J Grid Computing (2025) 23:15
https://doi.org/10.1007/s10723-025-09803-8

RESEARCH

CHPSO: An Efficient Algorithm for Task Scheduling and Optimizing Resource Utilization in the Cloud Environment

Hind Mikram · Said El Kafhali

Received: 24 December 2024 / Accepted: 20 March 2025 / Published online: 1 April 2025
© The Author(s), under exclusive licence to Springer Nature B.V. 2025

Abstract In cloud computing, efficient task scheduling is crucial for optimizing system performance and resource utilization. Addressing this, we introduce a novel algorithm designed to enhance task distribution across virtual servers based on their processing capabilities. Leveraging the total MIPS metric, the CHPSO algorithm (Chi-squared Particle Swarm Optimization) provides a comprehensive assessment of computational capacity, which facilitates strategic task allocation to improve the efficiency of cloud-based applications. Our method employs a two-pronged approach: firstly, it models task arrival times using a chi-squared distribution, which creates realistic workload scenarios that reflect variable arrival rates. Secondly, it adopts a heuristic strategy that balances the workload across VSs by considering their cumulative processing times. Additionally, we integrate a PSO algorithm to refine task-to-VS mapping, ensuring optimal task placement. The effectiveness of our algorithm is evaluated using key metrics such as response time, energy consumption, makespan, execution time, and resource utilization. Compared to existing state-of-the-art algorithms, CHPSO demonstrates significant improvements in these metrics, showcasing its capability to manage dynamic and complex environments in cloud computing. Our results indicate that CHPSO not only optimizes performance but also enhances resource efficiency, making it a valuable contribution to the field.

Keywords Cloud computing · Task scheduling · Resource utilization · Makespan · Energy consumption

H. Mikram · S. El Kafhali (✉)
Faculty of Sciences and Techniques, Computer, Networks, Modeling, and Mobility Laboratory (Ir2m), Hassan First University of Settat, B.P. 539, 26000 Settat, Morocco
e-mail: said.elkafhali@uhp.ac.ma

H. Mikram
e-mail: h.mikram@uhp.ac.ma

1 Introduction

Cloud computing represents a paradigm shift in computing, where data and services are hosted in highly scalable data centers deployed in the cloud, and IT services are delivered via the Internet. Traditional self-managed data centers often adopt a strategy of under-provisioning power and cooling infrastructure to achieve short-term utilization goals, assuming that only a fraction of the IT resources will be actively used. However, this approach typically leads to higher relative energy consumption and operational costs due to low server utilization rates [1].

The migration of enterprise IT operations to cloud data center environments offers significant potential benefits, leveraging the scalability and energy efficiency inherent in cloud infrastructure. By consolidating IT activities in large, energy-efficient cloud environments, enterprises can achieve higher levels

of resource utilization and realize economies of scale [2]. Furthermore, large-scale multi-tenant clouds that support homogeneous workloads can leverage statistical multiplexing on an unprecedented scale, accommodating hundreds of thousands of virtual servers while achieving exceptionally high levels of resource utilization and energy efficiency [3].

Moreover, cloud data centers are distributed across geographical locations, enabling ubiquitous access to computing resources and services. The distribution of cloud data centers allows for redundancy and fault tolerance, ensuring high availability and reliability of services [4]. Additionally, the distributed nature of cloud infrastructure facilitates scalability, allowing organizations to dynamically scale their resources based on demand fluctuations [5].

In the pursuit of optimal resource utilization and system performance improvement, load-balancing algorithms are essential components of cloud computing infrastructure. By intelligently redistributing workloads from heavily loaded nodes to underutilized ones, these algorithms not only enhance system performance but also play a crucial role in promoting energy efficiency [6]. Efficient load balancing minimizes the overall energy consumption of cloud data centers by ensuring that resources are utilized effectively and idle resources are minimized. Therefore, load-balancing algorithms contribute significantly to the energy efficiency objectives of cloud computing environments, aligning with the broader goal of achieving sustainability and reducing environmental impact [7].

Meanwhile, probability distributions play a crucial role in modeling various aspects of system behavior and resource utilization in cloud environments. While distributions like the normal and exponential are commonly used [8], they fall short in capturing the bursty and non-stationary nature of task arrivals—often underestimating tail events and variability. This can lead to inefficient resource utilization, increased processing delays, and overall suboptimal performance [9]. In contrast, the chi-squared distribution is frequently employed in statistical analysis and hypothesis testing [10] and is particularly valuable in cloud computing. It accurately models phenomena such as task arrival and processing times by capturing the inherent variability and stochastic nature of these events, thereby enabling better resource allocation and management decisions [11].

In this study, we propose a novel approach for task assignment and virtual server (VS) placement in cloud computing environments. Our methodology begins by adjusting incoming tasks using chi-squared distribution. This distribution occasionally generate larger jumps that help particles escape local minima, while also providing frequent, smaller updates that refine the swarm's position—thanks to its non-negative, right-skewed nature [12]. Additionally, the chi-squared distribution with k degrees of freedom allows to control the magnitude of the random variations more precisely than a purely uniform approach. This flexibility can be used to better calibrate the balance between exploration (larger moves) and exploitation (finer local search). Furthermore, recognizing that VSs differ in processing capabilities (e.g., MIPS) [13], our approach allocates tasks to the servers offering the shortest projected processing times, thereby avoiding hotspots or underutilization. To improve the accuracy of these estimates, the processing times of the VSs are dynamically updated based on the characteristics of the assigned tasks. Finally, we incorporate a Particle Swarm Optimization (PSO) algorithm to further refine the task assignment process, optimizing VS placement and enhancing system performance [14].

The primary objective of our proposed algorithm is to effectively distribute tasks among VSs, considering their processing capabilities. To achieve this, we employ the total MIPS metric as a comprehensive measure of the overall computational capacity, ensuring that tasks are allocated in a way that optimizes system performance and guarantees efficient execution of cloud-based applications. Our approach is built on three key contributions that work together synergistically:

- We introduce the chi-squared distribution to model the arrival times of cloudlets. The chi-squared distribution is non-negative and right-skewed, which makes it ideal for capturing the burstiness and variability of real-world workloads. This realistic modeling allows our approach to adapt to changing arrival rates and better reflect actual cloud environments.
- We employ a heuristic approach that balances the workload across VSs by considering their cumulative processing times. At each iteration, the cloudlet is assigned to the VS for which the addi-

tion minimizes the deviation from the estimated balanced execution time. This heuristic ensures that workload distribution across VSs is as even as possible. Key steps in this approach include,

- Calculating the total lengths of cloudlets and the total MIPS of VSs.
- Estimating an overall execution time based on the aggregate workload and available computational capacity.
- Iteratively assigning cloudlets to VSs based on processing time comparisons, aiming to minimize the overall execution time.

- Furthermore, we integrate the PSO algorithm to optimize the mapping of cloudlet-to-VS, ensuring that each task is placed in the most suitable computational environment.

Each of these contributions addresses different aspects of the scheduling challenge. The chi-squared distribution models the inherent variability in task arrivals, the heuristic ensures an effective initial distribution by balancing the load across heterogeneous VSs, and the PSO algorithm optimizes the task assignment further by exploring and refining the solution space.

The remainder of this paper is structured as follows: Sect. 2 reviews relevant literature in the field. Section 3 details the scheduling process and frames the problem within the context of a single-objective optimization model. Section 4 introduces our proposed hybrid approach, providing an in-depth analysis of its complexity. Section 5 discusses the experimental results and their implications. Limitations and future directions are presented in Sect. 6. Finally, Sect. 7 concludes with a summary of key findings and outlines potential directions for future research.

