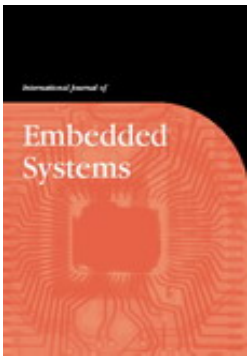


International Journal of Embedded Systems

ISSN online: 1741-1076 - ISSN print: 1741-1068
https://www.inderscience.com/ijes

An improved VM selection and allocation hybrid algorithm using grasshopper and firefly in cloud computing

Rachhpal Singh, Sarpreet Singh

DOI: 10.1504/IJES.2024.10068150

Article History:

Received:	23 February 2024
Last revised:	25 July 2024
Accepted:	24 September 2024
Published online:	10 February 2025

An improved VM selection and allocation hybrid algorithm using grasshopper and firefly in cloud computing

Rachhpal Singh* and Sarpreet Singh

Department of Computer Science,
Sri Guru Granth Sahib World University,
Fatehgarh Sahib, India
Email: rachhpal@pbi.ac.in
Email: ersarpreetvirk@gmail.com
*Corresponding author

Abstract: Cloud computing has emerged as a dynamic and resource-intensive domain, demanding innovative solutions to efficiently allocate and manage resources. In response, this paper introduces a pioneering optimisation algorithm that seamlessly integrates the Grasshopper Algorithm for virtual machine selection and the firefly algorithm (FA) for physical machine selection. The primary objective is to optimise critical quality of service parameters while effectively addressing challenges associated with service level agreement violation (SLA-V) and power consumption. Through a comprehensive series of rigorous evaluations, Grasshopper and FA consistently outperform existing solutions. The authors demonstrate significant reduction in SLA-V and power consumption, offering tangible benefits to cloud service providers. This work represents a promising advancement in cloud resource management, aligning with green computing initiatives and promising cost-saving opportunities.

Keywords: cloud computing; CC; resource allocation; grasshopper algorithm; firefly algorithm; FA; quality of service; QoS; SLA violation; power consumption; PC; optimisation; green computing; cost savings.

Reference to this paper should be made as follows: Singh, R. and Singh, S. (2024) 'An improved VM selection and allocation hybrid algorithm using grasshopper and firefly in cloud computing', *Int. J. Embedded Systems*, Vol. 17, Nos. 3/4, pp.251–266.

Biographical notes: Rachhpal Singh is currently an Assistant Professor in Computer Science at Punjabi University Centre for Emerging and Innovation Technology, Mohali, Punjab, India. He is pursuing his PhD degree with the Department of Computer Science, at Sri Guru Granth Sahib World University, Fatehgarh Sahib, Punjab, India. He did his Master of Technology (MTech) and Master of Computer Applications (M.C.A.) at Punjabi University Patiala (Punjab). He has teaching and research experience of more than 12 years in Computer Science. He has presented various papers at national and international conferences. His areas of interest include cloud computing, programming languages, and artificial intelligence.

Sarpreet Singh is working as an Assistant Professor in the Department of Computer Science at Sri Guru Granth Sahib World University. His areas of expertise are grid computing and cloud computing. He has published over 35 research papers in various international journals and conferences. He has supervised more than 40 students in their post-graduation courses for computing research.

1 Introduction

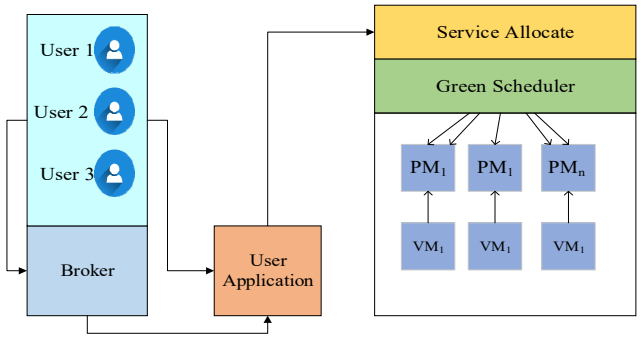
With the evolution of information technology (IT), the attraction towards achieving green cloud technology has also become more and more popular. Cloud computing (CC) is one of the technologies that have always been a keen are of interest for scientific community. Initially, as early as 2008, cloud was thought as an option for performing speedy executions (Panwar and Supriya, 2022). Later on, its horizons spread to application architecture while delivering services related to infrastructure as well. Today, cloud comprises of three layers, namely, infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS) (Guarda et al., 2021).

The execution of any application needs an operating platform and hence PaaS has become essential part of SaaS. The leading vendors such as Azure, AWS and Google cloud platform are available in the market to offer 100's of services with individual pricing that can be customised based on configuration options. Thus, the energy requirements of a cloud data centre is expected to rise from

200 TWh to 2967 TWh from 2016 to 2030 (Koot and Wijnhoven, 2021).

Virtual machine (VM) allocation and migration are critical techniques in CC that play a significant role in improving the job computation rate and overall system performance. In cloud environments, numerous users and applications simultaneously utilise shared resources, making efficient resource allocation and management essential for meeting performance requirements and optimising resource utilisation. VM allocation involves assigning VMs to physical machines (PMs) based on specific criteria, while VM migration refers to the process of moving VMs from one PM to another during runtime (Stillwell et al., 2010).

Figure 1 Wireless sensor network architecture involving aggregator node (see online version for colours)



The significance of VM allocation and migration lies in their ability to enhance the job computation rate in CC environments. By dynamically allocating VMs to appropriate PMs and migrating them when needed, it becomes possible to achieve load balancing, resource optimisation, and improved system performance. Efficient VM allocation ensures that each VM is placed on an appropriate PM with sufficient resources to meet its computational demands (Talwani et al., 2022). This avoids resource bottlenecks, maximises resource utilisation, and prevents overloading of PMs, resulting in faster job execution and improved responsiveness.

Furthermore, VM migration allows for workload management and adaptation to changing system conditions. For example, when a PM becomes overloaded or experiences hardware failures, VM migration can be employed to redistribute the workload and ensure uninterrupted service. By migrating VMs from heavily loaded PMs to underutilised ones, the computational load is evenly distributed across the cloud infrastructure, enabling faster job completion times and improved scalability (Wang et al., 2019).

In CC environments, VM allocation and migration strategies can be guided by various factors such as VM resource requirements, PM capabilities, network conditions, and workload characteristics. Advanced algorithms and techniques, including machine learning, optimisation algorithms, and predictive analytics, can be applied to automate and optimise the decision-making process (Manaswi and Sharma, 2024). These techniques consider factors such as PM performance, energy efficiency, cost, and user-defined preferences to determine the most suitable placement and migration strategies (Szabo et al., 2014). The benefits of efficient VM allocation and migration extend beyond speeding up job computation rates. They include improved resource utilisation, reduced energy consumption, and enhanced scalability and flexibility. By dynamically adjusting resource allocation based on workload demands and system conditions, cloud providers can deliver higher-quality services, meet service-level agreements (SLAs), and efficiently utilise their infrastructure resources. Swarm Intelligence has emerged as a powerful technique for solving complex optimisation problems by drawing inspiration from the collective behaviour of social insect colonies. One important application of Swarm Intelligence is in VM selection from overutilised PM to minimise overall PC. With the exponential growth of CC and data centres, energy efficiency has become a crucial concern. By efficiently allocating VMs to underutilised PMs and consolidating workloads, Swarm Intelligence can help reduce energy consumption, operational costs, and environmental impact (Singh et al., 2021; Meshkati and Safi-Esfahani, 2019). Efficient PC management plays a crucial role in the allocation and migration of VMs within a cloud data centre. By considering PC as a critical factor, sophisticated VM allocation and migration strategies can be designed to optimise resource utilisation while minimising energy waste. Effective allocation ensures that VMs are placed on PMs with sufficient capacity and compatible power profiles, leading to improved energy efficiency and reduced operational costs. Furthermore, intelligent VM migration techniques enable the consolidation of workloads onto a reduced number of PMs, promoting better resource utilisation and minimising overall PC. By prioritising power-aware VM allocation and migration, cloud data centres can achieve significant energy savings and enhance the sustainability of their operations.

