0.091, f^2_U = 0.091]$. Similarly, each rule is performed in the same procedure to obtain rule FoU as given in Table 10. The centroid of the rules consequent is calculated using the iterative KM procedure [48]. For each fuzzy rule type 2 fuzzy set (FS) output, the centroid of the consequent is calculated for lower and upper interval $y^m_L$and y^m_U. The total number of output FS for output possibility is 3, *i.e.,* low, medium, and high. From Eq. (40), the obtained centroid of the low, medium, and high output FS is [0.5,3.45], [4.45,6.75], and [7.5,9.5], respectively. The centroid of each rule consequent for the output possibility is shown in Table 6. Type reduction is performed using KM algorithm, and we find left point (L) = 3 and right point (R) = 2. So, the value of $y^L(x)$ is obtained to perform the firing interval switching from upper bound to lower bound, whereas the value of $y^U(x)$ required to perform the switching firing interval from lower bound to upper bound using Eq. (41) and Eq. (42), respectively. It is needed to reorder $y^m_L$and y^m_U in an ascending manner where $y^1_L \le y^2_L \le y^3_L \le y^4_L \le y^5_L \le y^6_L \le y^7_L \le y^8_L$ and $y^1_U \le y^2_U \le y^3_U \le y^4_U \le y^5_U \le y^6_U \le y^7_U \le y^8_U$. Hence, reordered output possibility for lower bound is [0.5, 3.45, 3.45, 4.45, 4.45, 4.45, 6.75, 7.5] and for upper bound is [0.5, 0.5, 3.45, 6.75, 6.75, 6.75, 6.75, 9.50].

$$y^L = \left(f^1_U \times y^1_U\right) + \left(f^2_U \times y^2_U\right) + \left(f^3_U \times y^3_U\right) + \left(f^4_L \times y^4_L\right) + \left(f^5_L \times y^5_L\right) + \left(f^6_L \times y^6_L\right) + \left(f^7_L \times y^7_L\right) + \left(f^8_L \times y^8_L\right) / \left(f^1_U + f^2_U + f^3_U + f^4_L + f^5_L + f^6_L + f^7_L + f^8_L\right)$$

$$= \frac{(0.187 \times 0.5) + (0.091 \times 0.5) + (0.290 \times 3.45) + (0.078 \times 4.45) + (0.980 \times 4.45) + (0.061 \times 4.45) + (0.294 \times 6.75) + (0.182 \times 7.50)}{(0.187 + 0.091 + 0.290 + 0.078 + 0.980 + 0.061 + 0.294 + 0.182)}$$

$$y^L = \frac{9.74}{2.16} = 2.25$$

$$y^R = \left(f^1_L \times y^1_L\right) + \left(f^2_L \times y^2_L\right) + \left(f^3_U \times y^3_U\right) + \left(f^4_U \times y^4_U\right) + \left(f^5_U \times y^5_U\right) + \left(f^6_U \times y^6_U\right) + \left(f^7_U \times y^7_U\right) + \left(f^8_U \times y^8_U\right) / \left(f^1_L + f^2_L + f^3_U + f^4_U + f^5_U + f^6_U + f^7_U + f^8_U\right)$$

$$= \frac{(0.042 \times 0.5) + (0.026 \times 3.45) + (0.290 \times 3.45) + (0.285 \times 6.75) + (0.346 \times 6.75) + (0.340 \times 6.75) + (0.546 \times 6.75) + (0.537 \times 9.50)}{(0.042 + 0.026 + 0.290 + 0.285 + 0.346 + 0.340 + 0.546 + 0.537)}$$

$$y^R = \frac{16.38}{2.41} = 6.79$$

Defuzzification is needed to calculate the crisp output of possibility using Eq. (43). The possibility crisp output is the average summation of y^L and y^R. The final crisp output y^2 with respect to rule#2 is:

$$Y^2 = \frac{2.25 + 6.79}{2} = 4.52.$$

The obtained crisp output value suggests a cloud system that the PM with its input values for rule# maintains a low possibility for destination optimal PM for VM migration. It clearly shows that the output value 4.52 has a low possibility for VM migration. Figure 8 shows a graphical representation of triangular MF for output possibility using Eq. (31). The x-axis shows the possibility with respect to the input fuzzy set, and the y-axis shows the DoM between 0 and 1. The *Low*, *Medium*, and

Table 10 Firing Interval of the eight rules

Rule#	Firing interval	Rule consequent
R^1 :	$[f_L^1, f_U^1] =$ $\left[\mu Low_L(Avl_{Mem}) \times \mu Low_L(Avl_{CPU}) \times \mu Low_L(Avl_{Load}), \mu Low_U(Avl_{Mem}) \times \mu Low_U(Avl_{CPU}) \times \mu Low_U(Avl_{Load})\right]$ $[0.30 \times 0.28 \times 0.5, 0.52 \times 0.60 \times 0.60] = [0.042, 0.187$	[0.5, 3.45]
R^2 :	$[f_L^2, f_U^2] =$ $\left[\mu Low_L(Avl_{Mem}) \times \mu Low_L(Avl_{CPU}) \times \mu High(Avl_{Load}), \mu Low_U(Avl_{Mem}) \times \mu Low_U(Avl_{CPU}) \times \mu High_U(Avl_{Load})\right]$ $[0.30 \times 0.28 \times 0.31, 0.52 \times 0.60 \times 0.29] = [0.026, 0.091]$	[0.5 ,3.45]
R^3 :	$[f_L^3, f_U^3] =$ $\left[\mu Low_L(Avl_{Mem}) \times \mu High_L(Avl_{CPU}) \times \mu Low_L(Avl_{Load}), \mu Low_U(Avl_{Mem}) \times \mu High_U(Avl_{CPU}) \times \mu Low_U(Avl_{Load})\right]$ $[0.30 \times 0.84 \times 0.5, 0.52 \times 0.93 \times 0.60] = [0.014, 0.290]$	[0.45 ,6.75]
R^4 :	$[f_L^4, f_U^4] =$ $\left[\mu Low_L(Avl_{Mem}) \times \mu High_L(Avl_{CPU}) \times \mu High_L(Avl_{Load}), \mu Low_U(Avl_{Mem}) \times \mu High_U(Avl_{CPU}) \times \mu High_U(Avl_{Load})\right]$ $[0.30 \times 0.84 \times 0.31, 0.52 \times 0.93 \times 0.59] = [0.078, 0.0.285]$	[0.5, 3.45]
R^5 :	$[f_L^5, f_U^5] =$ $\left[\mu High_L(Avl_{Mem}) \times \mu Low_L(Avl_{CPU}) \times \mu Low_L(Avl_{Load}), \mu High_U(Avl_{Mem}) \times \mu Low_U(Avl_{CPU}) \times \mu Low_U(Avl_{Load})\right]$ $[0.70 \times 0.28 \times 0.5, 0.96 \times 0.60 \times 0.60] = [0.98, 0.346]$	[0.45, 6.75]
R^6 :	$[f_L^6, f_U^6] =$ $\left[\mu High_L(Avl_{Mem}) \times \mu Low_L(Avl_{CPU}) \times \mu High_L(Avl_{Load}), \mu High_U(Avl_{Mem}) \times \mu Low_U(Avl_{CPU}) \times \mu High_U(Avl_{Load})\right]$ $[0.70 \times 0.28 \times 0.31, 0.96 \times 0.60 \times 0.59] = [0.061, 0.340]$	[0.45, 6.75]
R^7 :	$[f_L^7, f_U^7] =$ $\left[\mu High_L(Avl_{Mem}) \times \mu High_L(Avl_{CPU}) \times \mu Low_L(Avl_{Load}), \mu High_U(Avl_{Mem}) \times \mu High_U(Avl_{CPU}) \times \mu Low_U(Avl_{Load})\right]$ $[0.70 \times 0.84 \times 0.5, 0.96 \times 0.93 \times 0.60] = [0.294, 0.546]$	[7.5, 9.5]