2 Related Work

In recent research, various methodologies have been explored for task scheduling within cloud computing infrastructures, each offering distinct advantages and limitations. This section delves into several of these approaches, shedding light on their respective merits and shortcomings. In the realm of cloud computing, multi-objective optimization is critical for enhancing system performance and efficiency. The authors of [15] addressed the issue of multi-objective task scheduling using a Genetic Algorithm and an Energy-Conscious Scheduling Heuristic (GAECS). Their goal was to optimize task scheduling by reducing makespan—the total time taken to complete all tasks—and minimizing energy consumption. The process involved prioritizing tasks, assigning them to processors, and employing an energy-conscious model. However, the GAECS approach involved generating and evaluating populations of potential solutions (chromosomes) iteratively. As the number of tasks and processors grows, the time required for each generation increases significantly.

The author of [16] presented Time Varying Inertia Weight Particle Swarm Optimization (TVIW-PSO), an enhanced version of PSO that incorporated a dynamic inertia weight. This modification aimed to balance exploration and exploitation of the solution space, leading to improved scheduling outcomes. Although the paper suggested that the TVIW-PSO method enhanced scalability, its real-world performance in large-scale cloud environments remained insufficiently validated. The effectiveness of the approach in handling a vast number of tasks and resources may not be fully established, which could limit its applicability in larger cloud infrastructures.

Several studies have explored task scheduling in fog computing, aiming to reduce service delays and energy consumption while maintaining task priorities and deadlines. In [17], the authors proposed a hybrid Particle Swarm Optimization and Simulated Annealing (PSO-SA) algorithm. Their approach prioritized tasks while optimizing the fitness function, leading to improved workflow execution for IoT devices. The study addressed key challenges such as heterogeneous fog networks and dynamic task arrivals, which had not been previously considered in conjunction with deadline and priority constraints. Although the primary focus of their work was to optimize energy consumption while ensuring task deadlines were met, the inclusion of workflow dependencies and priority constraints added significant complexity to the scheduling process. Balancing these factors required additional computational overhead, which may limit the algorithm's scalability in larger, more dynamic environments.

In [18], the authors addressed the challenge of task scheduling in fog computing for big data applications

by proposing a multi-objective hybrid GA-PSO algorithm. Their approach combined the global exploration capabilities of GA with the rapid convergence of PSO to effectively manage the inherent trade-offs in multi-objective optimization. The Hybrid GA-PSO algorithm delivered superior performance by avoiding local optima through the combined strengths of GA's exploration and PSO's exploitation. However, the study also acknowledged several limitations. The complexity introduced by the hybrid approach increased computational overhead, and uncertainties in task allocation across fog nodes remained unresolved. Additionally, the security and privacy protocols for handling sensitive large-scale data in fog environments require further refinement.

The authors of [19] proposed a hybrid framework that combined the Arithmetic Optimization Algorithm (AOA) for task scheduling with Markov chain-based load prediction to anticipate resource demands. This approach aimed to optimize task-to-resource mapping by leveraging the combined resources of fog and cloud layers, thereby reducing service delays, and makespan while ensuring that workload imbalances were minimized. Comprehensive simulations demonstrated that this hybrid method outperformed other metaheuristic techniques, such as PSO, GA, and Grey Wolf Optimization (GWO), especially in scenarios with high data volumes and dynamic fog node conditions. However, the approach also revealed limitations; the computational complexity of both the AOA and the Markov chain-based prediction posed scalability challenges, and the accuracy of load predictions was susceptible to sudden fluctuations in workload.

The diversity of user requirements plays a crucial role in resource allocation. Different jobs demand varying types of resources, such as CPU, memory, and storage, and they also differ in execution time. This variability makes scheduling more complex, requiring algorithms that can adapt to changing user demands. In [20], the authors proposed resource allocation method for cloud datacenters based on the Asynchronous Advantage Actor-Critic (A3C) algorithm. Their approach aimed to enhance both Quality of Service (QoS) and energy efficiency in dynamic environments by accounting for fluctuating resource availability and workload demands. The study highlighted the importance of adapting to real-time changes within cloud datacenters, including variations in resource usage, job requests, and overall system performance. However, the model assumed that job duration is known advance, which may not always be realistic in practical scenarios. In practice, users may not always specify the exact duration of their jobs, leading to potential inaccuracies in scheduling and resource allocation.

Accurate workload prediction is another critical factor in cloud resource management, as it enables cloud service providers (CSPs) to allocate resources efficiently while meeting user demands. The study in [21] identified two main challenges in workload prediction: high variance and high dimensionality, both of which can impact resource utilization and service level agreement (SLA) compliance. To tackle these challenges, the authors introduced the Deep Learning based Prediction Algorithm for cloud Workloads (L-PAW). This algorithm combined a top-sparse auto-encoder (TSA) to reduce data dimensionality with a gated recurrent unit (GRU) to capture long-term dependencies in workload patterns. While L-PAW demonstrated strong performance with highly auto-correlated and periodic workloads, it struggled with highly unpredictable workloads. The paper acknowledged that workload variability remains a challenge, suggesting that the model may not be universally effective across all cloud environments.

The dynamic nature of Mobile Edge Computing (MEC) environments, where system states and user demands frequently changed, posed significant challenges for resource allocation and computation offloading. To address these issues, the authors in [22] proposed a Personalized Federated Deep Reinforcement Learning-based Computation Offloading and Resource Allocation method (PFR-OA). This approach was designed to accommodate the personalized needs of users in smart communities, enabling more efficient policies for computation offloading and resource management. One of the key contributions of PFR-OA was the introduction of a new proximal term for optimizing local Q-value loss functions, which improved model training efficiency. Additionally, the method incorporated a partial-greedy participant selection mechanism that enhanced global model convergence and reduced the complexity of federated aggregation. These innovations made the approach more adaptive to the decentralized nature of federated learning in MEC environments. However, despite the advantages of federated learning in enabling decentralized data processing, PFR-OA faced challenges

related to data heterogeneity among users. Variations in data quality and quantity affected the performance of the global model, potentially leading to inconsistencies in decision-making across different edge nodes. This limitation suggested the need for further refinements to enhance the robustness of federated learning approaches in diverse MEC settings.

Workflow scheduling in distributed environments, particularly in cloud and fog computing, has been widely studied due to its complexity and impact on system performance. Several recent studies have explored heuristic-based and AI-driven approaches to improve scheduling efficiency while addressing key challenges such as resource utilization, execution time, and cost management. In [23], the authors introduced a taxonomy that categorizes the critical challenges in existing scheduling methods and provided a systematic overview of various techniques, highlighting their strengths and limitations. Their findings revealed that heuristic-based mechanisms are employed in 25% of the surveyed algorithms, while 75% incorporate AI-based and parametric modeling approaches. Makespan emerged as the most frequently addressed parameter, underscoring the emphasis on execution time optimization. While this survey offers valuable insights into existing methods and their trade-offs, it also highlights open research challenges, such as improving scalability, reliability, and security in scheduling algorithms.

Adding to the complexity of optimizing energy usage, the authors of [24] proposed the Deadline-constrained Energy-aware Workflow Scheduling algorithm, which included task sequencing, VND-based data center searches, task sequence adjustment, and VS searching with Dynamic Voltage Frequency Scaling (DVFS). The incorporation of DVFS aimed to reduce additional costs for service providers by optimizing energy consumption. However, this algorithm faced challenges related to computational complexity and increased execution time due to the integration of multiple optimization steps and dynamic adjustments.

Additionally, the RAFL framework proposed by [25] is a hybrid metaheuristic-based resource allocation framework that uses the PPSO-DA hybrid optimization algorithm to balance loads across active PSs and resource capacities in cloud computing environments. It efficiently explored the search space, utilizing the self-learning and social-learning approaches of PPSO and the diversification features of DA to avoid local optima. RAFL generated optimal VS placement plans to minimize load imbalance across PSs and resource capacities. However, its effectiveness depends on parameter settings and configurations, which can lead to suboptimal solutions or increased computational complexity. The scalability of RAFL to large-scale cloud environments is also a concern.