By leveraging swarm intelligence algorithms, such as ant colony optimisation (ACO), particle swarm optimisation (PSO), or artificial bee colony (ABC), grasshopper and firefly optimisation algorithm, VM selection from overutilised PMs can be optimised to minimise the overall PC (Elmagzoub et al., 2021; Alharbi et al., 2021). These algorithms mimic the collective behaviour and intelligence of social insects to find optimal solutions in a decentralised and self-organising manner. They can effectively explore the solution space, consider multiple factors like CPU utilisation, memory usage, and network traffic, and make intelligent decisions on VM placement and migration to underutilised PMs. By intelligently managing VM placement and workload consolidation using Swarm Intelligence, data centre operators can achieve significant energy savings. This not only reduces operational costs but also contributes to environmental sustainability by minimising carbon footprint. Furthermore, efficient VM selection and power management also improve the overall performance, reliability, and scalability of the cloud infrastructure. In conclusion, Swarm Intelligence offers a promising approach for VM selection from overutilised

PMs to minimise PC in data centres. By leveraging the collective intelligence of swarm-based algorithms, data centre operators can optimise VM placement and workload consolidation, leading to improved energy efficiency, reduced operational costs, and a greener computing environment.

The contributions of the proposed work in the same regard is as follows:

- Integration of optimised grasshopper algorithm for VM selection: The proposed solution incorporates the integration of the optimised grasshopper algorithm, a metaheuristic optimisation technique, for efficient VM selection. By leveraging the algorithm's capabilities, the solution aims to improve the selection process by considering various factors such as resource utilisation, load balancing, and minimising response time.
- Firefly algorithm (FA) for VM replacement to target PM: The solution also integrates the FA, another metaheuristic optimisation technique, for the replacement of VMs to the target PM. By applying the FA, the solution optimises the allocation of VMs to PMs based on factors such as resource availability, PM capacity, and minimising migration time.
- Evaluation of proposed work on the base of quality of service (QoS) parameters.
- Comparison of the proposed work with other state of art algorithms

Thus, the introduction section provides the background search on CC with reference to user demand and service allocation. The paper is organised in seven sections including introduction section. Section 2 provides an extensive literature review on existing strategies on VM selection and placement algorithms. The proposed method is described in Section 3 in multiple subsections, including the, system architecture and power modelling. Section 4 is dedicated for the problem formulation and Section 5 presents the proposed hybrid algorithm that combines two swarm intelligence techniques, namely, Grasshopper Optimisation and FA. The criteria used to assess this algorithm are also covered in this section. This is followed by results and performance comparisons in Section 6. Section 7 represents the conclusion section of the paper.

2 Related work

Several novel approaches have been investigated in the recent literature on VM management in CC environments. Enhancing energy economy and performance as well as optimising VM placement and migration techniques through sophisticated algorithmic interventions have been the main areas of attention. The main conclusions of multiple investigations published in recent past be compiled in this review, along with any pertinent downsides.

Energy-aware strategies for data centre resource allocation were first presented by Beloglazov et al. (2012). Their method satisfied operational constraints such as QoS while minimising energy use. Their heuristic approach's main drawback was its reliance on pre-established thresholds and parameters, which may make it difficult to adjust to workload fluctuations or real-time modifications in the conditions of data centres.

A firefly optimisation technique for energy-aware VM migration was proposed by Kansal and Chana (2016). Although novel, bio-inspired algorithms such as firefly may not scale well in bigger settings and required a great deal of parameter adjusting to adapt to complex and dynamic cloud environments.

PSO was used by Chou et al. (2016) for dynamic power-saving in cloud data centres, a technique that demonstrated notable gains in energy efficiency. PSO's stochastic nature, however, may cause problems with convergence and possibly uneven performance between runs.

Naik et al. (2020) introduced FHCS for VM migration and task scheduling in cloud data centres, aiming at improving resource utilisation and performance. Although the hybrid strategy sought to enhance overall performance and energy efficiency, the incorporation of several optimisation strategies may result in a rise in computing overhead and algorithmic complexity.

Dubey and Sharma (2020) presented an extended intelligent water drop approach for VM allocation in a secure CC framework, focusing on enhancing security while efficiently allocating resources.

Singh and Singh (2021) approach leveraged the internet learning capabilities of artificial neural networks to dynamically allocate VMs based on changing workload demands in cloud environments. By employing neural networks, their method aimed to adaptively allocate resources, optimising the utilisation of cloud infrastructure while ensuring efficient allocation of virtual resources to meet performance requirements. The primary disadvantage in this case was the need for substantial computer power and big datasets in order to properly train the neural networks.

Tarahomi et al. (2021) proposed a power-aware VM allocation mechanism in cloud data centres using a micro genetic-based approach, aiming to optimise power consumption (PC) while meeting performance requirements. Although genetic algorithms are renowned for their ability to explore a wide range of solutions, it is important to carefully calibrate genetic operators in order to prevent premature convergence.

Zaffar (2021) study focused on modelling and forecasting sunspot cycles using ARMA (p, q)-GARCH (1, 1) models. While not directly related to CC, this research showcases the application of advanced modelling techniques.

Mohammed and Zeebaree (2021) conducted a comprehensive review of CC services, including IaaS, PaaS, and SaaS. Their study provides valuable insights into the features and suitability of these services for various

applications, aiding in better decision-making for cloud deployments.

Tran et al. (2022) introduced a VM migration policy for multi-tier applications in CC based on the Q-learning algorithm, focusing on optimising performance and resource utilisation. It proved to be quite successful in accommodating environment-specific needs. On the other hand, the learning-based strategy was sensitive to the original policy parameters and required long training periods.

Talwani et al. (2022) proposed a machine learning-based approach for VM allocation and migration in CC. They harnessed the power of machine learning techniques to optimise resource allocation, which can significantly enhance the efficiency and performance of cloud data centres. It offered adaptive and predictive resource management capabilities. However, the opacity and complexity of machine learning models may make maintenance and debugging difficult.

Singh and Singh (2023) introduced a bio-inspired approach for VM migration. Their method utilised re-initialisation and decomposition based-whale optimisation techniques to improve VM placement and migration strategies. This approach offers the potential for more efficient VM allocation and resource utilisation in cloud environments. Large-scale ecosystems were easier to manage because to the creative decomposition technique, but there was still a big problem with performance being dependent on beginning population and parameter settings.

In order to accomplish effective VM placement in cloud data centres, Durairaj and Sridhar (2023) proposed the multi-objective mayfly optimisation algorithm for VM placement (MOM-VMP), which is augmented using principal component analysis (PCA). One significant limitation of the approach was its computational complexity, which could result in longer processing times when making real-time judgements on VM placement, even though it demonstrated promise in terms of optimising resource use and lowering energy consumption.

An upgraded version of the simple and efficient simulated annealing (SESA) method for VM migration was created by Kaur et al. (2023). The goal of this methodology was to lower the cost of migration and downtime. The method may not scale well in extremely dynamic contexts and its performance was largely dependent on the original parameters, even with the enhancements.

An improved ordinal optimisation method was developed by Yadav and Mishra (2023) with the goal of lowering scheduling overhead in task scheduling. Their approach was novel in reducing operating expenses and boosting throughput, but as the study focused mostly on simulations, real-world application may reveal other difficulties, such as adaptability to various cloud infrastructures.

An efficient energy-consuming genetic-based technique for VM consolidation was proposed by Radi et al. (2023). Although this method proved to be energy-efficient, the slow convergence rates of genetic algorithms may make them unsuitable for usage in situations where making fast decisions is essential.

A fuzzy and predictive strategy was used by Zolfaghari (2024) to improve energy-performance awareness during VM migrations. This strategy effectively balanced workload and energy consumption, but it may not be as dependable in unexpected operating environments due to its reliance on precise forecasts and the intricacy of fuzzy logic systems.

Mehta et al. (2024) investigated enhanced whale optimisation options for VM placement that complies with SLA requirements. Although the application of whale optimisation may be computationally expensive and unsuitable for time-sensitive VM placement scenarios, their techniques optimised the placement process by taking SLA restrictions into consideration.

Vijaya and Srinivasan (2024) used a multi-objective meta-heuristic approach with an emphasis on energy efficiency to address VM placement. Although the meta-heuristic's multi-objective character could make it difficult to balance conflicting aims, which could have an impact on the optimality of solutions, their method proved successful in maximising the use of energy and resources.

Overall, it has been observed that the research in the recent years has made a substantial contribution to the field of CC, especially in terms of optimising VM administration. All of the approaches, albeit novel, may have inevitable limitations that reduce their usefulness in real-world applications. These difficulties include the necessity for real-time adaptation, scalability concerns, high computational needs, and reliance on starting parameter all of which are essential for CC settings. It is recommended that more study be done to enhance these methods, guarantee wider applicability, and increase computing efficiency.

3 Proposed work

The proposed work incorporates the system model and design in terms of the cloud data centres, power modelling and the system architecture along with the proposed solution.