Table 10 (continued)

Rule#	Firing interval	Rule consequent
R^8 :	$\left[f_L^8, f_U^8\right] =$ $\left[\mu High_L\left(Avl_{Mem}\right) \times \mu High_L\left(Avl_{CPU}\right) \times \mu High_L\left(Avl_{Load}\right), \mu High_U\left(Avl_{Mem}\right) \times \mu High_U\left(Avl_{CPU}\right) \times \mu High_U\left(Avl_{Load}\right)\right]$ $[0.70 \times 0.84 \times 0.31, 0.96 \times 0.93 \times 0.59] = [0.182, 0.537]$	[0.45, 6.75]

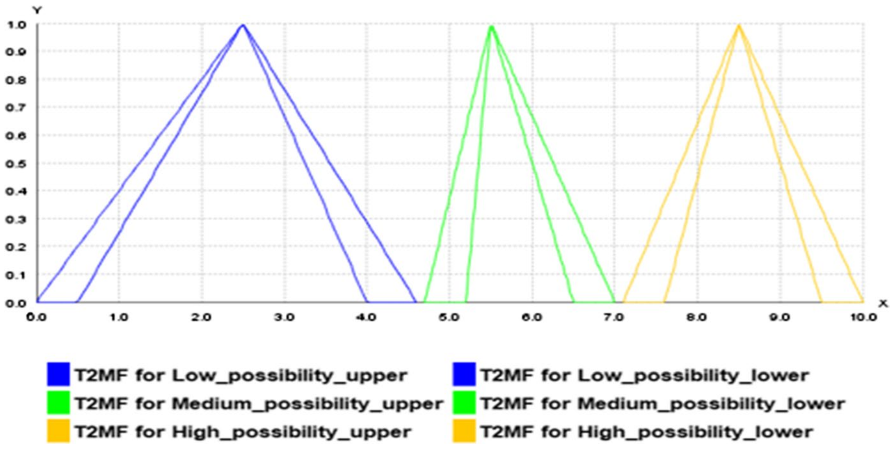


Fig. 8 Member function (MF) for otput possibility

Table 11 FoU MF upper and lower bounds

	Low PM Possibility	Medium PM Possibility	High PM Possibility
FoU L(MF)	$\Delta(y : 0.5, 2.5, 4.0)$	$\Delta(y : 4.7, 5.5, 7.0)$	$\Delta(y : 7.1, 8.5, 9.5)$
FoU U(MF)	$\Delta(y : 0.0, 2.5, 4.6)$	$\Delta(y : 4.7, 5.5.5, 6.5)$	$\Delta(y : 7.6, 8.5, 10)$

High linguistic variables for possibility are obtained by using Eq. (47). *Low* possibility is considered when the value of y is between 0 and 4.6. Similarly, *Medium* and *High* are obtained using Eq. (47). Table 11 depicts the lower and upper MF bounds for output possibility for its linguistic variables.

$$\text{if Possibility}_{PM_j} \text{ is} \begin{cases} 0 \le Y \le 4.6, & \text{Low} \\ 4.7 \le Y \le 7.0, & \text{Medium} \\ 7.1 \le Y \le 10, & \text{High} \end{cases} \tag{47}$$

Similarly, to check the performance of IT2FLS with other rule bases similar mathematical steps have been performed for each set of rules by the obtained values of Mth PM properties. Table 12 shows the overall obtained rule# with their respective input values of three sets (memory, CPU, and load) and their respective output (possibility). It is explained that rule#1 has an output possibility of 1.3 which results in a low possibility for PM_{des} for VM migration from Eq. (47). Similarly, rule#2 and rule#4 have obtained low possibilities with values 4.52 and 4.23. Rule# 3, 5, 6, and 8 have achieved 4.98, 5.91, 6.84, and 7.0 crisp output values for medium possibility, respectively. The selection of PM_{des} for migration is to be made only when PM has high available memory, high available CPU, and low load. Finally, Rule#8 with its crisp output 8.99 has obtained high possibility to become optimal PM_{des} for VM migration.

The algorithm checks for VM_U in PM once PM_{des} is identified. If any PM has VM_U, then that VM_U will be turned off after migration in order to save energy. Hence,

Table 12 Crisp input and obtained crisp output after defuzzification

Rule	*Avl_Mem*(x_1)	*Input*		*Output*
		Avl_CPU(x_2)	*Avl_Load* (x_3)	*Possibility(O)*
R^1 :	1.0	2	3	1.3
R^2 :	1.5	3.8	7.8	4.52
R^3 :	1.9	7	5.8	4.98
R^4 :	2.5	8	6.2	4.23
R^5 :	7.2	1	4.2	5.91
R^6 :	6.1	1	6	6.84
R^7 :	8	8	2	8.99
R^8 :	7	6.1	5.1	7.0

this VM migration process results in load balancing among PMs along with energy efficiency. The complete working steps of VM migration-based IT2FLS algorithm are given in *Algorithm 3*.

Algorithm:3 IT2FL for optimal PM and VM migration (Stage III)