Furthermore, the authors of [26] contributed to the field with their proposal of a power-aware virtual machine allocation system (EALBPSO) for cloud data centers. This system introduced novel fitness functions aimed at reducing overall energy consumption and maximizing processor load balancing. An evolutionary algorithm-based approach was presented to efficiently converge to an optimal VS-to-PS mapping, while a new optimization technique, EALBPSO, was introduced for initial VS placement, focusing on minimizing power consumption and optimizing load balancing. However, the complexity of the method, scalability challenges, dependence on PSO performance, and real-world implementation obstacles remain areas of consideration.

The Adaptive Scheduling Algorithm Based Task Loading (ASA-TL) proposed by [27] offered several advantages. It efficiently distributed tasks across all available virtual servers, ensuring the security of the cloud data center from overloading. This was achieved through the fair assignment of tasks based on the value and state of the digital devices, optimizing resource utilization. Additionally, Task Loading (TL) effectively managed task requests, ensuring an even distribution among numerous processors. However, the absence of a comparison with hybrid algorithms limited the comprehensive understanding of ASA-TL's constraints. Without such comparisons, it was challenging to ascertain whether ASA-TL represented the optimal solution for all scenarios or if there were specific use cases where hybrid approaches might outperform it. Additionally, the algorithm's effectiveness heavily relies on accurate task assignment and load balancing mechanisms, which may require further optimization to handle diverse and unpredictable cloud environments effectively.

In related research, the authors of [28] developed a cooperative resource provisioning strategy for Multi-Cloud load balancing tailored for Continuous Writing Applications (CWA). This strategy was reported to enhance operational efficiency through

optimized resource distribution across multiple cloud providers and improved cost-effectiveness. It also offered increased scalability and flexibility for handling fluctuating workloads. The approach significantly reduced risks associated with vendor lock-in and improved disaster recovery capabilities by geographically dispersing data and services. Despite its benefits, the implementation faced challenges such as increased complexity in managing resources across diverse platforms, which raised overhead costs and complicated integration efforts. Moreover, security and compliance became more challenging as data spread across different regulatory environments, necessitating stringent management and protection measures. Additionally, inconsistencies in network performance and potential latency issues were observed, which could negatively impact application performance.

The chi-squared distribution has been employed in various algorithms to enhance feature selection and improve the detection accuracy in machine learning models. This statistical approach is particularly valuable in intrusion detection systems where distinguishing between normal and abnormal behavior is critical. Among the notable implementations is the human-in-the-loop mechanism, which integrates human judgment to refine the intrusion detection process in machine learning systems. In [29], the human-in-the-loop mechanism was utilized to address the challenges of intrusion detection. Researchers employed Pearson's Chi-Squared Distribution for optimal feature selection and coupled it with a Boost Decision Tree Classifier to classify nodes in network environments as normal or abnormal. This mechanism not only improved the intrusion detection rate but also minimized the detection time, which is crucial for timely responses to security threats. The approach efficiently managed the load across fog nodes, thereby enhancing the security of cloud data centers against overloads. Utilizing Pearson's Chi-Squared Distribution for optimal feature selection and a Boost Decision Tree Classifier for node classification, this method has demonstrated substantial improvements in detection rates and reduced false positives, as validated by simulations using the KDD Cup 1999 Data Set. The mechanism effectively integrates human judgment to refine detection accuracy and manage false alarms, ensuring swift isolation of intrusive nodes. Despite its benefits, the model's reliance on continuous human involvement and its performance in dynamic real-world scenarios remain areas for further investigation.

In this genomic study, researchers developed a model to predict viral hosts, employing chi-squared features along with other statistical and entropy-based features to extract relevant information from genomic data. The model utilized an ensemble of classifiers, including SVS, NN, RF, and an optimized CNN, demonstrating the robustness of combining chi-squared features with advanced machine-learning techniques to achieve high prediction accuracies. This approach highlighted the chi-squared distribution's utility in handling complex, high-dimensional datasets, making it a valuable tool across various scientific fields. The study achieved a prediction accuracy of approximately 97%, significantly improving upon existing models. Such applications underscored the versatility and effectiveness of the chi-squared distribution in diverse analytical contexts. However, the accuracy of such predictive models heavily depended on the quality and diversity of the dataset used. Any limitations in the dataset, such as bias or insufficient representativeness, could undermine the model's effectiveness and generalizability [30]. In the broader context of cybersecurity within IoT-enabled networks, efficient and fast intrusion detection systems (IDS) are essential. While our study focuses specifically on the application of the chi-squared distribution, other significant advancements include hybrid feature selection schemes that integrate multiple statistical tests. For example, one notable approach combines Chi-Square (χ^2), Pearson's Correlation Coefficient (PCC), and Mutual Information (MI) with the NSGA-II metaheuristic for optimizing feature selection [31]. These findings not only underscore the effectiveness of combining statistical techniques for feature optimization but also set a new benchmark for IDS in IoT environments. Such studies highlight the diverse potential of statistical techniques, including the ones our research builds upon, to significantly improve IDS efficiency.

A comprehensive overview of various task-scheduling strategies employed in cloud environments is presented in Table 1. This table provides a detailed comparison of the methodologies, metrics, advantages, and limitations of each approach.

Among the previous research, we find that some methods have extensively used the chi-squared

Table 1 Comparative analysis of existing works

Work	Algorithms	Metrics	Advantages	Limitations
[15]	(GAECS): Genetic Algorithm and Energy-Conscious Scheduling Heuristic	Energy consumption, makespan	It considers the combination of time and energy	The suboptimal overall performance in certain scenarios is caused by the hybridization of the method
[16]	TVIW-PSO: Time Varying Inertia Weight Particle Swarm Optimization	Execution Time, Load Balancing, Resource Utilization, cost efficiency, and scalability	This method enhances the traditional PSO by incorporating a time-varying inertia weight to improve scheduling outcomes	The effectiveness of the approach in handling a vast number of tasks and resources may not be fully established, which could limit its applicability in larger systems
[17]	a hybrid PSO–SA	Makespan, deadline satisfaction, deadline violation, energy consumption	The fitness function optimization while prioritizing tasks, ensuring that task deadlines are met, and reducing energy consumption	The inclusion of workflow dependencies and priority constraints significantly increased the complexity of the scheduling process
[18]	A hybrid GA and PSO	Execution time, response time, and completion time	The approach optimizes multi-objective task scheduling in fog environments	The paper lacks an extensive analysis of how the algorithm scales with increasing data loads or a growing number of fog nodes
[19]	Arithmetic Optimization Algorithm (AOA)	Delay, Makespan, and execution time	The approach optimizes task-to-resource mapping by leveraging the combined resources of fog and cloud layers, thereby reducing service delays, and makespan while ensuring that workload imbalances were minimized	The accuracy of load predictions was susceptible to sudden fluctuations in workload
[20]	Actor-Critic based Compute-Intensive Workload Allocation Scheme (A3C)	Latency and job dismissing rate with energy-efficiency	Load balancing optimization ensures QoS while reducing energy consumption	The approach assumes that the job duration is known beforehand, which may not always be realistic in practical scenarios
[21]	Deep Learning based Prediction Algorithm for cloud Workloads (L-PAW)	Average training time, Prediction accuracy, Performance display	The method combines TSA with a GRU block within an RNN structure to effectively capture long-term memory dependencies from historical workloads, enabling adaptive and precise predictions for highly variable cloud workloads	The variations in workloads can complicate accurate predictions, indicating that the model may not be universally effective across all workload type as mentioned in the paper
[22]	Personalized Federated deep Reinforcement learning based computation Offloading and resource Allocation method (PFR-OA)	Task execution success rates, delay and energy consumption	This method aims to address the personalized needs of users in smart communities, enabling the development of more efficient policies for computation offloading and resource allocation	It may still struggle with data heterogeneity among users, which can impact the performance of the global model, particularly when there are significant variations in data quality or quantity

Table 1 (continued)