3.1 Cloud data centres and resources

Consider a cloud data centre consisting of m heterogeneous PMs. Let P represent the set of PMs, where jPj = m. Each PM is denoted by PMj and characterised by a six-element tuple: PMj = (Idj, CPU_MIPSj, PEj, RAMj, BWj, Sizej). Here, P must be tuples characterised by unique identifiers, numerical attributes for CPU capacity, number of processing elements, RAM capacity, bandwidth, and storage size. These attributes are typically integers or floats. Here, Idj is the unique identity, CPU MIPSj represents CPU performance in million instructions per second, PEj denotes the processing element, RAMj represents the amount of main memory, BWj is the network bandwidth, and Sizej corresponds to secondary storage. Now, let's assume there are n total VMs in the data centre. V represents the set of

VMs, where jVj = n. Each VM is denoted by VMi and characterised by a six-element tuple: VMi = (Idi, CPU_MIPSi, RAMi, BWi, Sizei, UserIdi). Here, Idi is the unique identity, CPU_MIPSi, RAMi, BWi, and Sizei represent the requested amount of CPU-MIPS, memory, bandwidth, and size by the VM. UserIdi denotes the unique ID of the user. In this model, each VM is deployed to run a single application or service, and if needed, multiple VMs can be deployed for a single application or service. It is important to understand that if PM value based on six elements are not fulfilled, the PM will lack a complete description. As such it can lead to incorrect or inefficient resource allocation, scheduling, and overall system performance issues.

3.2 Power modelling

PC is a critical aspect of data centre operations. It is directly related to the energy consumed by PMs and the associated cooling systems. The power consumed by a PM consists of two components: static power (P_{static}) and dynamic power ($P_{dynamic}$). The static power refers to the power consumed by a PM even when it is idle or not executing any workload. On the other hand, dynamic power is the power consumed by a PM while executing tasks (Szabo et al., 2014).

The total power consumed by a PM can be represented as the sum of static and dynamic power, as shown in equation (1):

$$P_{total} = P_{static} + P_{dynamic} \tag{1}$$

The static power (P_{static}) is relatively constant and is mainly attributed to components such as the motherboard, power supply, and cooling fans. On the other hand, the dynamic power ($P_{dynamic}$) depends on the CPU utilisation and workload characteristics. It can be calculated using equation (2):

$$P_{dynamic} = P_{\max} \times U \times C \tag{2}$$

Here, $P_{\max}$ represents the maximum PC of the PM, U denotes the CPU utilisation, and C is a constant factor representing the dynamic PC per unit CPU utilisation.

The PC of PMs is known to be influenced by various factors, including CPU-MIPS utilisation, RAM utilisation, network bandwidth utilisation, and storage utilisation. Among these factors, CPU-MIPS utilisation has been observed to contribute the most to PC, accounting for approximately 40%–75% of the total PC

The power model is formulated based on the linear relationship between CPU-MIPS utilisation and PC. The PC of a PM, denoted as power (CMU), is defined as a function of CPU-MIPS utilisation (CMU) and is expressed by the following equation:

$$Power(CMU) = C * P\max_i + (1 - C) * P\max_i * CMU \tag{3}$$

Here, $P\max_i$ represents the maximum PC of a PM with full capacity or 100% CPU-MIPS utilisation. The value of C, ranging between 0 and 1, represents the fraction of power consumed by an idle PM. The specific value of C depends on the type of PM being used (e.g., PM1, PM2, etc.)

To further analyse the energy consumption of a PM over a specific time interval [t_0, t_1], denoted as E, the PC is integrated over that interval. This integration is carried out by considering the time-varying CPU-MIPS utilisation, denoted as $CMU(t)$, which represents the utilisation at a given time t. The energy consumption E can be calculated using the following equation:

$$E = \int_{t_0}^{t_1} Power\left(CMU(t)\right)dt \tag{4}$$

By utilising this power model, it becomes possible to estimate the energy consumption of the PMs within the cloud data centre. This modelling approach allows for detailed analysis of PC trends and provides insights into resource utilisation and optimisation strategies for minimising energy consumption.

4 Problem formulation

The problem addressed in this research is the optimisation of VM selection and allocation in CC environments to reduce overall PC and prevent SLA violations. Existing methods lack efficient optimisation and fail to consider important factors such as resource utilisation, load balancing, and response time. The proposed solution integrates the optimised grasshopper algorithm for VM selection and the FA for VM replacement, aiming to minimise PC and improve SLA compliance. The problem can be mathematically formulated as follows:

$$Minimise\ PC: minimise \sum_{i=1}^{m} (PM\_power) \tag{5}$$

Subject to:

$$\sum_{j=1}^{n} \left(VM_{resource_{utilisation}}\right) \le PM_{capacity} \tag{6}$$

$$\sum \left(VM_{PC}\right) \le SLA_{threshold} \tag{7}$$

where

- $\Sigma(PM_{power})$ represents the total PC of all PMs in the cloud infrastructure.
- $\Sigma(VM_{resource\_utilisation})$ represents the total resource utilisation of all selected VMs.
- $PM_{capacity}$ denotes the capacity of a PM in terms of available resources.
- $SLA_{threshold}$ denotes the maximum allowable response time as per the SLA agreement.

The objective is to minimise the total PC while ensuring that the total resource utilisation, response time, and migration time satisfy their respective thresholds. The integration of the optimised grasshopper algorithm and the FA will provide an efficient solution for VM selection and replacement, considering these optimisation constraints. To control PC in the approach, optimise CPU utilisation and use dynamic voltage and frequency scaling (DVFS). Additionally, implement efficient task scheduling and

resource allocation policies to reduce both static and dynamic PC.

5 Proposed work

The proposed work is divided into two parts. The first part incorporates the preliminary allocation of a VMi to a PMj if the PMj has enough available resources for the VM. In order to do so, a broadcast mechanism is designed that is inspired by modified best fit decreasing (MBFD) algorithm in which each possible PM is analysed in such a manner that the following constraints are satisfied.

$$PMj.\ resources\ > VMi.\ resource\ demand$$

$$Arg\ \min(AllocationPC)$$

The proposed work can be presented using the following work flow diagram shown in Figure 2 with yellow box presenting the innovative step.

The first part also finds the hotspot PMs viz. the PMs that are either overutilising the resources or underutilising the resources. Once the hotspots are identified, the VMs from overutilised PMs are to be selected in order to make it neutral whereas the underutilised PM loses all its VMs.

$$\left(O_u \text{ if } |Norm\left(PM_{CPU}, PM_{PC}\right) \geq Threshold\ \right) \tag{8}$$

$$\left(U_u \text{ if } |\ Norm\left(PM_{CPU}, PM_{PC}\right) \leq Threshold\right) \tag{9}$$

where O_u is the overutilised PMs where U_u is the underutilised PMs.

The threshold is calculated by taking a margin of 10%–30% of the average threshold.From the overutilised PMs, the VMs are selected using proposed O-GrA (optimised grasshopper) algorithm. Once the VMs are selected from Ou, the O-FrA (optimised firefly) algorithm is applied for the replacement of the VMs over the remaining PMs.

Algorithm B-MBFD (Broadcast MBFD)

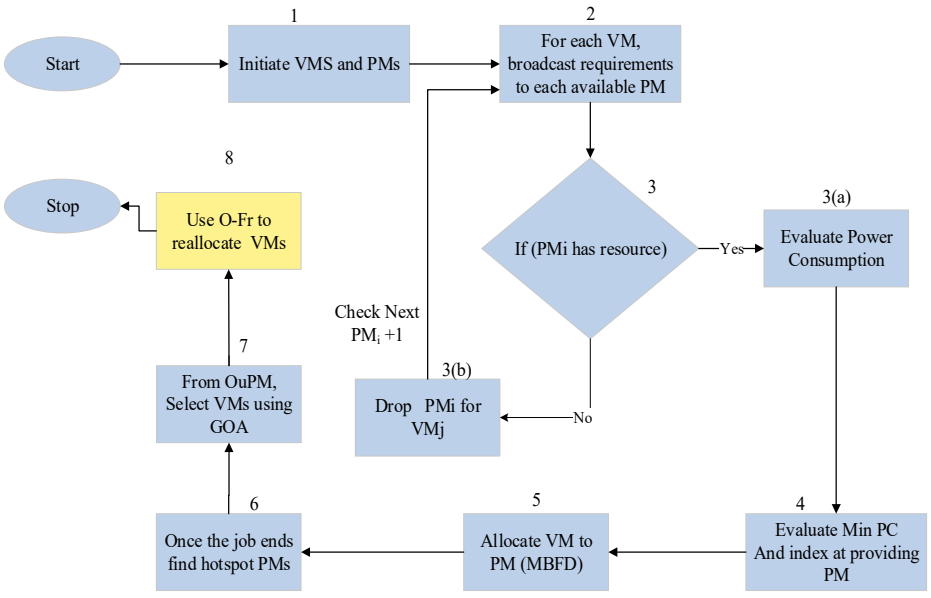
Input: VMs (list of virtual machines), PMs (list of physical machines)