Crisp Input : *Avl_Mem, Avl_CPU, Avl_Load*
Crisp Output: *Possibility* (optimal PM_{des} for migration)

```
1.Compute load on PM using Eq.(23)
2:If anyone of PM overloaded($PM_{over}$)
3.for each VMs in $PM_{over}$
4:Compute MT(VM) using Eq.(24)
5:Select $V_{op}$ that satisfies Eq.(25)
6:Initialize IT2F logic algorithm
7:Fuzzifier←Crisp Input (Fuzzy input Set)
8: for each PM
9: Calculate,
        Avail_Mem_PM=broker.getVmList().get(i).getHost().getStorage();
        Avail_CPU_PM=broker.getVmList().get(i).getMips();
        Load_PM=broker.getVmList().get(i).getCloudletScheduler().runningCloudlets();
10: Set up the lower and upper membership functions making up the overall IT2F sets for each input and output using Eq.(36).
11: Set up the antecedents and consequents to associates inputs.
12: Set up the rulebase and perform lower and upper firing interval using Eq.(37) and Eq.(38).
13: Inference ←Rules and Fuzzifier
        rulebase.addRule(new IT2_Rule(new IT2_Antecedent[]{LowMem, LowCPU,lowLoad}, lowPossibility));
        rulebase.addRule(new IT2_Rule(new IT2_Antecedent[]{LowMem, LowCPU,highLoad}, lowPossibility));
        rulebase.addRule(new IT2_Rule(new IT2_Antecedent[]{LowMem, HighCPU,lowLoad}, mediumPossibility));
        rulebase.addRule(new IT2_Rule(new IT2_Antecedent[]{LowMem, HighCPU,highLoad}, lowPossibility));
        rulebase.addRule(new IT2_Rule(new IT2_Antecedent[]{HighMem, LowCPU,lowLoad}, mediumPossibility));
        rulebase.addRule(new IT2_Rule(new IT2_Antecedent[]{HighMem, LowCPU,highLoad}, mediumPossibility));
        rulebase.addRule(new IT2_Rule(new IT2_Antecedent[]{HighMem, HighCPU,lowLoad}, highpossibility));
        rulebase.addRule(new IT2_Rule(new IT2_Antecedent[]{HighMem, HighCPU,highLoad}, mediumPossibility));
14: Rule_Combination_Output_Processing←IT2_Fuzzy_Sets (from) inference
15: for each IT2_Fuzzy_Sets Perform Type Reduction using Eq.(40).
17: Evaluate centre of sets type reduction to compute the left and right most points using Eq.(41) and Eq.(42).
        Avail_Memory_PM.setInput(Avail_Memory);
        Avail_CPU_PM.setInput(Avail_CPU);
        Load_PM.setInput(Load);
18:Perform Defuzzification Centroid method using Eq.(43)
19:Obtain non-fuzzy crisp output values
20: If output possibility is high then select PM as optimal $PM_{des}$ for migration
21: Move $PM_{des}$ ← $V_{op}$
22:If anyone of PM has $VM_U$
23:Goto step3
24:Turn off $VM_U$ in $PM_{under}$
25:end for
26:end for
27:end for
28:until destination PM found or maximum iteration reached
29:return destination PM for VM migration or feasible solution
```

4.5 Computational complexity

The proposed CMODLB algorithm is a hybridization of ANN-LB, TOPSIS-PSO, and VM migration using IT2FLS technique, which may get high computational complexity. The complexity of the complete algorithm is calculated for each working stage. The ANN-based backpropagation neural method takes $O(n^4)$ complexity for feedforward and $O(n^5)$ for backpropagation where 'n' is the number of inputs (VMs). VM load is calculated with the complexity of $O(n^5 + n^4)$ [41]. The computational complexity of BOK-means is of $O(N^2)$, where N is the number of iterations [42] for Bayesian optimization. K-mean clustering algorithm takes the complexity of $O(n)$ for computing distance between items. The reassigning cluster has the complexity of $O(km)$. For centroid, it takes the complexity of $O(mn)$; each of the above steps takes t iterations with $O(tkmn)$ complexity, where 'k' is the number of clusters, 'm' is the objects, and 'n' is the dimensionality of the vector. In PSO-TOPSIS algorithm, PSO has the complexity of $O(ntlog(n))$ in which 'n' is the number of populations and t is the number of iterations. Besides this, the complexity of the TOPSIS algorithm is $O(n^2 + n + 1)$ for FV computation [35]. $(4p + 1)M$ is the required design degree of freedom for IT2FL [43] where p is the number of antecedents corresponding to each M rule.

5 Experimental evaluation

This section explains the experimental evaluation of the proposed CMODLB load balancing method in a cloud system environment. This section also comprises subsections. In the simulation environment, the details about the proposed cloud environment are provided, while in the performance matrices, each significant matric is explained. Comparison of the CMODLB method with existing task scheduling and load balancing methods is shown in the comparative analysis section. Our proposed work is compared with other existing algorithms CESCC strategy [8], WRR method [13], DHCI [14], two-level mechanism [17], TaPRA method [18], HBB-LB [22], BSO algorithm [24], utilization model [29], TOPSIS-PSO [35], SA, GA, GA-SA, GA-GELS, PSO, PSO-SA and PSO-GELS [36] and FUGE [45] for its performance analysis.

Table13 explores the shortcomings that occurred in former scheduling and load balancing techniques. These shortcomings are taken as essential features for the experimental evaluation of the CMODLB proposed method.

5.1 Simulation setup

The proposed CMODLB in cloud environment uses JAVA (JDK 1.7) including Java runtime environment, Java class libraries, and Java tools. NetBeans 7.4 and eclipse are used for simulation. The essential classes for DCs, VMs, and computational resources are provided by the CloudSim tool [44], which supports modeling

Table 13 Limitations of previous works

Existing method	Demerits
Two-level mechanism [17]	Supports only a single server for task scheduling
	Increases waiting time for user tasks
TaPRA method [18]	Only suitable for the single-task assignment problem
BSO algorithm [24]	Load balancing among PMs is not achieved
Utilization model [29]	Increases number of VM migrations
TOPSIS-PSO [35]	Load balancing is required

and replication of datacenters, virtualized server hosts, energy-aware computational resources, federated cloud, etc., of large-scale environment.

In Table 14, significant parameters considered in the simulation of CMODLB are depicted with their corresponding values. The values of parameters are categorized into physical machines, virtual machines, and tasks. Two to six numbers of PM having up to four processing units have been used with 4 GB RAM and 11 TB storage capacity with a maximum of 9600 MIPS. Around 10–50 numbers of VMs are used having a MIPS of up to 2400.

5.2 Performance metrics

This section discusses the significant performance metrics which are considered in the experimental evaluation. Definition of each metric is provided along with its importance. Significant metrics considered in this section are: completion time, transmission time, MakeSpan, number of VM migrations, resource utilization, and system load fairness.

5.2.1 Completion time

Completion time for a task that is assigned to a VM is a metric which is the summation of execution time (E_T), transmission time (T_T), and waiting time (W_T) of Sth task on Lth VM and is expressed as follows:

$$Completion\ time = E_T + T_T + W_T. \tag{48}$$

If the load on a particular VM is more, then the tasks assigned to it suffer from high waiting time, leading to large completion time and overall high MakeSpan. Therefore, this metric should be as low as possible.

5.2.2 Transmission time

The purpose of these metrics is to calculate the transmission time, a task needs to reach a particular VM. Transmission time is the proportion of Sth task size by the bandwidth of Lth VM [35]. T_T must be less to achieve better resource utilization and MakeSpan.

Table 14 Simulation parameters for cloud environment

	Parameter	Value
Physical machine	Number of PMs	2–6
	Number of processing units in one PM	4
	MIPS	9600
	Storage capacity	11 TB
	RAM	4 GB
	Scheduling interval	30 ms
	Monitoring interval	180 ms
Virtual machine	Number of VMs in each PM	10–50
	MIPS	2400
	Number of processing units	4
Task	Number of tasks	50–2000
	Maximum task length	20,000
	Task size	500
	MIPS required	10,000
	Average RAM	512 MB
	Average bandwidth	100,000 Mbps

$$T_T = {}^{Size\ of\ task^S}\!\big/_{Bandwidth\ of\ VM^L}. \tag{49}$$

5.2.3 MakeSpan

MakeSpan is the total amount of time required by a VM to execute the assigned tasks [35]. It is also defined as the time difference between completion time of last job (C_{last}) and starting time of first job (S_{first}) and is expressed as,

$$Make\ span = \left(C_{last} - S_{first}\right). \tag{50}$$

This metric should be as low as possible. MakeSpan can be minimized by assigning optimal VM to user task, which can be achieved by task scheduling. Balanced VMs achieve minimum MakeSpan. It is also defined as the sum of completion time of tasks assigned on a particular VM and is given as

$$Makespan = \sum_{P=1}^{R} Completion\ time_p.$$

Here, R represents the number of tasks assigned to VM, whereas completion $time_p$ represents the completion time of pth task on that VM.