Work	Algorithms	Metrics	Advantages	Limitations
[24]	DEWS: Deadline-constrained Energy-aware Workflow Scheduling	Relative Percentage Deviation (RPD), Resource Utilization (RU)	Reduced Electricity Costs, considering data transmission and dynamic electricity	Increased Execution Time caused by including VND-based searching and task sequence adjustments
[25]	RAFL: Resource allocation framework for load balancing	Mean Resource capacity imbalance, Mean Load imbalance	Anticipate workload changes and proactively distribute VSs across PSs to prevent overloading and underutilization	It might need further adaptation or fine-tuning for different cloud resource management scenarios
[26]	Energy-Aware Load Balancing Particle Swarm Optimization on processors equipped with DVFS (EALBPSO)	Energy consumption, Migration rate, Load distribution	Decrease energy consumption and ensure a fair workload distribution by minimizing physical activity	The method's effectiveness may be hindered by the low number of iterations in PSO
[27]	Adaptive Scheduling Algorithm Based Task Loading (ASA-TL)	Response time, processing time, and overall expense rate	Data assignment is based on the significance and status of digital devices	The algorithm's effectiveness heavily relies on accurate task assignment and load-balancing mechanisms
[28]	Optimal user Scheduling for MultiCloud (OSMC)	Number of users, resource utilization, cost per user	Improve the distribution of computational tasks among multiple cloud platforms, leveraging the diverse capabilities and costs of different cloud services	Complexity in managing resources across diverse platforms; may increase overhead costs
[29]	Intrusion behavior in fog computing	intrusion detection rate, data transmission ratio, identification time, and prediction rate	• It considers multiple factors: fog node availability, reliability, data freshness, and active nodes	It is less scalable for large-scale deployments
[30]	Genomic Host Prediction	Prediction accuracy, feature extraction quality	Utilized chi-squared features and an ensemble of classifiers to achieve high accuracy (97%) in genomic prediction	Accuracy heavily depends on data quality and diversity
[31]	Hybrid Feature Selection IDS	Feature reduction, prediction accuracy	Combined Chi-Square, PCC, MI with NSGA-II to optimize IDS feature selection, achieving 99.48% accuracy	Effective but complex, requiring careful tuning and data management

distribution for feature extraction across various domains. We chose to apply it in the cloud computing environment. Drawing inspiration from these studies, our research underscores the critical role of accurately modeling the arrival times of tasks in cloud environments. This is particularly pivotal given the dynamic nature of cloud services, which often exhibit highly variable and unpredictable task arrival times. Traditional statistical models, which typically assume uniformity, fall short of capturing the complex and stochastic nature of real-world cloud computing workloads. Our approach, therefore, introduces a novel application of the chi-squared distribution to more accurately represent these arrival times, a method not previously employed in cloud computing research. Even though the concept of modeling arrival times is not new, we simplify it by employing the chi-squared distribution. This simple yet effective change can significantly enhance task execution without the need for overly complex algorithms. By leveraging this straightforward statistical approach, we can achieve substantial improvements in accurately predicting and managing task arrivals, ultimately optimizing the performance and efficiency of cloud computing systems.

In conjunction with this statistical modeling, we implement a heuristic strategy to balance the workload across VSs. This begins with an initial calculation of the total lengths of cloudlets and the total MIPS of VSs to estimate the overall computational demand. Cloudlets are then iteratively assigned to VSs based on processing time comparisons, aiming to minimize the total execution time. The advantage of using processing time comparisons lies in their ability to provide a more accurate and real-time assessment of task allocation efficiency. By considering the processing capabilities of each VS and the specific requirements of each cloudlet, we ensure that tasks are distributed in a way that maximizes resource utilization and minimizes delays.

Furthermore, we enhance this task mapping through the integration of PSO. This optimization technique explores various configurations to optimize cloudlet-to-VS assignments, ultimately striving to achieve the best possible performance metrics. The integration of PSO allows for continuous refinement of task assignments, adapting to changes in workload and system performance, which further enhances the overall efficiency and responsiveness of the cloud computing environment.

3 Scheduling Problem Formulation

Cloud computing offers a pay-as-you-go, on-demand business model, where users may request the creation and termination of an unlimited number of virtual machines based on their needs. Consequently, VSs are dynamically created and terminated in cloud data centers, leading to resource fragmentation on servers and resulting in degraded server resource utilization. VS consolidation serves as a strategic tool to address these issues in virtualized data centers, aiming to pack active VSs into the minimum number of physical machines while considering multidimensional resource demands to save energy and maximize server resource utilization.

Building upon PSO, a swarm intelligence algorithm inspired by the behavior of particles in nature, our approach extends our preliminary findings presented in [11]. PSO has gained significant popularity as an effective optimization technique across various domains. To the best of our knowledge, our current methodology represents the first attempt to integrate the chi-squared distribution with PSO for addressing scheduling problems in cloud computing environments, a novel expansion of the concepts initially explored in our earlier work. We now proceed to formally define the scheduling problem and introduce our novel CHPSO algorithm, specifically tailored to address this complex challenge.

3.1 Background

Scheduling solutions for virtual machine consolidation in cloud computing environments aim to optimize the placement of virtual machines onto physical servers to achieve better resource utilization and energy efficiency. These solutions typically involve algorithms that analyze the resource demands of virtual machines, the capacities of physical servers, and the current workload distribution to determine the most efficient assignment of virtual machines to servers.

We investigate task scheduling within a heterogeneous cloud data center composed of various Physical Servers denoted as $\boldsymbol{PS} = \{\boldsymbol{S}1, \boldsymbol{S}2, ..., \boldsymbol{Sn}\}$ and tasks

$TS = \{TS1, TS2, ..., TSm\}$. In addition, Virtual Servers are considered part of the cloud resources. Each machine in the data center offers distinct computational capacities measured in Millions of Instructions Per Second (***MIPS***), which is crucial for understanding the performance capabilities required for efficient task scheduling. Our approach emphasizes resource optimization and aligns task complexity with server capability. By maximizing computational power utilization, we aim to enhance overall system performance, potentially reducing runtime. This approach recognizes that different tasks may have varying computational requirements. Matching these tasks to servers based on their MIPS values allows for optimized performance. For instance, tasks with higher computational demands can be allocated to servers with greater MIPS capacities, ensuring efficient resource utilization and minimizing processing bottlenecks. This strategic allocation of tasks based on computational power not only optimizes system performance but also enhances the scalability and reliability of cloud computing environments.

The datacenter configuration consists of **servers** and **tasks**. Each cloud server, whether physical or virtual, is defined by its computational capabilities, particularly its CPU and memory usage, ensuring optimal utilization according to its capacity. Tasks, on the other hand, are characterized by their CPU and memory requirements, which determine the computational resources needed and influence how tasks are assigned to servers.

3.2 System Model

The problem formulation revolves around finding an efficient task-scheduling strategy that balances load across VSs while considering dynamic cloudlet arrivals and VS capabilities. The proposed solution combines a heuristic approach with a PSO algorithm to optimize cloudlet-to-VS mapping for improved execution times within a cloud datacenter.

Our proposed architecture, as illustrated in Fig. 1, outlines a CHPSO strategy specifically designed for cloud data centers. This architecture efficiently allocates tasks among VS by integrating a heuristic workload balancing method with chi-squared-based modeling of cloudlet arrival times. At the core of the system is the Task Planner, which uses chi-squared modeling to adjust the arrival times, enabling the strategy to adapt seamlessly to workload variations. Complementing this is the CHPSO-Enhanced Task Scheduler, which employs a heuristic to adjust task distribution based on cumulative processing times and the computational capabilities of virtual servers, ensuring efficient load balancing. Additionally, VM Monitors continuously track resource utilization and provide real-time feedback to further refine task placement using PSO algorithm.