Output: VM_to_PM (Dictionary representing VM to PM allocation)

```
Sort VMs in decreasing order based on resource requirements
Initialise an empty dictionary VM to PM
for each VM in VMs do
    Initialise a variable best_PM as None
    Initialise a variable min_remaining_capacity as infinity
    for each PM in PMs do
    if PM's capacity is greater than or equal to VM's resource requirements then
        if best_PM is None or PM's remaining capacity is less than min_remaining_capacity then
            Set best_PM as PM
            Set min_remaining_capacity as PM's remaining capacity
        end if
    end if
end for
if best_PM is not None then
        Assign VM to best_PM
        Update best_PM's remaining capacity by subtracting VM's resource requirements
        Add VM toPM allocation to VM to PM dictionary
    end if
end for
return VM to PM
```

The B-MBFD algorithm provides the allocation table containing four parameters, VMj viz the VM that is allocated to PMi, utilised CPU in the allocation, total PC of the allocation.

Figure 2 Work flow (see online version for colours)



Once the allocation is complete, the proposed work uses O-GrA which is an advancement of the existing grasshopper algorithm with the amendments in the evaluating fitness function and input structure of the grasshopper.

5.1 Grasshopper algorithm

The grasshopper algorithm is a metaheuristic optimisation technique that has emerged as an effective solution for complex optimisation problems, including the selection of VMs from overutilised PMs in CC environments. Its stepwise approach and its significance in VM selection from overutilised PMs can be outlined as follows:

1 Initialisation: The algorithm begins by initialising a population of candidate solutions, representing potential VM-to-PM allocations. The initial population can be generated randomly or based on heuristics, ensuring diversity in the solutions. The input data is derived from the custom dataset which is generated through simulations.

2 Fitness evaluation: Each candidate solution undergoes fitness evaluation using a defined fitness function. The function considers multiple factors such as PC, resource utilisation, SLA violations, and other relevant metrics. Fitness evaluation guides the algorithm to prioritise solutions that minimise PC and meet SLA requirements.

3 Affinity calculation: The affinity between VMs and PMs is computed, assessing the compatibility and suitability of each pairing. Affinity calculation considers factors like resource requirements, performance characteristics, and constraints. This step aids in identifying the most suitable VMs to allocate on overutilised PMs.

4 Selection: A selection mechanism is employed to choose candidate solutions for further exploration. Selection can be based on fitness values, dominance relations, or rank-based strategies. Solutions with better fitness or higher ranks, indicative of superior performance, are given higher probabilities for reproduction or exploration.

5 Variation operators: Variation operators such as crossover, mutation, or recombination are applied to the selected solutions, generating new offspring solutions. These operators introduce diversity and exploration, facilitating potential improvements in the VM-PM allocation.

6 Local search: Local search techniques can be incorporated to refine the solutions and explore neighbouring solution spaces. This step involves exploiting local optima and fine-tuning the allocation through small adjustments or swaps in the VM-PM assignments.

7 Termination criteria: The algorithm continues iterating through selection, variation, and local search until a termination criterion is met. Common termination criteria include a specific number of iterations, a predefined fitness threshold, or the convergence of solutions.

Table 1 Variables of proposed O-GrA algorithm

Variable	*Description*
A	Data matrix containing objective values
L	Indices matrix specifying data indices
N	Number of iterations
K	Number of clusters for K-Means clustering
Hg	Number of hoppers in a group
Lf	Number of levy flights
d_th	Threshold distance
λ	Penalty factor
Cf	Convergence factor
O	Objective values array
H	Grasshopper population
B	Baby hopper
G	Cluster labels
C	Cluster centroids
FqH	Food quality for individual hopper
FqG	Food quality for group
Sp	Surviving probability
Pos	Levy flight position
Fit	Fitness
I	Index with maximum fitness

Algorithm 1 Grasshopper VM migration algorithm

```
Input: Data matrix A, indices matrix L, number of iterations N, number of clusters K, parameters Hg, Lf, dth, λ, Cf
Output: Updated grasshopper population H
Initialize objective values array O
for j ← 0 to N - 1 do
    O[j] ← CollectObjectiveValues(A, L[j][0])
end for
Initialize grasshopper population H with collected objective values
Apply K-Means clustering to O with K clusters, obtaining labels G and centroids C
Initialize baby hopper B with zeros
 for j ← 0 to N - 1 do
    B ← InitializeBabyHopper(H[j])
    for l ← 0 to Lf - 1 do
        for m ← 1 to Hg - 1 do
            B[m] ← SelectGroupMember(H, N)
        end for
```

```
        FqH, FqG ← CalculateFoodQuality(B, C[G])
        Sp ← UpdateSurvivingProbability(Sp, FqH, FqG, λ)
        Pos ← GenerateLevyPosition()
        B ← UpdateBabyHopperPosition(B, Pos, C[G])
        Sp ← ApplyPenaltyFactor(Sp, λ)
    end for
      Fitness(Sp)= √((∑_{i=1}^{Lf}) Sp[i] )
    I ← FindMaxFitnessIndex(Sp)
  if Sp[I] > Cf then
      H[I] ← B
   end if
end for
```

The grasshopper algorithm's significance in VM selection from overutilised PMs lies in its ability to optimise resource allocation, minimise PC, and ensure SLA compliance. By harnessing its optimisation capabilities, the algorithm helps identify superior VM-PM allocations, resulting in enhanced energy efficiency, reduced operational costs, and improved QoS for cloud users. The grasshopper algorithm represents a valuable approach to address resource allocation challenges in cloud data centres, particularly in the context of overutilised PMs, where efficient VM selection is crucial for sustainable and high-performance CC environments.

The proposed work is inspired with the architecture of grasshopper algorithm and optimises it in terms of pairing and fitness evaluation as follows:

1 Input parameters:
- The algorithm takes several input parameters:
- Data matrix A containing objective values.
- Indices matrix L specifying data indices.
- Number of iterations N.
- Number of clusters K for K-Means clustering.
- Parameters Hg, Lf, hdth, λ, and Cf for controlling various aspects of the algorithm.

2 Objective values collection:
- The algorithm starts by collecting objective values from the data matrix A based on the indices provided in the matrix L.

3 Initialisation:
- It initialises the grasshopper population H with the collected objective values.
- Applies K-means clustering to the objective values array O to obtain cluster labels G and centroids C.

4 Baby hopper initialisation:
- Initialises the baby hopper B with zeros.

5 Main Loop:
- Iterates through each grasshopper in the population.
- For each grasshopper, performs a set of actions:
- Initialises the baby hopper B with values from the grasshopper.
- Executes a loop for a predefined number of levy flights:
- Selects group members randomly and updates the baby hopper.
- Calculates food quality for individual hopper (FqH) and group (FqG).
- Updates surviving probability (Sp).
- Generates a random position for levy flight (Pos).
- Updates the baby hopper position based on levy flight and centroid (C[G]).
- Applies a penalty factor (λ) to the surviving probability.

6 Fitness calculation:
- Calculates fitness based on the surviving probability.
- The fitness function is defined as the square root of the sum of surviving probabilities for all levy flights.

7 Selection:
- Finds the index with the maximum fitness.
- If the maximum fitness exceeds a predefined convergence factor Cf, updates the grasshopper with the new baby hopper position.

8 Output:
- Outputs the updated grasshopper population.

5.2 Firefly algorithm

FA is a metaheuristic algorithm proposed by Yang in 2009. The FA draws its inspiration from the social behaviour of fireflies and their flashing patterns. The light intensity and the attractiveness of a firefly are key components in the formulation of the algorithm.

The algorithm tries to mimic the behaviour of fireflies wherein the less bright fireflies tend to move towards the brighter ones. This is done in search of mate. If no brighter firefly is found, they move randomly. Moreover, the brightness of a firefly is seen as the objective function and is determined by the landscape of the problem under consideration.