5.2.4 Number of VM migrations

Since VM migration consumes energy, CPU, bandwidth, and time, it is necessary to control the number of migrations performed in the system. This metric provides the number of migrations carried out in the system. VM migration takes place frequently if the system is unbalanced. Hence, this metric plays a vital role in the evaluation of load balancing.

5.2.5 Resource utilization

In a cloud infrastructure, resources are pooled to serve multiple consumers simultaneously. It is important to utilize these resources efficiently. An efficient task assignment process can achieve this by assigning each task to the optimal underloaded VM. Resource utilization can be calculated in terms of CPU utilization, memory utilization, and bandwidth utilization.

5.2.6 Load fairness

To analyze the performance of the proposed method having a large system load. The metric provides load fairness using its system load. The calculation of load fairness depends on the completion time of each task. It has been included that if a cloud system has a better completion time of tasks, then it has the potential to achieve higher load fairness. This analysis will predict the efficiency of system load with respect to the number of user tasks. System load fairness LF is given as [17],

$$LF = \left(\sum_{t=1}^{N} CT_t\right)^2 \Big/ N \sum_{t=1}^{N} \left(CT_t\right)^2 \tag{51}$$

where CT_t is the completion time of t tasks and N represents the total number of tasks.

5.3 Comparative analysis

In this section, the proposed CMODLB method has been compared with some of the state-of-the-art methods.

5.3.1 Analysis of completion time

This metric estimates the total execution time that comprises the waiting time of each task given in Eq. (48). The metric value should be low for an efficient cloud system. Figure 9 shows the comparative analysis among TaPRA method, BSO algorithm, and the proposed CMODLB method in terms of completion time. Here, completion time increases as the number of tasks increases. It is quite clear that BSO algorithm has maximum completion time as compared to TaPRA and CMODLB algorithm. This is for an imbalanced load on PMs in DCs. Hence, a large number of tasks cannot be accomplished in PM. In TaPRA method, completion time is slightly increased with the increase in the set of tasks. It completes 2000 tasks in approximately 42 s which the CMODLB does in 30 s. The average completion time of TaPRA method is 20.6 s, whereas the average completion time of the CMODLB method is 14.2 s. But BSO algorithm provides the average completion time of around 50 s which is higher than TaPRA method and the CMODLB method. Since TaPRA method allows a single-task assignment in each step, it takes more time than the CMODLB method. In TaPRA method, task assignment process is not able to consider execution time in order to minimize completion time. Clustering of VMs approach reduces the complexity of finding appropriate VM for the execution of tasks with respect to its current load. If the VM having less load executes the task, then the waiting time of tasks on that VM will be very less. This approach offers the lowest execution and transmission time. Due to the involvement of efficient load balancing and task assignment processes, the CMODLB method achieves 31.067% and 71.6% less completion time compared to TaPRA and BSO, respectively, as shown in Table 15. Therefore, load balancing-based CMODLB method reduces task completion time significantly.

5.3.2 Analysis of transmission time

The transmission time metric evaluates the actual transfer time of a task to reach the assigned VM. It includes the size of tasks and bandwidth of a VM as given in Eq. (49). For a well-organized cloud system environment, this metric must remain minimum. Figure 10 shows the comparative analysis among the PSO method, TOPSIS-PSO algorithm, and proposed CMODLB method in terms of transmission time. The experimental analysis is performed for 10 to 40 numbers of tasks having 10 numbers of VMs. Here, transmission time increases the number of tasks. It is noticeable that the PSO algorithm suffers from high transmission time compared to TOPSIS-PSO and the CMODLB method because the task scheduling process in PSO takes more time than TOSIS-PSO and CMODLB method. In PSO method, the transmission time slightly increases with the increase in the number of tasks. It gives nearly 0.711 s for 10 tasks, while TOPSIS-PSO takes 0.644 s and the CMODLB method achieves 0.60 s for the same. The average transmission time of PSO and TOPSIS-PSO method is 1.39 and 1.32 s, respectively, whereas the average transmission time of the CMODLB method is 1.30 s. The PSO and TOPSIS-PSO algorithms provide average transmission time around 0.09 and 0.02 s, respectively, which are higher than the CMODLB method. Due to efficient load balancing and

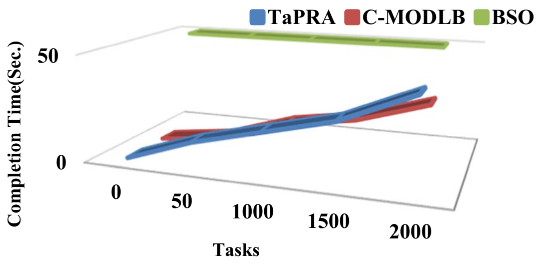


Fig. 9 Comparative performance analysis on completion time

Table 15 Comparison of completion time (in seconds) for CMODLB

#Tasks	TaPRA	BSO	CMODLB
0	1	50	1
50	12	50	5
1000	20	50	15
1500	28	50	20
2000	42	50	30

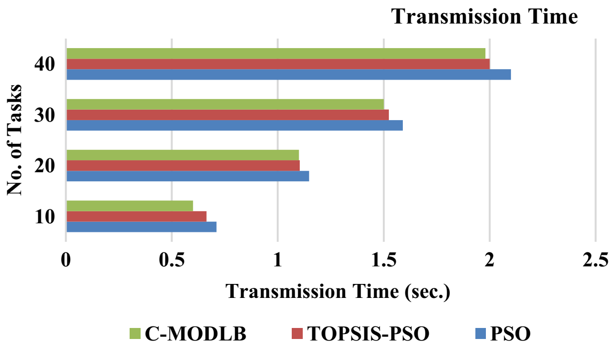


Fig. 10 Comparative performance analysis on transmission time

task assignment processes, the CMODLB takes 6.65% and 2.12% less transmission time than PSO and TOPSIS-PSO, respectively, as shown in Table 16.