Task management and monitoring involve several key components that ensure efficient task processing and resource allocation within a cloud computing system. The **Task Queue** temporarily stores incoming tasks before processing, serving as a buffer to accommodate user requests. The **Task Planner** analyzes these tasks and makes strategic decisions regarding resource allocation and scheduling by considering factors such as task complexity, server capabilities, priorities, resource availability, and workload characteristics. Based on these decisions, the **Task Scheduler** retrieves tasks from the queue and assigns them to appropriate servers, ensuring that tasks are executed on the most suitable computational resources. Additionally, **VM Monitors** track the performance and status of virtual machines, providing critical data that helps the Task Planner and Scheduler make informed decisions for efficient task allocation and system optimization.

3.3 Scheduling Model

In our proposed technique, scheduling and resource provisioning are integrated and treated as an optimization issue. Although our algorithm primarily focuses on resource provisioning, it also significantly impacts resource management by optimizing resource usage within the cloud environment. To clarify, consider the following explanation:

3.3.1 Assumptions for Task Scheduling and Execution

Task scheduling and execution rely on several key factors. **Task Length** represents the computational resources required for each task, indicating its complexity and system demand. **MIPS** quantifies the processing power of each VM, determining how quickly tasks can be executed. **Task Assignment** ensures that each task is allocated to a VM that can execute it in

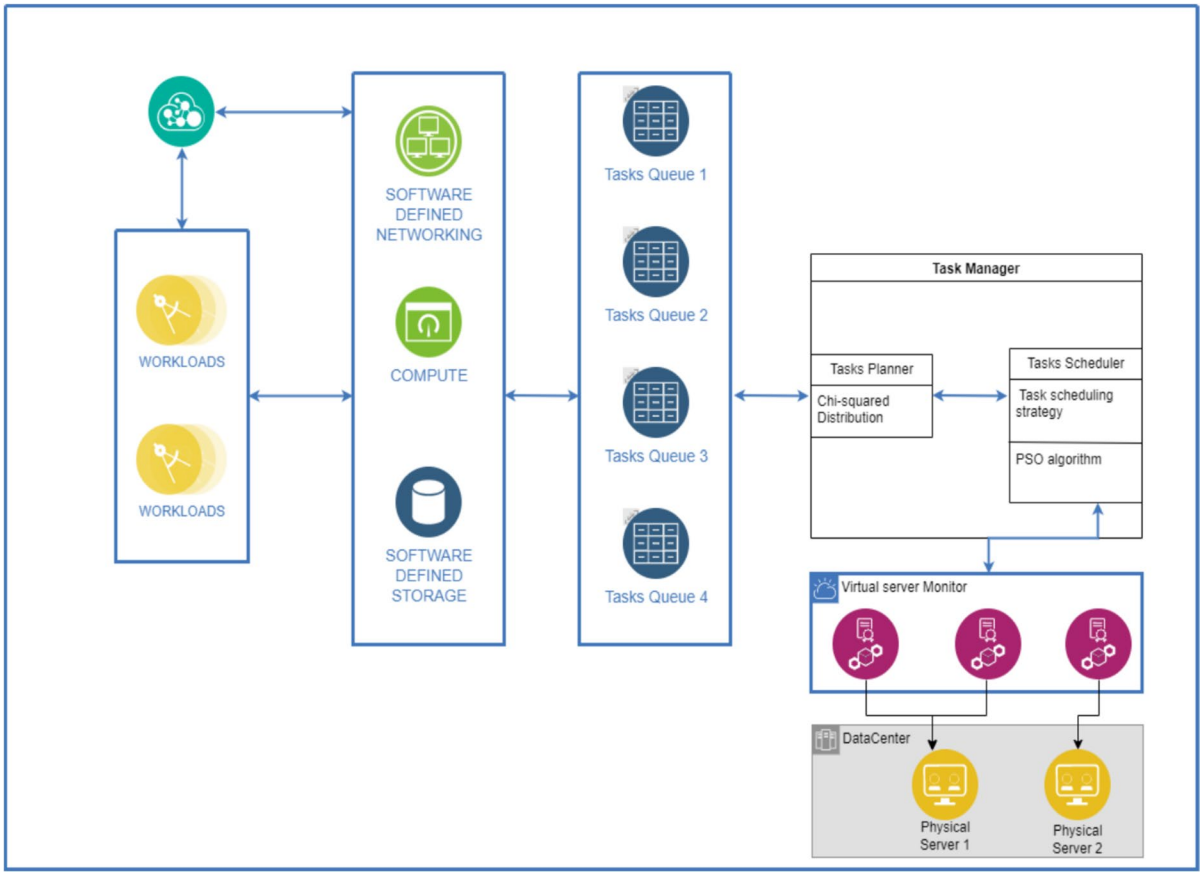


Fig. 1 A proposed system architecture designed specifically for CHPSO

the shortest possible time, considering the VM's current load and processing power relative to the task's demands. **Dynamic Task Allocation** enables real-time task distribution based on the observed load on each VM and the specific processing capabilities required, allowing for adaptive workload redistribution. Finally, **Non-Preemptive Execution** ensures that once a task starts running on a VM, it continues uninterrupted until completion, providing predictable execution times.

We suppose that the **task execution matrix** $\boldsymbol{TSE_{k\times m}}$ is created of $\boldsymbol{k\times mTE(vs_i, TS_j)}$ utilizing $\boldsymbol{k}$ of virtual servers ($\boldsymbol{vs}$) and $\boldsymbol{m}$ of tasks ($\boldsymbol{TS}$) using Eq. (1).

$$TSE_{k\times m} = \begin{bmatrix} TE_{1,1}TE_{1,2}\ldots TE_{1,j}\ldots TE_{1,m} \\ \vdots\vdots\vdots\vdots \\ TE_{i,1}TE_{i,2}\ldots TE_{i,j}\ldots TE_{2,m} \\ \vdots\vdots\vdots\vdots \\ TE_{k\times 1}TE_{k\times 2}\ldots TE_{k\times j}\ldots TE_{k\times m} \end{bmatrix} \quad (1)$$

In the $TSE_{k\times m}$ matrix, each row indicates the execution time of a task on a specific virtual server, while each column reflects the execution time of a task across several virtual servers. Let $\boldsymbol{TE_{i,j}}$ be the **execution time for task** $\boldsymbol{TS_j}$, which corresponds to $\boldsymbol{vs_i}$. $TE_{i,j}$ is determined using Eq. (2).

$$TE_{ij} = \frac{M_j}{Mp_i \times Pe_i} \quad (2)$$

where M_j is the size of task TS_j. Mp_i represents the speed of vs_i. Pe_i represents the number of processing elements.

The makespan (MS) is the maximum time to finish each of the VSs in the schedule, as given in Eq. (3).

$$\text{MS} = \max_{1\le i\le k}\{CT_i\} \quad (3)$$

With CT_i is the completion time for tasks on the virtual server vs_i. It represents the total time required to complete all tasks assigned to the VS formulated by Eq. (4).

$$CT_i = \frac{\sum_{j=1}^{\text{tasksNbr}} \text{TS}_j.\,\text{length}}{\text{VS}_i \cdot\ \text{pesnumber} \times \text{VS}_i \cdot\ \text{mips}} \tag{4}$$

where tasksNbr represents the number of tasks that need to be completed on the virtual server i. TS_j. length represents the amount of computational work required for TS_j. VS_i.pesnumber represents the number of individual processors or cores available in the virtual server i. $\text{VS}_i \cdot$ mips represents the processing speed of the virtual server i.

Resource utilization (RU) measures how effectively the virtual server resources are being used relative to the total time and server count. High utilization rates can indicate efficient use of resources but might also point to potential bottlenecks if too high. It is calculated using Eq. (5).

$$RU = \frac{\sum_{i=1}^{VSNbr} CT_i}{\text{MS} \times VSNbr} \tag{5}$$

where CT_i is the completion time for tasks on the virtual server i. ***VSNbr*** is the available number of virtual servers. MS is the makespan.