Let us explain the algorithm in detail using mathematical notations:

- *Objective function*: First, we define the objective function f(x), where x = (x1, ..., xd) denotes the d-dimensional space of decision variables that are to be optimised.
- *Light intensity*: The light intensity I of a firefly at a particular location x can be determined by the value attained at that location by the objective function. The light intensity I(x) is thus proportional to the function f(x). The exact relationship between the light intensity

and the function can vary depending on the problem. For maximisation problems, the brighter fireflies are those with higher function values, whereas for minimisation problems, the brighter fireflies are those with lower function values.

- *Attractiveness*: The attractiveness beta of a firefly is seen as being proportional to the light intensity seen by adjacent fireflies. Thus, for a firefly i at location xi observing another firefly j at location x_j, if firefly j is brighter (i.e., has higher light intensity), the attractiveness beta of firefly j to firefly i is calculated as beta = beta0 * exp(-gamma*r_{ij}). Here, beta0 is the attractiveness of a firefly at zero distance (i.e., the attractiveness at r = 0), gamma is a light absorption coefficient, and r_{ij} is the Euclidean distance between x_i and x_j.
- *Movement*: In terms of movement, a firefly i is moved towards a brighter firefly j using the following formula: $x_i = x_i + beta*(x_j - x_i) + alpha*(rand-0.5)$. In this equation, alpha is the step size, rand is a random number drawn from a uniform distribution in the interval [0, 1], and $beta*(x_j - x_i)$ represents the attractiveness. The term alpha*(rand-0.5) introduces randomness in the movement, allowing fireflies to move randomly when there are no brighter fireflies.

Algorithm 2 O-Fr Algorithm

```
1     sfmc←0                                          →Selected for migration counter
2   alpha ←0.1                                        →Alpha
3   beta ←0.2                                         →Beta
4   gamma ←0.1                  →Gamma
5   ss←10                       →Step size
6   Outputs: Reallocated VMs
7   for sfm ← 1 to Lngth[SFM] do          →Selected for migration
8     cCD ← vmC[sfm]                                  →Current CPU demand
9     cRD ← vmRR[sfm]                                 →Current resource demand
10    sfmc ← sfmc + 1
11    pF←{}                                           →Possible flies
12    pC←{}                                           →Possible costs
13    pL ←{}                                          →Possible Light
14    cIS←cRDx 1000                                    →Current instruction set
15    for jj←1 to pmC do                    →PM count
16      aR←pmR[jj]                                    →Available resource
17      aC←pmC[jj]                                    →Available CPU
18    AV1 ←ConstantArray[0,{1,5}]           →Attraction Value1
19    AV←AV1[[1]]                                     →Attraction Value
20    {AIf1,AIf2,AIf3}←pmCosting [Length[pF] ⟶ At1 ⟶ cost, At2, PC, & At3 ⟶ CPUUtilisation
21    LIA ←{}                                         →Light Intensity All
22    for aif ←1 to Length[AIf1] do
23              L1 ← AIf1[aif]x cIS + (AIf2[aif] / aC) x AIf3{aif]      →Light intensity
24            AppendTo[LIA,LI]
25      end for
26  mG←1000 Max Generations
27  for mg←1 to mg do
28          ff1c ←0                                   →Firefly1 counter
29            for ff1←1 to Length [pF] do
30              ff2c←0                                →Firefly2 counter
31                for ff2←1 to Length [pF] do
32                  if ff1≠ff2 then
33                    if ff2c>Length[LIA] then
34                      ff2c←0
35                    end if
36                  dist←LIA[[ff1c]]-LIA[[ff2c]]                →Distance
37                ff2c←ff2c+1
38              aVal←β*exp(–γr^2).r+α                →A value
```

```
39              AV[[ff1c]] ← AV[[ff1c]] + 1/aVal
40            ff2c←ff2c+1
41          end if
42        end for
43          ff1c←ff1c+1
44        end for
45    end for
46        {mV,mI}←findMax[AV]                    →Max value,Max index
47        Print['VM:',sfm will be allocated to:pF[[mI]]
48    end for
```

Table 2 Utilised variables for FA

Variable name	*Description*
$\text{selected}_{\text{for}_{\text{migration}}}$ (SFM)	List of VMs that have been selected for migration.
VM_{CPU}	List that represents the CPU demand of each VM.
$\text{VM}_{\text{res}_{\text{req1}}}$	List that represents the resource demands of each VM.
PM_{res}	List representing the available resources for each physical machine (PM).
PM_{CPU}	List representing the available CPU for each physical machine (PM).
PM count (PMC)	Count of PMs.

Next, using the above pseudo-code, the O-Fr is implemented in the context of a VM migration issue, where the aim is to determine the optimal allocation of VMs to PMs to minimise some cost functions.

- Initialise parameters *alpha*, *beta*, *gamma*, and *stepsize* for the FA.
- Iterate over each VM *sfm* in the $selected_{for_{migration}}$ list.

  The counters *sfmcounter* and *sfm* are used to index into the VM_{CPU} and $VM_{res_{req1}}$ lists and if they exceed the length of these lists, they are reset or set to a random value within range.
- The current CPU and resource demand for VM *sfm* are computed. This is done by summing the corresponding entries in VM_{CPU} and $VM_{res_{req1}}$.
- The code then checks for each PM *jj* if it has sufficient CPU and resources to satisfy the demands of the current VM. If a PM has enough resources, it is considered a potential host for the VM and its index is added to the list $possible_{flies}$.
- If no PM is found that can host the VM, a random PM is chosen as a potential host.
- The code initialises the attraction values and calculates some costs using the $pm_{costing}$ function for each potential host PM. The exact nature of these costs is not clear from the pseudo code but they likely involve some measure of the 'cost' or 'suitability' of assigning the VM to a PM.
- The light intensity of each potential host (firefly) is calculated. The light intensity is a measure of the 'attractiveness' or 'suitability' of a PM as a host for the VM.
- In a nested loop, each potential host PM (firefly) is compared with every other potential host. If the light intensity of PM *ff*2 is greater than PM *ff*1, then *ff*1 is attracted towards *ff*2, moving in the decision space towards *ff*2. This is represented by updating the attraction value of *ff*1. Finally, after all potential host PMs have been compared, the PM with the maximum attraction value is chosen as the host for the current VM. The allocation of this VM to the chosen PM is then printed.

6 Result and discussion

This section evaluates the performance of O-Gr combined O-Fr algorithm and compare it with established state-of-the-art algorithms. The assessment employs key metrics such as total PC, total utilised CPU, total number of migrations, and SLA violation. Synthetic workloads were used in our experiments to gauge the algorithm's effectiveness. The results are juxtaposed with those of prominent VM migration algorithms, including Talwani et al. (2022), Kansal and Chana (2016) and Singh and Singh (2021). Findings indicate that O-Fr shows promise, demonstrating reduced total PC, optimised CPU utilisation, minimised migrations, and mitigated SLA violations.

SLA violation in the context of PC can be defined as the extent to which the actual PC of a system exceeds the specified power limit or threshold defined in the SLA.

$$SLA\ violation\ (\%) = \left[(Actual\ PC - SLA\ power\ limit) / SLA\ power\ limit\right] \quad (10)$$

Mathematically, the power limit (PL) based on the average value of neutral PMs can be expressed as:

$$PL = (1/N) * \Sigma Pi \tag{11}$$

where

PL is the power limit (average PC).

N is the total number of neutral PMs.

ΣPi represents the sum of PC values of all neutral PMs

The improvement observed in the PC in kW OGrA+OFr with an average PC of 10.76524162 kW for 1,250 VMs, can be attributed to the utilisation of the Grasshopper and Firefly optimisation techniques within the OGrA+OFr algorithm. When compared to other methodologies, such as PC in kW Talwani et al. (2022). (11.73675885 kW), 'PC in kW Kansal and Chana (2016)' (11.74478644 kW), and 'PC in kW Singh and Singh (2021)' (11.63079994 kW), it becomes evident that the OGrA+OFr approach outperforms the others in terms of power efficiency. These nature-inspired algorithms, such as grasshopper and firefly, excel in efficiently selecting and placing VMs by considering factors like resource demands and VM migration strategies. By leveraging the collective intelligence of these optimisation methods, the OGrA+OFr algorithm achieves superior VM placement and migration decisions, ultimately leading to a significant reduction in PC.