5.3.3 Analysis of MakeSpan

This metric gives the overall completion time of all the tasks in VMs. The metric results are low when the load balancing method is well formulated. The calculation for MakeSpan is performed using Eq. (50). The performance of the proposed work is analyzed through three sets of tests in terms of MakeSpan:

(1) Test I: Experiment in Test I is performed for 15–60 numbers of tasks in 10 numbers of VMs to analyze the performance of the CMODLB with respect to static algorithms such as MaxMin and Round Robin.
(2) Test II: 100–300 numbers of tasks are simulated for Test II. Results are compared with various dynamic algorithms such as FUGE [45], ACO, MACO, and TOPSIS-PSO [35] with the similar cloud setup configuration having 1000–20,000 task length, 50 numbers of VMs with 500–1000 VM bandwidth, 256–2048 VM memory (RAM), and 10 numbers of data centers with 2–6 numbers of PM.
(3) Test III: To analyze the distributed nature of the proposed CMODLB algorithm, some grid computing-based algorithms [36] like SA, GA, GA-SA, GA-GELS, PSO, PSO-SA, and PSO-GELS are considered for performance evaluation. The experimental parameter values and simulation platform are kept similar for the proposed and existing algorithms for Test III.

5.3.3.1 Test I (MaxMin vs. R.R vs. CMODLB) In the Test I experiment, the proposed CMODLB algorithm is compared with MaxMin and R.R algorithm for MakeSpan metric for 10 numbers of VMs and 15–60 numbers of tasks. Table 17 represents the obtained results for 15, 30, 45, and 60 tasks for MaxMin, R.R, and the CMODLB, respectively. Figure 11 shows that the proposed algorithm achieves 65.54% and 68.26% lesser MakeSpan than MaxMin and R.R, respectively. This analysis shows that the proposed CMODLB algorithm provides better performance in a cloud environment.

5.3.3.2 Test II: (TOPSIS-PSO vs. FUGE vs. ACO vs. MACO vs. CMODLB) Experiment is performed to check the performance of the proposed CMODLB method with respect to MakeSpan for dynamic nature-based algorithms. The comparative analysis is done with existing algorithms [35, 45] for 100, 200, and 300 tasks. Table 18 depicts the obtained values from the simulation. Algorithm FUGE and TOPSIS-PSO have performed better than ACO and MACO, whereas CMODLB performed better among all the algorithms, as shown in Fig. 12.

5.3.3.3 Test III: (SA vs. GA vs. GA-S vs. GA-GELS vs. PSO vs. PSO-SA vs. PSO-GELS vs. CMODLB The comparison of different algorithms is analyzed with similar simulation configurations. The MakeSpan for the proposed CMODLB and existing algorithms [36], *i.e.,* SA, GA, GA-SA, GA-GELS, PSO, PSO-SA, and PSO-GELS, is depicted in Tables 19, 20, 21, and 22 for 100, 300, 500, and 1000 iterations, respectively, on 50, 100, 300, and 500 number of tasks having 10 numbers of resources. From Fig. 13a–d, it may be seen that MakeSpan for the CMODLB method decreases as there is an increase in the number of tasks as compared to SA, GA, GA-SA, GA-GELS, PSO, PSO-SA, and PSO-GELS algorithms. Figure 13 clearly shows that the proposed CMODLB method is highly efficient compared to PSO-GELS and other methods.

The involvement of efficient load balancing and task scheduling provides lower completion time and also helps in minimizing MakeSpan. The analysis shows that the proposed CMODLB algorithm is performing better in a cloud environment.

5.3.4 Comparative analysis on number of VM migrations

Load balancing among PMs is carried out by VM migrations, which consumes energy and time due to migration. Hence, it is necessary to minimize the number of migrations in the system. Figure 14 demonstrates the comparative analysis of the number of VM migrations of the CMODLB method with various other existing

Table 16 Comparison of transmission time (in seconds) for CMODLB

#Tasks	PSO	TOPSIS-PSO	CMODLB
10	0.711	0.664	0.60
20	1.148	1.104	1.10
30	1.590	1.524	1.50
40	2.100	2.00	1.98

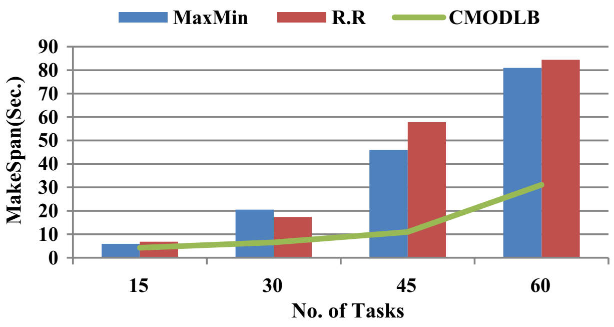


Fig. 11 Comparative performance analysis for Test I

methods, viz. HBB-LB, WRR, and utilization. The CMODLB method took only one VM migration, whereas the utilization method took 25 VM migrations. Since VMs are balanced in the CMODLB method, there is a smaller number of VM migration leading to less PM overloading. Hence, in the CMODLB method, the number of VM migrations is low.

5.3.5 Analysis of resource utilization

In this section, the resource utilization performance of the CMODLB method is compared with DHCI and CESSC methods. Maximum utilization of resources indicates minimum wastage of cloud system resources. The analysis of resource utilization also illustrates the effectiveness of the cloud system to utilize bandwidth, energy, CPU, and memory.

Figure 15 demonstrates the comparative study on resource utilization with respect to time. From Fig. 15, it is clear that the wastage of resources in the CMODLB algorithm is less that indicates utilization of resources is maximum due to the efficient task assignment process. CMODLB method provides resource utilization above 50%, while the CESCC method is able to provide resource utilization between 28

Table 17 Comparison of MakeSpan (in seconds) for Test I

#Tasks	MaxMin	R.R	CMODLB
15	5.89	6.8	4.25
30	20.5	17.4	6.52
45	45.96	57.8	10.97
60	80.93	84.4	31.07

Table 18 Comparison of MakeSpan (in seconds) for Test II

#Tasks	FUGE	ACO	MACO	TOPSIS-PSO	CMODLB
100	40	70	60	37.14	36.44
200	90	120	110	85.79	83.35
300	155	190	170	150.59	146.17

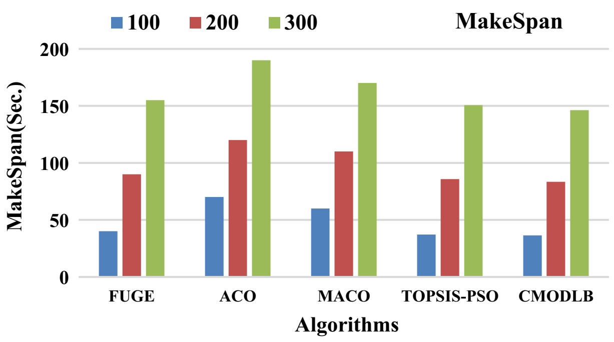


Fig. 12 Comparative MakeSpan analysis for Test II

Table 19 CMODLB MakeSpan value for 100 iterations

ITERATIONS 100								
TASKS	SA	GA	GA-SA	GA-GELS	PSO	PSO-SA	PSO-GELS	CMODLB
50	136.742	99.198	95.562	90	86.144	89.586	85.186	74.77
100	307.738	183.49	190.353	181.028	168.718	167.33	166.094	121.46
300	973.728	638.082	597.8	581.842	528.568	511.532	494.66	309.71
500	1837.662	1105.56	1072.362	1087.216	918.336	887.195	911.099	758.4