The energy consumption of a physical server can be quantified and defined by the following Eq. (6),

$$EC = EC_{\text{idle}} + \left(EC_{max} - EC_{\text{idle}}\right) \times SU \tag{6}$$

where SU represents the server's utilization, EC_{idle} is the idle energy consumption, and EC_{max} is the maximum energy consumption.

The total energy consumption for all servers in the data center is then given by,

$$EC_{\text{total}} = \sum_{i=1}^{n} EC_i\left(SU_i\right) \tag{7}$$

where $EC_i\left(SU_i\right)$ denotes the energy consumption of an individual PS based on its utilization SU_i, and n is the total number of physical servers in the data center.

The abbreviations used in this research are listed in Table 2.

Table 2 Abbreviations used in this work

Abbreviations	Descriptions
CPU	Central Processing Unit
CT	Completion Time
EC	Energy Consumption
IT	Information Technology
MIPS	Millions of Instructions Per Second
MS	Makespan
PDF	Probability Density Function
Pe	Processing element
PS	Physical Server
PSO	Particle Swarm Optimization
RT	Response Time
RU	Resource Usage
SU	Server Utilization
TE	Execution Time
TS	Tasks
TSE	Tasks execution
VS	Virtual Server
$\boldsymbol{Mp_i}$	The speed of virtual server
maxEV	maximum number of Evaluation

3.3.2 *Optimization Objectives and Constraints*

In addressing the complex challenge of task scheduling in cloud computing environments, our algorithm is designed with the primary objective of minimizing the makespan (MS), which is critical for enhancing overall system efficiency and responsiveness.

The objective function $F(y)$ for our proposed scheduling algorithm is formulated to minimize the makespan while considering the operational constraints of the system. This is expressed mathematically as,

$$min(F) = min(\text{Makespan}) \tag{8}$$

subject to several key constraints that ensure feasible and efficient task allocation across VSs. The constraints are defined as follows: Each task TS_k must be assigned exactly once, ensuring complete task allocation without duplication or omission, represented by $\sum_{k=1}^{m} x_{ik} = 1$ for every task TS_k. Additionally, the allocation must not exceed the available resources of any VS, formulated as $\sum_{k=1}^{m} a_{ik} \times x_{ik} \leq b_i$ for each VS_i, where a_{ik} is the resource demand of the task TS_k on VS_i, and b_i is the total available resource on VS_i [32].

By focusing on minimizing makespan, our algorithm aims to reduce the total time required to complete all scheduled tasks, thereby reducing delays in processing. This approach not only improves the operational performance of cloud services but also enhances user satisfaction by delivering faster computational service.

3.4 CHPSO Theoretical Analysis

Our approach integrates three key components: (i) modeling cloudlet arrival times using the chi-squared distribution, (ii) balancing the workload across VSs via a heuristic based on cumulative processing times, and (iii) refining the cloudlet-to-VS mappings using PSO. Below, we detail and justify each element.

3.4.1 Chi-Squared Distribution for Modeling Cloudlet Arrivals

In many cases, user requests or task arrivals are modeled using a Poisson distribution because they represent count data over a fixed period [33]. However, the chi-squared distribution might not be the primary model for counting user requests, it plays a crucial role to quantify and test the variability in task processing or service times, which is critical when ensuring quality of service in cloud environments.

Particles in CHPSO algorithm are initialized with positions sampled from a chi-squared distribution. This ensures a wide and diverse coverage of the solution space (a broader Pareto front), thereby increasing the likelihood that at least some particles start near optimal regions.

The probability density function (PDF) of the non-central chi-squared distribution is given by Eq. (9).

$$f(x;k,\lambda) = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2}(\lambda/2)^i}{i!} \frac{x^{(k/2+i-1)}e^{-x/2}}{2^{k/2+i)}\Gamma(k/2+i)} \tag{9}$$

where x is the random variable,k is the degrees of freedom, $\lambda > 0$ is the non-centrality parameter, and Γ represents the gamma function. This formula allows us to incorporate a degree of randomness that reflects the variability and unpredictability of cloud computing workloads, thereby enhancing the realism of our simulations [34].

To illustrate the benefit of using the chi-squared distribution over uniform random assignment, consider the following. When $\boldsymbol{m}$ tasks ($\boldsymbol{TS}$) are distributed uniformly across $\boldsymbol{k}$ VSs, the load variance is approximately,

$$\sigma^2_{\text{uniform}} = \frac{m}{k} \tag{10}$$

This uniform approach does not account for the bursty and variable nature of task arrivals, often leading to clustering on certain servers and causing early imbalances. In contrast, our method uses chi-squared sampling, where the probability of assigning a task to a particular server is weighted by its expected workload. Specifically, the probability can be expressed as,

$$P\left(TS_i \rightarrow vs_j\right) \propto \frac{\lambda_j}{\sum_{j=1}^{k} \lambda_j} \tag{11}$$

with λ_j representing the expected load on server j. This weighted assignment leads to a reduced load variance, approximately given by

$$\sigma^2_{\text{chi-squared}} = \frac{m}{k}\left(1 - \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max}}\right) < \sigma^2_{\text{uniform}} \tag{12}$$

Where $\lambda_{max} and \lambda_{min}$ are the maximum and minimum expected loads, respectively. The term $\frac{\lambda_{max}-\lambda_{min}}{\lambda_{max}}$ quantifies the heterogeneity in VS capacities. Since $\lambda_{max} > \lambda_{min}$ [35], this formula **reduces initial imbalance** and accelerates convergence. By reducing initial load variance, CHPSO requires fewer iterations to reach a stable load distribution, improving convergence speed compared to standard PSO.

3.4.2 Heuristic Task Balancing Across VSs

Our heuristic workload balancing strategy further refines task assignments by iteratively allocating each task to the VS that minimizes the deviation from its estimated balanced execution time. Specifically, at each iteration, the heuristic evaluates the current cumulative processing time $\boldsymbol{C_j}$ for each server (using Eq. (2)) and then assigns the task—whose processing requirement is T_i—to the server for which the difference.

$$\Delta j = \left| C_j + T_i - TE_j \right|, \tag{13}$$

Where T_i is the processing requirement of the current task. TE_j represents the estimated balanced execution time for server j.

By starting with an initial distribution that already has a reduced variance $\sigma^2_{\text{chi-squared}}$, the heuristic is operating on a more balanced baseline. This results in smaller deviations Δj across servers, meaning that the heuristic can more effectively fine-tune the task assignments.

3.4.3 Integration with Particle Swarm Optimization

After generating an initial assignment using our heuristic, we further optimize the mapping of cloudlets to VSs by incorporating PSO. In our model, each potential solution is represented as a vector $\boldsymbol{x}$, where each element x_i indicates the virtual server assigned to cloudlet i.

Each particle in the swarm represents a candidate mapping, and its position is updated using the standard PSO equations

$$\begin{aligned} v_i(t+1) &= w \cdot v_i(t) + c_1 r_1 \left(p_i - x_i(t)\right) + c_2 r_2 \left(g - x_i(t)\right) \\ x_i(t+1) &= x_i(t) + v_i(t+1) \end{aligned} \tag{14}$$

where, $x_i(t)$ is the current position, $x_i(t+1)$ is the update position. $v_i(t)$ is the particle velocity, $v_i(t+1)$ is the particle updated velocity. ω is the inertia weight,$c_1 and c_2$ are acceleration coefficients, $r_1 and r_2$ are random coefficients, $pbest_i and$ gbest are the personal and global best positions respectively.