In the case of the OGrA+OFr, 'SLA-V OGrA+OFr records a notably low SLA violation of 0.085716074, showcasing its effectiveness in meeting service-level requirements. This metric, 'SLA-V OGrA+OFr', represents the degree to which the OGrA+OFr algorithm adheres to the predefined service level agreement (SLA). In contrast, 'SLA-V Talwani et al. (2022)', (0.094696604), 'SLA-V Kansal and Chana (2016)' (0.094192428), and 'SLA-V Singh and Singh (2021)' (0.093157854) show slightly higher SLA violation values, indicating that these algorithms may experience relatively more significant deviations from the specified SLAs. The OGrA+OFr algorithm excels in directly minimising SLA violations, underscoring its ability to maintain service quality while optimising PC and resource allocation.

Table 3 Comparative analysis of PC

Total number of PMs	*Total number of VMs*	*Power consumption in kW OGrA+OFr*	*Power consumption in kW Talwani et al. (2022)*	*Power consumption in kW Kansal and Chana (2016)*	*Power consumption in kW Singh and Singh (2021)*
10	50	8.20241825	9.41036164	9.40947362	9.26905356
20	100	8.44126569	8.56809205	8.64360942	8.70471717
30	150	8.647098	9.10156469	9.3189534	8.67120337
40	200	8.8920332	8.97634621	10.0730238	9.3227766
50	250	9.06739081	10.20679	10.1552512	9.35812619
60	300	9.25533748	10.0228888	10.8907078	9.51590199
70	350	9.43958284	10.0930988	10.4301495	9.8674329
80	400	9.6215538	11.1911705	11.1450082	11.0378167
90	450	9.8212016	10.8792815	10.6537092	11.0515121
100	500	10.2049953	11.5245482	11.9843064	11.1810021
110	550	10.2772765	11.0913943	11.252443	11.1234703
120	600	10.6015424	11.9280635	10.7312915	11.0877528
130	650	10.7257006	12.4941373	10.7279229	11.6185171
140	700	11.0526153	11.9946018	11.3564497	11.7312727
150	750	11.1078118	11.2899846	12.2598884	12.4741325
160	800	11.4837127	11.9720174	11.7305129	12.7390514
170	850	11.7539896	12.2965465	13.0518997	12.9858648
180	900	11.8876926	13.0532678	13.2727998	13.0514914
190	950	12.0129771	12.8039946	13.6639306	13.3707
200	1,000	12.2651721	14.1663371	13.3355978	13.1338179
210	1,050	12.59634	13.1468172	13.599417	14.1558084
220	1,100	12.6333702	13.247967	13.0814719	14.5019697
230	1,150	12.7242109	14.5345263	14.8984566	13.474453
240	1,200	13.3305686	14.7415385	13.7018972	13.8101399
250	1,250	13.0851828	14.683635	14.2514896	13.532014

Table 4 SLA-V comparison

Total number of PMs	*Total number of VMs*	*SLA-V OGrA+OFr*	*SLA-V Talwani et al. (2022)*	*SLA-V Kansal and Chana (2016)*	*SLA-V Singh and Singh (2021)*
10	50	0.08403747	0.09401515	0.09188212	0.0987554
20	100	0.08419547	0.09679518	0.0963706	0.08511192
30	150	0.08473805	0.08728784	0.08635815	0.09041836
40	200	0.08442284	0.09274834	0.09874877	0.09077092
50	250	0.08416002	0.09929489	0.0867523	0.08465388
60	300	0.08528328	0.09518106	0.09105944	0.08821415
70	350	0.08450827	0.09389514	0.08854359	0.09704826
80	400	0.08452146	0.09356391	0.08486396	0.09099129
90	450	0.08557475	0.09541033	0.08609236	0.0866346
100	500	0.08481286	0.08642543	0.09413995	0.0967023
110	550	0.08640727	0.0944647	0.10108373	0.09632576
120	600	0.08467212	0.09485055	0.09754149	0.08992207
130	650	0.08537913	0.09246261	0.09721514	0.09033496
140	700	0.08612586	0.08909842	0.09757344	0.08989068
150	750	0.08604972	0.09264382	0.09603147	0.09608064
160	800	0.08580055	0.09288515	0.09602369	0.09769692
170	850	0.08629814	0.09981152	0.09041113	0.091239
180	900	0.08724471	0.09545	0.10284989	0.09067882
190	950	0.08766691	0.10238738	0.1030168	0.09069712
200	1,000	0.08735716	0.09901291	0.09553461	0.09282469
210	1,050	0.08588595	0.09726938	0.09198792	0.0964512
220	1,100	0.08636457	0.10016552	0.09244575	0.09832093
230	1,150	0.08785977	0.09479683	0.10104138	0.10001901
240	1,200	0.08644173	0.09809122	0.09928741	0.09873317
250	1,250	0.08709379	0.08940782	0.08795561	0.10043031

Table 5 Number of migrations

Total number of PMs	*Total number of VMs*	*Number of migrations OGrA+OFr*	*Migration count Talwani et al. (2022)*	*Migration count Kansal and Chana (2016)*	*Migration count Singh and Singh (2021)*
10	50	14	16	17	17
20	100	28	30	31	31
30	150	39	41	42	45
40	200	72	77	77	76
50	250	81	83	85	82
60	300	86	89	90	94
70	350	125	126	128	131
80	400	114	117	116	119
90	450	132	136	139	133
100	500	150	151	152	151
110	550	195	207	209	197
120	600	194	198	197	211
130	650	212	214	215	219
140	700	243	262	266	244
150	750	242	269	272	244
160	800	239	241	242	251
170	850	235	243	245	236

Table 5 Number of migrations (continued)

Total number of PMs	*Total number of VMs*	*Number of migrations OGrA+OFr*	*Migration count Talwani et al. (2022)*	*Migration count Kansal and Chana (2016)*	*Migration count Singh and Singh (2021)*
180	900	234	236	237	246
190	950	252	255	255	267
200	1,000	265	274	275	266
210	1,050	347	335	348	349
220	1,100	330	354	355	333
230	1,150	323	351	353	326
240	1,200	338	340	343	343
250	1,250	408	421	423	410

Figure 3 Improvement graph for SLA (see online version for colours)

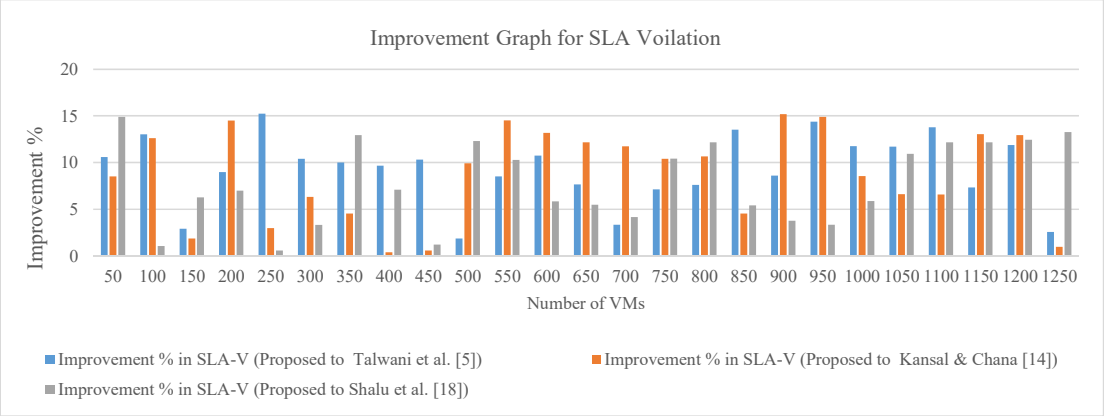
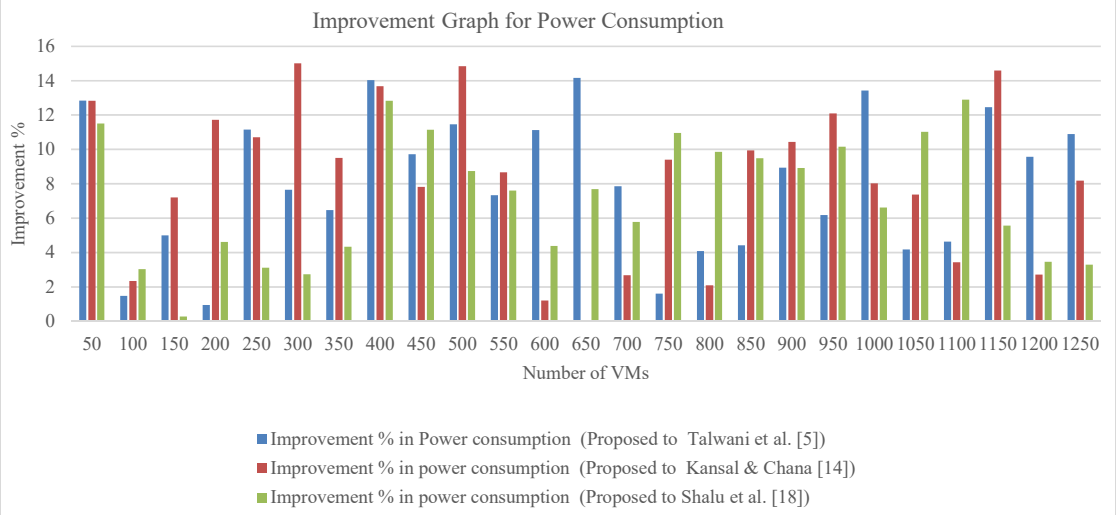