Table 20 CMODLB MakeSpan value for 300 iterations

ITERATIONS 300								
TASKS	SA	GA	GA-SA	GA-GELS	PSO	PSO-SA	PSO-GELS	CMODLB
50	131.12	89.486	86.98	84.298	85.312	87.684	84.174	60.83
100	233.2	172.628	179.062	175.598	170.452	167.16	166.422	102.38
300	911.68	570.466	532.968	600.862	518.268	526.71	527.086	367.72
500	1492.616	1071.014	1037.942	1055.504	890.354	896.586	881.991	614.2

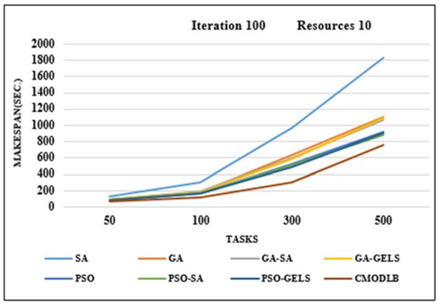
Table 21 CMODLB MakeSpan value for 500 iterations

ITERATION 500							
TASKS	GA	GA-SA	GA-GELS	PSO	PSO-SA	PSO-GELS	CMODLB
50	117.994	85.264	84.614	83.362	85.22	83.774	51.29
100	261.664	175.084	171.817	168.714	176.688	170.732	102.38
300	855.896	863.55	521.302	552.828	506.026	509.04	309.71
500	1640.528	993.964	1000.561	954.652	849.066	894.495	758.4

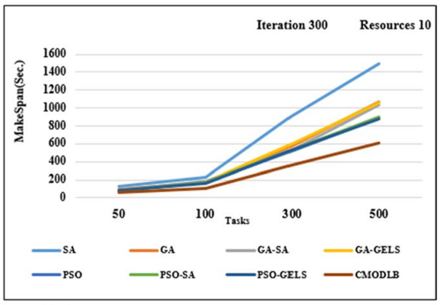
Table 22 CMODLB MakeSpan value for 1000 iterations

ITERATION 1000

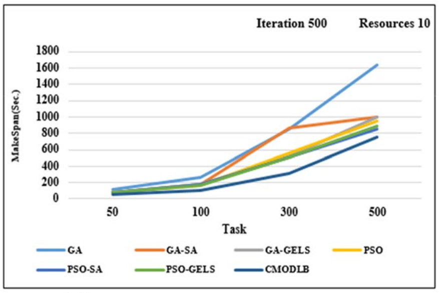
TASKS	SA	GA	GA-SA	GA-GELS	PSO	PSO-SA	PSO-GELS	CMODLB
50	108.151	82.566	84.856	84.402	87.436	86.106	85.72	60.83
100	223.158	167.755	160.84	160.694	170.716	174.648	169.784	102.38
300	867.48	545.146	542.52	533.336	520.953	511.33	421.388	309.71
500	1495.312	933.798	935.794	932.23	856.9	888.534	851.614	614.2



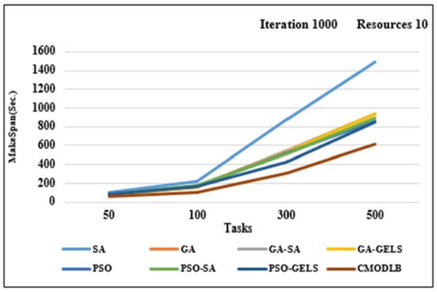
(a) Comparative analysis on MakeSpan for 100 iteration



(b) Comparative analysis on MakeSpan for 300 iteration



(c) Comparative analysis on MakeSpan for 500 iteration



(d) Comparative analysis on MakeSpan for 1000 iteration

Fig. 13 Comparative analysis on MakeSpan for various iterations

and 50% for 50 min. The resource utilization of the DHCI method oscillates between 30 and 60%. The CMODLB method achieves 75% utilization, which is much effective than DHCI and CESCC. Load balancing, scheduling of tasks, and migration of VMs are always concerned with resources; hence, it is clear that implementing the CMODLB method will achieve better resource utilization and reduce wastage of resources.

5.3.6 Load fairness

This metric measures the fairness of the system with respect to load and is calculated by using Eq. (51). This metric should be as high as possible to achieve better performance [17]. In Fig. 16, system load is taken on X-axis, while the fairness of the system is taken on Y-axis. Here, fairness signifies the presentation of the proposed method in system load. The proposed CMODLB method is compared with the existing two-level method. The two-level method provides system fairness about 1 as constant for different system loads. The task scheduling process is not effective in the two-level method, and hence it has constant load fairness for different system loads, which makes it less efficient in heavy-load systems. However, the CMODLB method attains better performance with a heavy system load. It offers up to 1.12 system performance for 100% system load. It shows that the proposed method is reliable for heavy system load. Hence, the performance of load balancing is improved in the CMODLB method compared to the existing load balancing methods.

6 Conclusion

In this paper, to resolve load balancing issues in both VMs and PMs, a novel hybrid clustering, multi-criteria and VM migration-based approach (CMODLB) is proposed. To achieve our objectives, the cloud environment is designed with three entities: VM clustering using a load balancer and VM manager, the TOPSIS-PSO method for efficient task scheduling, and IT2FL for selection of optimal PM for VM migration. The first two entities approached load balancing at VM level, whereas the third entity maintains PM-level load balance. VM manager groups the VMs into underloaded and overloaded VMs, and balancer manages the clusters in order to preserve the uniqueness. For this purpose, BOEK-means with ANN algorithm is used. Task scheduling process allocates tasks to optimal underloaded VM using multi-objective-based existing TOPSIS-PSO algorithm. Optimal VM for task assignment is selected based on significant metrics such as execution time, transmission time, and CPU utilization.

The above two processes balance load among VMs, while VM migration intends to balance load across the PMs. VM migration aims to minimize load and energy