However, because our chi-squared initialization reduces load variance and our heuristic assigns tasks more evenly, the search space that PSO must explore is already much more favorable. This improved starting point enables PSO to converge more quickly towards high-quality solutions. Furthermore, we enhance the standard PSO update by incorporating a gradient-based heuristic term, which refines the search process even further, resulting in the modified update equation below,

$$\begin{aligned} v_i(t+1) =& w \cdot v_i(t) + c_1 r_1 \left(p_i - x_i(t)\right) + c_2 r_2 \left(g - x_i(t)\right) \\ &+ \alpha \cdot \nabla H(x_i(t)) \end{aligned} \tag{15}$$

where $\nabla H(xi(t))$ is the gradient of the heuristic function that quantifies the deviation in workload balance (i.e., $\Delta j = \left| C_j + T_i - TE_j \right|$), converges almost surely to a stable equilibrium where $\nabla H\left(x_i(t)\right) \to 0$. This term actively steers the particles toward solutions that minimize imbalances.

If standard uniform sampling yields a load variance of Eq. 10, our approach (CHPSO) achieves a lower variance (Eq. 12), effectively reducing the search space for PSO. As a result, fewer iterations are required to converge to an optimal solution.

3.4.4 CHPSO vs. PSO Convergence

In standard PSO, the number of iterations required for convergence is heavily influenced by the complexity of the search space, which, in turn, depends on the initial imbalance of task distribution. A more imbalanced initial state leads to a broader and less structured search space, requiring more iterations for PSO to find an optimal solution. Additionally, the inclusion of a heuristic gradient term helps ensure that the swarm converges to a stable equilibrium where the fitness function (Eq. 8) is minimized. In essence, the modified velocity update accelerates stabilization by steering particles toward the best-known solutions while also directly accounting for workload balance.

4 Methodology

This section introduces the novel CHPSO approach for cloud task scheduling, aiming to minimize the makespan and enhance resource efficiency in cloud computing environments. The CHPSO method involves several phases: initialization, optimization, and assignment. Initially, task arrival times are adjusted using a chi-squared distribution method to simulate realistic workload scenarios. During the optimization phase, a task scheduling algorithm dynamically adjusts to changing task demands and VS availability. In the assignment phase, tasks are allocated to VSs using the PSO algorithm such that overall system performance is optimized, ensuring efficient use of computational resources and reduced task completion times. This method is presented in Fig. 2.

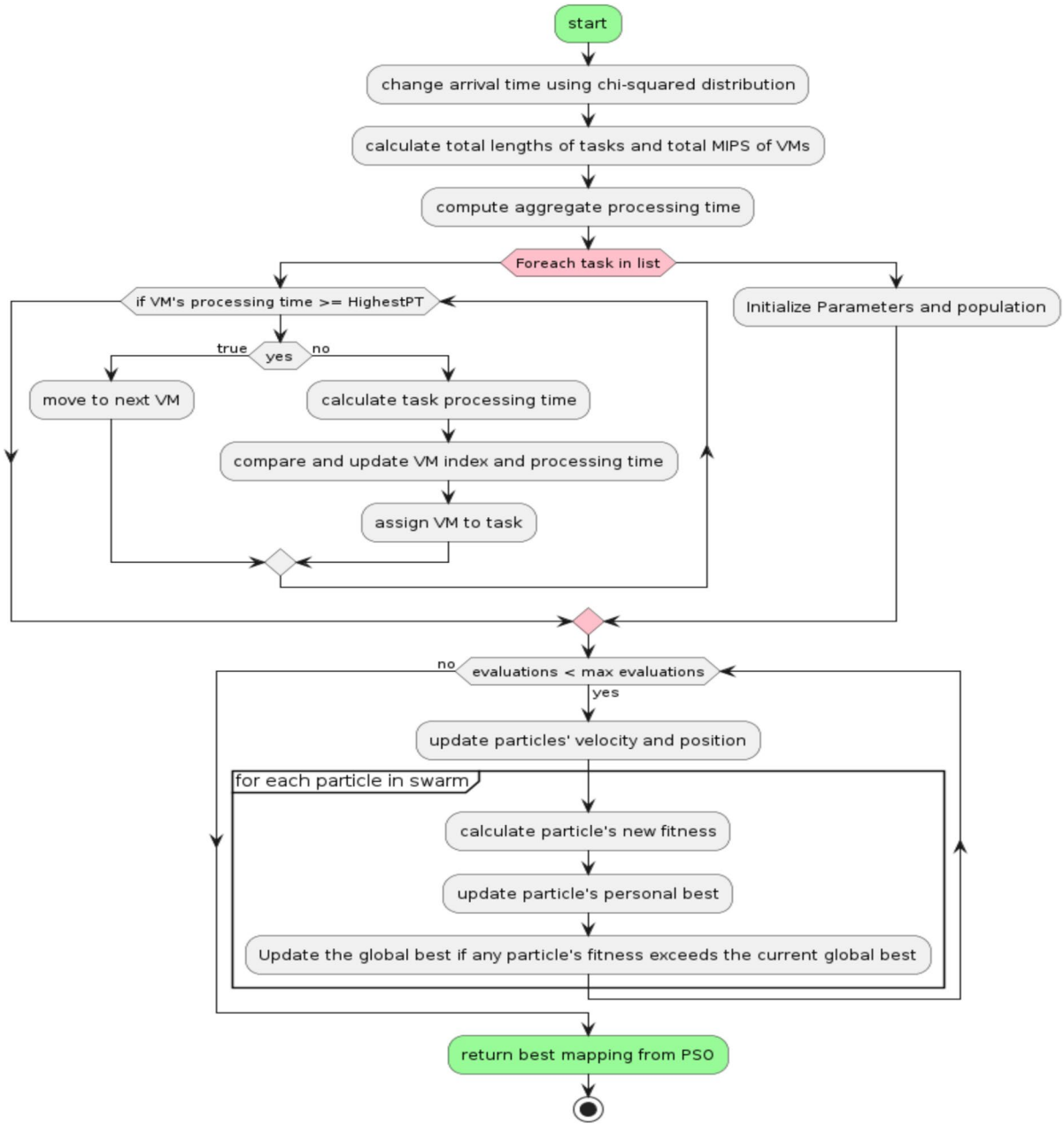


Fig. 2 Flowchart of CHPSO

4.1 Arrival Time Phase

In the arrival time phase of our algorithm, we model the task arrival times using a non-central chi-squared distribution, which is particularly suited for arrival time distribution [36]. This choice is inspired by the need to simulate realistic and varied computational loads dynamically. In cloud systems, user requests often follow a Poisson process, where arrivals are random but conform to an average rate (λ) over time. The Poisson distribution, which assumes a constant arrival rate (memoryless property) and struggles to model sporadic bursts of activity, making it less suitable for capturing sudden spikes in workload. The Gaussian distribution,

which symmetrically distributes values around a mean and even permits negative intervals, which is unrealistic for modeling arrival times in a real system [12].

In contrast, the chi-squared distribution is strictly positive and inherently right-skewed. This skewness mimics the bursty nature of cloud environments, where periods of low activity are occasionally interrupted by sudden surges in task submissions. By initializing particles with values proportional to the workload complexity using a chi-squared distribution, the algorithm adapts to the scale of the problem.

Algorithm 1: Arrival Times Scheduling

1: Input : CloudletList
2: Initialize distribution for arrival time (non-central chi-squared)
3: For each cloudlet in the CloudletList:
4: Store the sampled arrival time in history
5: Calculate the schedule time
6: If scheduleTime > 0:
7: Schedule the cloudlet to be sent to the assigned VS at scheduleTime
8: endFOR

Algorithm 1 provides a structured approach to dynamically scheduling cloudlets based on sampled arrival times from a non-central chi-squared distribution, which can help simulate more realistic scenarios in cloud computing environments by incorporating variability in task arrival times.

4.2 Task Scheduling Phase

In the initial scheduling in the proposed CHPSO, the Task scheduling strategy is used to compute the capabilities of both tasks and virtual servers. The pseudocode of Algorithm 2 provides a high-level representation of load balancing, capturing the key steps and logic involved in task scheduling among VSs.