Figure 4 Improvement in PC (see online version for colours)



The 'average migration count OGrA+OFr', with an average value of 191.64, highlights the efficiency of the OGrA+OFr algorithm in minimising the total number of VM migrations for a scenario involving 1,250 VMs. This metric represents the average count of VM migrations required during the optimisation process. In contrast, 'average migration count Talwani et al. (2022)' (205.08), 'average migration count Kansal and Chana (2016)' (204.04), and 'average migration count Singh and Singh (2021)' (200.64) exhibit slightly higher average migration counts. This suggests that these alternative algorithms may require more migrations on average to achieve resource optimisation. The OGrA+OFr algorithm's ability to reduce the average migration count highlights its effectiveness in resource allocation and VM placement, making it a compelling choice for minimising disruptions in large-scale virtualised environments.

Figure 5 Improvement in number of migration (see online version for colours)

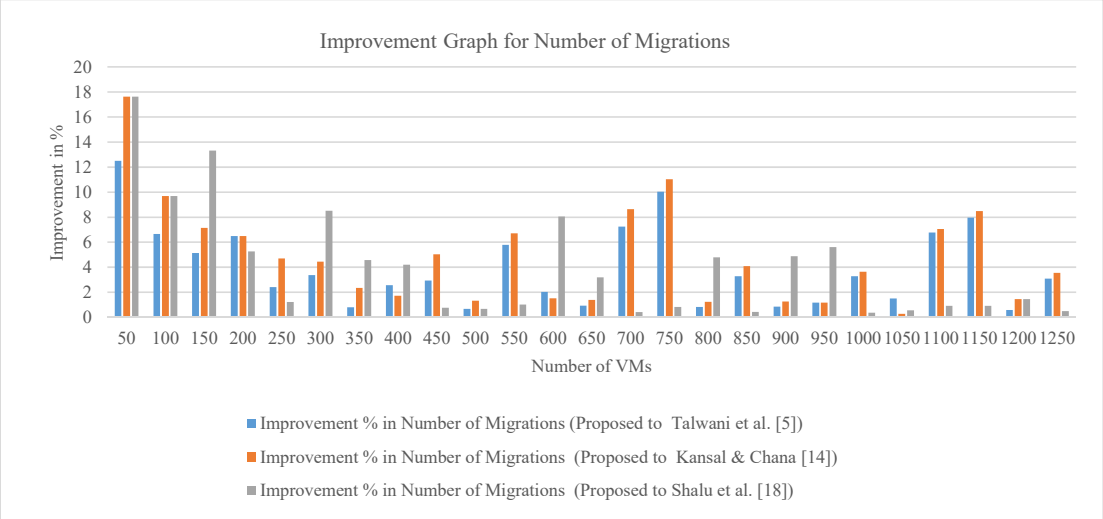


Figure 3 visually illustrates the substantial improvements achieved by the OGrA+OFr optimisation algorithm in mitigating service level agreement violation (SLA-V) when compared to two existing algorithms, Talwani et al. (2022) Kansal and Chana (2016) and Singh and Singh (2021). This graphical representation presents a comprehensive view of the algorithm's performance across various scenarios, including different numbers of PMs and VMs. The vertical axis represents the percentage improvement in SLA-V, which ranges from approximately 0.4% to 15%. Each data point on the graph corresponds to a specific scenario, making it evident that the OGrA+OFr algorithm consistently enhances the QoS delivery, thereby reducing SLA breaches. These improvements hold significant implications for ensuring a reliable and robust CC environment, particularly when accommodating diverse workloads and resource demands. Figure 4 provides a visual depiction of the positive impact of the OGrA+OFr optimisation algorithm on PC reduction in comparison to Talwani et al. (2022), Kansal and Chana (2016) and Singh and Singh (2021). This graphical representation further emphasises the algorithm's effectiveness in optimising power utilisation within CC environment.

The vertical axis represents the percentage improvement in PC, which spans from approximately 0.9% to 15%. Each data point on the graph corresponds to a specific scenario involving varying numbers of PMs and VMs. The upward trendline in Figure 4 underscores that the OGrA+OFr algorithm consistently achieves improved power efficiency. These improvements have dual benefits, contributing to environmental sustainability by reducing energy consumption and generating cost savings for cloud service providers. The ability to maintain high-quality service while minimising resource waste aligns with contemporary green computing initiatives and enhances the overall efficiency of cloud infrastructure. In Figure 5, a comparative analysis of the improvement percentages over the proposed OGrA+OFr method as discussed over the three different methods. Three different papers, namely Talwani et al. (2022), Kansal and Chana (2016), and Singh and Singh (2021), have proposed strategies to enhance the efficiency of OGrA+OFr in reducing the number of migrations. The improvement percentages, depicted in the graph, demonstrate the variability in effectiveness across these methods. Talwani et al. (2022) exhibit improvements ranging from 0.662% to 12.5%, while Kansal and Chana (2016) show enhancements ranging from 1.28% to 17.65%. Similarly, Singh and Singh (2021) present improvements ranging from 0.423% to 13.33%. These findings underline the diverse approaches taken to mitigate migrations in OGrA+OFr and highlight the potential for significant efficiency gains.

7 Conclusions

In this paper, a novel optimisation algorithm has been introduced to enhance resource allocation and management within CC environments. The approach leverages the Grasshopper optimisation algorithm for VM selection and the OFr algorithm for PM selection, with the aim of optimising critical QoS parameters while effectively addressing concerns related to SLA-V and PC. The integration of the OGrA for VM selection represents a notable innovation in this research. This phase plays a pivotal role in identifying the most suitable VMs for migration, considering parameters such as CPU utilisation and PC. The evaluation demonstrates that the OGrA+OFr algorithm consistently outperforms existing solutions in terms of SLA-V, achieving substantial improvements ranging from approximately 0.4% to 15%. This highlights the algorithm's robustness in elevating the QoS delivery, a vital aspect in CC. Complementing the VM selection, the algorithm leverages the OFr algorithm for PM selection. This phase is pivotal in ensuring that VMs are strategically placed on PMs capable of efficiently accommodating their resource demands. This leads to a substantial reduction in PC and an overall enhancement in system performance. The OGrA+OFr consistently reduces PC, with improvements

ranging from approximately 0.9% to 15%, aligning seamlessly with green computing initiatives and presenting cost-saving opportunities for cloud service providers. The comprehensive evaluation underscores the algorithm's versatility across diverse scenarios, encompassing varying numbers of PMs and VMs. The consistent positive trend in improvement percentages across these scenarios reaffirms the algorithm's reliability and adaptability in diverse CC environments.

In comparison to existing algorithms, including those developed by Talwani et al. (2022), Kansal and Chana (2016), and Singh and Singh (2021), the OGrA+OFr algorithm emerges as a promising solution. It adeptly addresses the complexities of VM selection and PM placement, aligning itself with contemporary green computing initiatives and ensuring cost-effective resource management. The OGrA+OFr algorithm consistently reduces PC across various scenarios, with improvements ranging from approximately 0.9% to 15%. This signifies its efficiency in resource allocation and environmental sustainability efforts. In the evaluations, the OGrA+OFr algorithm exhibits a notable reduction in the total number of migrations required, minimising operational disruptions and resource overhead. In comparison to the algorithm developed by Singh and Singh (2021), our OGrA+OFr algorithm showcases significant improvements in SLA-V, underscoring its effectiveness in ensuring consistent service quality. The OGrA+OFr algorithm consistently outperforms Singh and Singh's (2021) algorithm in terms of PC reduction, highlighting its efficiency in optimising resource utilisation.

Disclaimer

Authors have not received any funding for this research work.

Data availability

The data will be made available on request to the corresponding author.