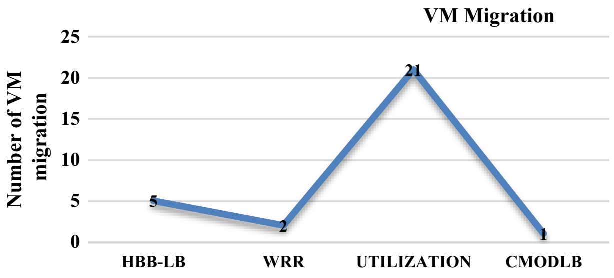


Fig. 14 Comparative performance analysis on number of migrations

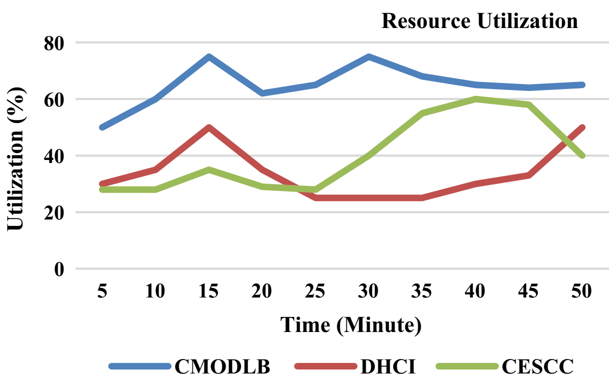


Fig. 15 Comparative performance analysis on resource utilization

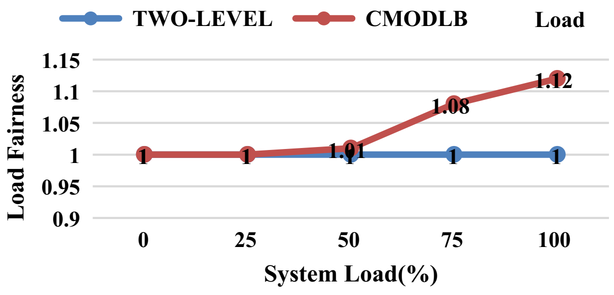


Fig. 16 Comparative performance analysis on load

consumption on PMs. Soft computing-based IT2F logic is incorporated to select optimal destination PM for VM migration to maintain PM load and energy-efficient cloud. The obtained experimental results show that the proposed CMODLB load balancing technique manages improved load balancing along with effective completion time, transmission time, MakeSpan, resource utilization, and load fairness. This IT2F logic method for calculating optimal destination PM for VM migration is novel and remarkable. In the future, we aim to cover load balancing with various machine learning tools and methods to improve energy efficiency which is lacking in the current proposed model. This will include storage intensive tasks and storage IOPS/transfer-based VM capacity in real-time cloud environment.

Acknowledgment The first author (Sarita Negi) acknowledges Prof. Man Mohan Singh Ruthann, Dr. Rohit Mahar, and the Department of Computer Science and Engineering, H N B Garhwal University (Srinagar Garhwal), Uttarakhand, for their immense support and resources.

References

1. Sadiku NOM, Musa M, S, D Momoh, O, (2014) Cloud computing: Opportunities and challenges. IEEE Potentials 3(1):34–36
2. Diaz M, Martin C, Rubio B (2016) State-of-the-art challenges, and open issues in an integration of internet of things and cloud computing. J Netw Comput Appl 67:99–117. https://doi.org/10.1016/j.jnca.2016.01.010
3. Milani AS, Navimipour NJ (2016) Load balancing mechanisms and techniques in the cloud environments: systematic literature review and future trends. J Netw Comput Appl 71:86–98. https://doi.org/10.1016/j.jnca.2016.06.003
4. Zhi-H Zhan, Xiao-F Liu, Yue-Jiao Gong, Zhang J (2015) Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Comput Surv 47(4):1–33. https://doi.org/10.1145/2788397
5. Hua H, Guangquan X, Shanchen P, Zenghua Z (2016) AMTS: adaptive multi-objective task scheduling strategy in cloud computing. China Commun 13(4):162–171. https://doi.org/10.1109/CC.2016.7464133
6. Yuan H, Bi J, Tan W, Zhou M, Li BH, Li J (2017) TTSA: an effective scheduling approach for delay bounded tasks in hybrid clouds. IEEE Trans Cybern 47(11):3658–3668. https://doi.org/10.1109/TCYB.2016.2574766
7. Zhong Z, Chen K, Zhai X, Zhou S (2016) Virtual machine-based task scheduling algorithm in a cloud computing environment. Tsinghua Sci Technol 21(6):660–667. https://doi.org/10.1109/TST.2016.7787008
8. Xu X, Cao L, Wang X (2016) Resource pre-allocation algorithms for low-energy task scheduling of cloud computing. J Syst Eng Electron 27(2):457–469. https://doi.org/10.1109/JSEE.2016.00047
9. Sharma SCM, Rath AK (2017) Multi-Rumen Anti-Grazing approach of load balancing in cloud network. Int J Info Technol 9(2):129–138. https://doi.org/10.1007/s41870-017-0022-y
10. Singha A, Junejab D, Malhotra M (2015) Autonomous agent based load balancing algorithm in cloud computing. Procedia Comput Sci 45:832–841. https://doi.org/10.1016/j.procs.2015.03.168
11. Gutierrez-Garcia JO, Ramirez-Nafarrate A (2015) Agent-based load balancing in cloud data centers. Cluster Comput 18(3):1041–1062. https://doi.org/10.1007/s10586-015-0460-x
12. Chun-C L, Hui-H C, Der-J D (2014) Dynamic multiservice load balancing in cloud-based multimedia system. IEEE Syst J 8(1):225–234. https://doi.org/10.1109/JSYST.2013.2256320
13. Chitra DD, Uthariaraj VR (2016) Load balancing in cloud computing environment using improved weighted round robin algorithm for nonpreemptive dependent tasks. Sci World J 2016:1–14. https://doi.org/10.1155/2016/3896065
14. Tao D, Lin Z, Wang B (2017) Load feedback-based resource scheduling and dynamic migration-based data locality for virtual Hadoop clusters in OpenStack-based clouds. Tsinghua Sci Technol 22(2):149–159.https://doi.org/10.23919/TST.2017.7889637
15. Xie R, Wen Y, Jia X, Xie H (2015) Supporting seamless virtual machine migration via named data networking in cloud data center. IEEE Trans Parallel Distrib Syst 26(12):3485–3497. https://doi.org/10.1109/TPDS.2014.2377119
16. Mosleh Mohammed AS, Radhamani G, Hazber Mohamed AG, Hasan SH (2016) Adaptive cost-based task scheduling in cloud environment. Sci Program 2016:1–9. https://doi.org/10.1155/2016/8239239
17. Liu Y, Li C, Li L (2016) Distributed two-level cloud-based multimedia task scheduling. Automat Contr Comput Sci 50(3):41–150. https://doi.org/10.3103/S0146411616030044
18. Shi L, Zhang Z, Robertazzi T (2017) Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in cloud. IEEE Trans Parallel Distrib Syst 28(6):1607–1620. https://doi.org/10.1109/TPDS.2016.2625254
19. Li Y, Chen M, Dai W, Qiu M (2017) Energy optimization with dynamic task scheduling mobile cloud computing. IEEE Syst J 11(1):96–105. https://doi.org/10.1109/JSYST.2015.2442994
20. Keng-M C, Pang-W T, Chun-W T, Chu-S Y (2015) A hybrid meta-heuristic algorithm for VM scheduling with load balancing in cloud computing. Neural Comput Appl 26(6):297–1309. https://doi.org/10.1007/s00521-014-1804-9
21. Eswaran S, Rajakannu M (2017) Multiservice load balancing with hybrid particle swarm optimization in cloud-based multimedia storage system with QoS provision. Mobile Netw Appl 22(4):760–770. https://doi.org/10.1007/s11036-017-0840-y

22. Dhinesh Babu LD, Venkata Krishna P (2013) Honey bee behavior inspired load balancing of tasks in cloud computing environments. Appl Soft Comput 13:2292–2303. https://doi.org/10.1016/j.asoc.2013.01.025
23. Negi S, Panwar N, Vaisla K S, Rauthan MMS (2020) Artificial Neural Network Based Load Balancing in Cloud Environment. Advances in Data and Information Sciences. Lecture Notes in Networks and Systems, 94. /https://doi.org/10.1007/978-981-15-0694-9_20.
24. Jeyakrishnan V, Sengottuvelan P (2017) A Hybrid Strategy for Resource Allocation and Load Balancing in Virtualized Data Centers Using BSO Algorithms. Wireless Pers Commun 94(4):2363–2375. https://doi.org/10.1007/s11277-016-3481-8
25. Polepally V K, Chatrapati K S (2017) Dragonfly optimization and constraint measure-based load balancing in cloud computing. Cluster Computing. Springer.https://doi.org/10.1007/s10586-017-1056-4
26. 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. https://doi.org/10.1109/TPDS.2015.2402655
27. Tsakalozos K, Verroios V, Roussopoulos M, Delis A (2017) Live VM Migration under Time-Constraints in Share-Nothing IaaS-Clouds. IEEE Trans Parallel Distrib Syst 28(8):2285–2298. https://doi.org/10.1109/TPDS.2017.2658572
28. Kansal Nidhi J, Chana I (2016) Energy-aware Virtual Machine Migration for cloud computing—a firefly optimization approach. J Grid Comput 14(2):327–345. https://doi.org/10.1007/s10723-016-9364-0
29. Farahnakian F, Pahikkala T, Liljeberg P, Plosila J, TrungHieu N, Tenhunen H (2016) Energy-aware VM Consolidation in Cloud Data Centers Using Utilization Prediction Model. IEEE Transactions on Cloud Computing (99).https://doi.org/10.1109/TCC.2016.2617374
30. Patel G, Mehta R, Bhoi U (2015) Enhanced Load Balanced Min-Min algorithm for Static Meta-task Scheduling in Cloud Computing. Procedia Computer Science (Elsevier). https://doi.org/10.1016/j.procs.2015.07.385
31. Lakraa AV, Yadav DK (2015) Multi-Objective Tasks Scheduling Algorithm for Cloud Computing Throughput Optimization. Procedia Computer Science, Elsevier 48:107–113. https://doi.org/10.1016/j.procs.2015.04.158
32. Zhang P, Zhou Meng C (2017) Dynamic Cloud Task Scheduling Based on a Two-Stage Strategy. IEEE Trans Autom Sci Eng 99:1–12. https://doi.org/10.1109/TASE.2017.2693688
33. Zuo X, Zhang G, Tan W (2014) Self-Adaptive Learning PSO-Based Deadline Constrained Task Scheduling for Hybrid IaaS Cloud. IEEE Trans Autom Sci Eng 11(2):564–573. https://doi.org/10.1109/TASE.2013.2272758
34. Awada AI, El-Hefnawyb NA, Abdelkader HM (2015) Enhanced Particle Swarm Optimization for Task Scheduling in Cloud Computing Environments. Procedia Comput Sci Elsevier 65:920–929. https://doi.org/10.1016/j.procs.2015.09.064
35. Panwar N, Negi S, Rauthan MMS, Vaisla KS (2019) TOPSIS–PSO inspired non-preemptive tasks scheduling algorithm in cloud environment. Clust Comput. https://doi.org/10.1007/s10586-019-02915-3
36. Pooranian Z, Shojafar M, Abawajy Jemal H, Abraham A (2013) An efficient meta-heuristic algorithm for grid computing. J Comb Optim, Springer 30(3):413–434. https://doi.org/10.1007/s10878-013-9644-6
37. Brochu E, Cora Vlad M, Freitas Nando D (2013) A Tutorial on Bayesian Optimization of Expensive Cost Functions with Application to Active User Modeling and Hierarchical Reinforcement Learning. https://arxiv.org/abs/1012.2599
38. Nyikosa F M, Osborne M A, Roberts S J (2018) Bayesian Optimization for Dynamic Problems. https://arxiv.org/abs/1803.03432
39. Wagner C (2013) Juzzy – A Java based Toolkit for Type-2 Fuzzy Logic. IEEE. Symposium on Advances in Type-2 Fuzzy Logic Systems (T2FUZZ). https://doi.org/10.1109/T2FZZ.2013.6613298
40. Mendel JM, John RI, Liu F (2006) Interval Type-2 Fuzzy Logic Systems Made Simple. IEEE Trans Fuzzy Syst 14(6):808–821
41. Kasper Fredenslund (2018) March 25. https://kasperfred.com/series/introduction-to-neural-networks/computational-complexity-of-neural-networks

42. Baptista R, Poloczek M (2018) Bayesian optimization of combinatorial structures. In: Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR. 80. https://arxiv.org/abs/1806.08838
43. Ren Q, Balazinski M, Baron L (2011) Type-2 TSK fuzzy logic system and its type-1 counterpart. Int J Comput Appl 20(6):0975–8887. https://doi.org/10.5120/2440-3292
44. Buyya R, Ranjan R, Calheiros R N (2019) Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities. High Performance Computing & Simulation HPCS'09. 1–11. https://doi.org/10.1109/HPCSIM.2009.5192685
45. Shojafar M, Javanmardi S, Saeid A, Nicola C (2015) FUGE: a joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method. Clust Comp 18(2):829–844
46. Chen Y (2019) Study on Centroid Type-Reduction of Interval Type-2 Fuzzy Logic Systems Based on Noniterative Algorithms. Compl Hindwai 2019:1–12. https://doi.org/10.1155/2019/7325053
47. Mendel JM (2013) On KM algorithms for solving type-2 fuzzy set problems. IEEE Trans Fuzzy Syst 21(3):426–446
48. Liang Q, Mendel J (2000) Interval Type-2 fuzzy logic systems: theory and design. IEEE Trans Fuzzy Syst 8:535–550
49. Singh H, Tyagi S, Kumar P (2020) Crow–penguin optimizer for multiobjective task scheduling strategy in cloud computing. Int J Commun Syst 33(5):e4467. https://doi.org/10.1002/dac.4467
50. Prassanna J, Venkataraman N (2019) Threshold based multi-objective memetic optimized round robin scheduling for resource efficient load balancing in cloud. Mobile Netw Appl 24:1214–1225. https://doi.org/10.1007/s11036-019-01259-x
51. Neelima P, Rama Mohan Reddy A (2020) An efficient load balancing system using adaptive dragonfly algorithm in cloud computing. Cluster Comput 23:2891–2899. https://doi.org/10.1007/s10586-020-03054-w

Authors and Affiliations

Sarita Negi[1] · Man Mohan Singh Rauthan[2] · Kunwar Singh Vaisla[3] · Neelam Panwar[2]

✉ Sarita Negi
sarita.negi158@gmail.com

Man Mohan Singh Rauthan
mms_rauthan@rediffmail.com

Kunwar Singh Vaisla
vaislaks@rediffmail.com

Neelam Panwar
neelam.panwar001@gmail.com

[1] Computer Science and Engineering, Uttarakhand Technical University, Dehradun 248007, Uttarakhand, India

[2] Computer Science and Engineering, Hemvati Nandan Bahuguna Garhwal University, Garhwal, Srinagar 249161, Uttarakhand, India

[3] Computer Science and Engineering, Bipin Tripathi Kumaon Institute of Technology, Dwarahat, Almora 263653, Uttarakhand, India