References

Alharbi, M.T., Qawqzeh, Y., Jaradat, A., Nazim, K. and Sattar, A. (2021) 'A review of swarm intelligence algorithms deployment for scheduling and optimization in cloud computing environments', *PeerJ Computer Science*, August, Vol. 7, p.e696, DOI: 10.7717/PEERJ-CS.696.

Beloglazov, A., Abawajy, J. and Buyya, R. (2012) 'Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing', *Future Generation Computer Systems*, May, Vol. 28, No. 5, pp.755–768, DOI: 10.1016/J.FUTURE.2011.04.017.

Chou, L-D., Chen, H-F., Tseng, F-H., Chao, H-C. and Chang, Y-J. (2016) 'DPRA: dynamic power-saving resource allocation for cloud data center using particle swarm optimization', *IEEE Systems Journal*, Vol. 12, No. 2, pp.1554–1565.

Dubey, K. and Sharma, S.C. (2020) 'An extended intelligent water drop approach for efficient VM allocation in secure cloud computing framework', *Journal of King Saud University-Computer and Information Sciences*, Vol. 34, No. 7, pp.3948–3958.

Durairaj, S. and Sridhar, R. (2023) 'MOM-VMP: multi-objective mayfly optimization algorithm for VM placement supported by principal component analysis (PCA) in cloud data center', *Cluster Computing*, June, pp.1–19, DOI: 10.1007/S10586-023-04040-8/METRICS.

Elmagzoub, M.A., Syed, D., Shaikh, A., Islam, N., Alghamdi, A. and Rizwan, S. (2021) 'A survey of swarm intelligence based load balancing techniques in cloud computing environment', *Electronics*, November, Vol. 10, No. 21, p.2718, DOI: 10.3390/ELECTRONICS10212718.

Guarda, T., Augusto, M.F., Costa, I., Oliveira, P., Villao, D. and Leon, M. (2021) 'The impact of cloud computing and virtualization on business', *Communications in Computer and Information Science*, Vol. 1485, pp.399–412, DOI: 10.1007/978-3-030-90241-4_31/COVER.

Kansal, N.J. and Chana, I. (2016) 'Energy-aware virtual machine migration for cloud computing-a firefly optimization approach', *Journal of Grid Computing*, Vol. 14, No. 2, pp.327–345.

Kaur, A., Kumar, S., Gupta, D., Hamid, Y., Hamdi, M., Ksibi, A., Elmannai, H. and Saini, S. (2023) 'Algorithmic approach to virtual machine migration in cloud computing with updated SESA algorithm', *Sensors (Basel, Switzerland)*, July, Vol. 23, No. 13, p.6117, DOI: 10.3390/S23136117.

Koot, M. and Wijnhoven, F. (2021) 'Usage impact on data center electricity needs: a system dynamic forecasting model', *Applied Energy*, June, Vol. 291, p.116798, DOI: 10.1016/J.APENERGY.2021.116798.

Manaswi, D. and Sharma, D.M. (2024) 'Artificial intelligence empowering the digital world', *Futuristic Trends in Artificial Intelligence*, March, Vol. 3, No. 10, pp.83–90, DOI: 10.58532/V3BGAI10P2CH3.

Mehta, S., Kaur, P. and Agarwal, P. (2024) 'Improved whale optimization variants for SLA-compliant placement of virtual machines in cloud data centers', *Multimedia Tools and Applications*, January, Vol. 83, No. 1, pp.149–171, DOI: 10.1007/S11042-023-15528-1/METRICS.

Meshkati, J. and Safi-Esfahani, F. (2019) 'Energy-aware resource utilization based on particle swarm optimization and artificial bee colony algorithms in cloud computing', *Journal of Supercomputing*, May, Vol. 75, No. 5, pp.2455–2496, DOI: 10.1007/S11227-018-2626-9/METRICS.

Mohammed, C.M., Zeebaree, S.R.M. et al. (2021) 'Sufficient comparison among cloud computing services: IaaS, PaaS, and SaaS: a review', *International Journal of Science and Business*, Vol. 5, No. 2, pp.17–30.

Naik, B.B., Singh, D. and Samaddar, A.B. (2020) 'FHCS: hybridised optimisation for virtual machine migration and task scheduling in cloud data center', *IET Communications*, Vol. 14, No. 12, pp.1942–1948.

Panwar, R. and Supriya, M. (2022) 'Dynamic resource provisioning for service-based cloud applications: a Bayesian learning approach', *Journal of Parallel and Distributed Computing*, October, Vol. 168, pp.90–107, DOI: 10.1016/J.JPDC.2022.06.001.

Radi, M., Alwan, A.A. and Gulzar, Y. (2023) 'Genetic-based virtual machines consolidation strategy with efficient energy consumption in cloud environment', *IEEE Access*, Vol. 11, pp.48022–48032, DOI: 10.1109/ACCESS.2023.3276292.

Singh, S. and Singh, D. (2021) 'Artificial neural network-based virtual machine allocation in cloud computing', *Journal of Discrete Mathematical Sciences and Cryptography*, Vol. 24, No. 6, pp.1739–1750.

Singh, H., Tyagi, S., Kumar, P., Gill, S.S. and Buyya, R. (2021) 'Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: analysis, performance evaluation, and future directions', *Simulation Modelling Practice and Theory*, September, Vol. 111, p.102353, DOI: 10.1016/J.SIMPAT.2021.102353.

Singh, S. and Singh, D. (2023) 'A bio-inspired VM migration using re-initialization and decomposition based-whale optimization', *ICT Express*, Vol. 9, No. 1, pp.92–99.

Stillwell, M., Schanzenbach, D., Vivien, F. and Casanova, H. (2010) 'Resource allocation algorithms for virtualized service hosting platforms', *Journal of Parallel and Distributed Computing*, September, Vol. 70, No. 9, pp.962–974, DOI: 10.1016/J.JPDC.2010.05.006.

Szabo, C., Sheng, Q.Z., Kroeger, T., Zhang, Y. and Yu, J. (2014) 'Science in the cloud: allocation and execution of data-intensive scientific workflows', *Journal of Grid Computing*, Vol. 12, No. 2, pp.245–264, DOI: 10.1007/S10723-013-9282-3.

Talwani, S., Singla, J., Mathur, G., Malik, N., Jhanjhi, N.Z., Masud, M. and Aljahdali, S. (2022) 'Machine-learning-based approach for virtual machine allocation and migration', *Electronics*, Vol. 11, No. 19, p.3249.

Tarahomi, M., Izadi, M. and Ghobaei-Arani, M. (2021) 'An efficient power-aware VM allocation mechanism in cloud data centers: a micro genetic-based approach', *Cluster Computing*, Vol. 24, No. 2, pp.919–934.

Tran, C.H., Bui, T.K. and Pham, T.V. (2022) 'Virtual machine migration policy for multi-tier application in cloud computing based on Q-learning algorithm', *Computing*, Vol. 104, No. 6, pp.1285–1306.

Vijaya, C. and Srinivasan, P. (2024) 'Multi-objective meta-heuristic technique for energy efficient virtual machine placement in cloud data centers', *Informatica*, February, Vol. 48, No. 6, pp.1–18, DOI: 10.31449/INF.V48I6.5263.

Wang, Z., Sun, D., Xue, G., Qian, S., Li, G. and Li, M. (2019) 'Ada-Things: an adaptive virtual machine monitoring and migration strategy for internet of things applications', *Journal of Parallel and Distributed Computing*, October, Vol. 132, pp.164–176, DOI: 10.1016/J.JPDC.2018.06.009.

Yadav, M. and Mishra, A. (2023) 'An enhanced ordinal optimization with lower scheduling overhead based novel approach for task scheduling in cloud computing environment', *Journal of Cloud Computing*, December, Vol. 12, No. 1, pp.1–14, DOI: 10.1186/S13677-023-00392-Z/TABLES/8.

Yang, X.S. (2009) 'Firefly algorithms for multimodal optimization', in *International Symposium on Stochastic Algorithms*, October, pp.169–178, Springer Berlin Heidelberg, Berlin, Heidelberg.

Zaffar, A. (2021) 'Modeling and forecasting of sunspots cycles: an application of ARMA (p, q)-GARCH (1, 1) model', *Research Square*, pp.1–17, DOI: https://doi.org/10.21203/rs.3.rs-412946/v1.

Zolfaghari, R. (2024) 'Energy-performance aware virtual machines migration in cloud network by using prediction and fuzzy approaches', *Engineering Applications of Artificial Intelligence*, May, Vol. 131, p.107825, DOI: 10.1016/J.ENGAPPAI.2023.107825.