The algorithm begins by computing the total required lengths of all tasks and the processing capabilities of the available VSs (line 3). It then establishes a high processing time benchmark to guide the task distribution process (line 4). The algorithm iteratively compares each task's processing time on both its assigned VS and potential alternative VSs (lines 6–11). If a more efficient VS is found (lines 8–9), the task is reassigned to optimize processing efficiency. The task is then officially assigned to the chosen VS (line 11), and the process continues until all tasks are allocated. This strategy ensures tasks are distributed in a manner that aims to prevent any single VS from becoming a bottleneck, thereby enhancing overall system performance and reducing the total execution time.

Algorithm 2: Tasks scheduling strategy

1: Input: CloudletList, VSList
2: Initialize processing times and capacities
3: Calculate total cloudlet length and total MIPS capacity
4: HighestProcessingTime= cloudletTotalLength/ VSTotalMips
5: Iterate over each cloudlet and VS to assign cloudlets based on processing time efficiency:
6: **For each cloudlet:**
7: **If VS's current load is above the highest processing time, move to next VS**
8: Calculate cloudlet processing time on current and potential VS
9: Assign cloudlet to VS where it minimizes the increase in processing time
10: Update VS's total processing time with the newly assigned cloudlet's time
11: Set cloudlet's VS ID to the selected VS
12: End iteration

4.3 PSO Assignment Phase

In this phase, the PSO method aimed at enhancing the preliminary task allocations in the previous subsection. Utilizing PSO, the assignment of tasks to virtual servers is optimized continuously through an iterative process. This is driven by an objective function, formally defined in Eq. (8), which is designed to minimize the makespan. By adjusting task allocations based on both local and global best solutions discovered during the optimization process, this phase effectively responds to evolving workload demands and varying capacities of VSs, ensuring that tasks are executed efficiently and promptly.

Algorithm 3: PSO Assignement

```
1: Input: MaxIteration
2: Output: Mapping
3:  Initialize Parameters and population
4:  PSO process (Algorithm 4)
5:  Determine Best Solution.
6:  return Mapping
```

The initialization of the PSO Assignment algorithm is responsible for creating particles in the swarm for each iteration. Particle initialization involves configuring the initial position and velocity. Then, the algorithm initializes the number of cloudlets to determine the problem's dimensionality and establishes the particle swarm's size. This setup is crucial for starting the PSO's iterative optimization process aimed at refining task assignments. The goal is to discover the optimal task-to-VS mapping that minimizes the makespan. Followed by the execution of the optimization process using Algorithm 4, which represents the main logic of the PSO.

Algorithm 4: PSO process

```
1: Determine the initial best particle based on the fitness of the initial population.
2: for(i=0;i<max_evaluations ;i++)
3:      Update Velocities.
4:      Update Positions.
5:      Calculate the fitness for each particle's new position.
6:      Update each particle's personal best if the current fitness is better.
7:      Update the global best if any particle's fitness exceeds the current global best.
8: endFor
```

The optimization process begins by identifying the best particle from the initial population based on the fitness value, which is measured by the makespan function, representing the total time required to complete all tasks (Line 1). The PSO then iteratively refines the solutions over a set number of evaluations (Line 2). Each iteration involves adjusting the velocities of particles (Line 3) and updating their positions (Line 4), influenced by both personal and global best positions found so far. After the position updates, the algorithm recalculates the fitness for each particle to evaluate the new potential solutions (Line 5). If a particle's new position yields a better fitness score than its previous best, the personal best is updated accordingly (Line 6). Moreover, if this new fitness score surpasses the current global best, the global best is also updated, reflecting the new optimal solution (Line 7). This loop (Line 8) ensures continuous improvement towards an optimal solution, striving to minimize the makespan and thereby enhance scheduling efficiency.

Algorithm 5: CHPSO Algorithm

```
1: Input: MaxIteration, CloudletListSize, VSListSize
2: Output: Mapping
3: Initialize the distribution of cloudlets using the Arrival Times Scheduling algorithm (Algorithm 1)
4: Assign the CloudletList to the broker using the Tasks Scheduling Strategy algorithm (Algorithm 2)
5: for(i=0;i< MaxIteration;i++)
6:       PSO Assignement algorithm (Algorithm 3)
7: endFor
8: return Mapping
```

The pseudocode for the proposed CHPSO approach is presented in Algorithm 5, highlighting its multi stage process that combines the efficiency of heuristic-based decision-making with the global search capabilities of PSO. By leveraging the statistical properties of the chi-squared distribution, CHPSO better captures real-world task arrival patterns, enhancing its effectiveness in dynamic environments.

4.4 Complexity Analysis

The complexity of the proposed algorithm is derived by analyzing its constituent operations. Let $\boldsymbol{m}$ denote the number of tasks, $\boldsymbol{k}$ the number of virtual servers (VSs), $\boldsymbol{p}$ the number of particles in the swarm, and $\boldsymbol{maxEV}$ the maximum number of evaluations in the optimization process. The complexity of initializing task arrival times using a chi-squared distribution is $\boldsymbol{O(n)}$, assuming this operation is performed individually for each task. Calculations of the total lengths of tasks and total MIPS of VSs are performed in $\boldsymbol{O(m)}$ and $\boldsymbol{O(k)}$, respectively.

The core of the algorithm involves a nested loop where each task is evaluated against available VSs to determine the optimal assignment, leading to a complexity of $\boldsymbol{O(m \times k)}$. Additionally, the PSO component, which iteratively updates particles' velocities and positions, has a complexity of $\boldsymbol{O(maxEV \times p)}$. Within each iteration, the fitness evaluation—dependent on both the number of tasks and VSs—further compounds this complexity, resulting in a dominant term of $\boldsymbol{O(maxEV \times p \times max(m,k))}$.

5 Simulation Experiments

This section outlines the experimental setup and evaluation parameters used to assess the effectiveness of the proposed hybrid technique (CHPSO). We will compare the CHPSO technique against several established algorithms including HSGA [37], which uses a hybrid genetic algorithm approach to efficiently schedule workflow graphs by prioritizing tasks based on their topological impact and generating an optimal initial population through a combination of Best-Fit and Round Robin methods; Greedy PSO (G-PSO) [38], which accelerates convergence by quickly determining initial particle values with a greedy algorithm; standard PSO, a widely known metaheuristic that, despite its simplicity and fast convergence, is prone to premature convergence and limited search space exploration; and FCFS [39], a basic scheduling strategy that assigns tasks in order of arrival as a straightforward benchmark. All simulation experiments are implemented in Java using a CloudSim framework [40]. These experiments are conducted on a PC equipped with a 2.40 GHz processor and 20 GB of RAM.

5.1 Performance Metrics

The evaluation of scheduling and optimization algorithms requires understanding how key performance metrics influence each other. **Makespan (MS)**, which represents the longest time required to complete all tasks, is often used to measure efficiency. Reducing makespan generally improves **resource usage (RU)**, as tasks are distributed more effectively across VSs.

However, pushing for a lower makespan can sometimes lead to **higher energy consumption (EC)** since more computational resources may be activated to speed up execution. This trade-off is crucial in balancing efficiency and sustainability.

Similarly, **RU** plays a key role in system performance. Higher RU typically means that available resources are being well utilized, which can help lower **execution time (TE)** and improve **response time (RT)** by processing tasks more efficiently. However, if RU is too high, it may cause server congestion, increasing response time due to system overload. On the other hand, underutilized resources lead to inefficiencies, potentially increasing both **MS** and **TE**.

EC is another critical factor, particularly in large-scale cloud environments. Strategies that minimize EC, such as dynamic voltage scaling or workload consolidation, often come at the cost of increased **RT** and **MS** since tasks may be executed on fewer or less powerful servers. Conversely, prioritizing faster execution by keeping more resources active reduces RT and MS but increases EC.

RT, which measures how quickly a system reacts to incoming tasks, is particularly important for real-time applications. Lower RT is generally desirable, but achieving it often means running more resources at higher power, increasing **EC**. In contrast, energy-saving strategies, such as putting idle servers to sleep, can introduce delays, leading to a higher RT.

Finally, **TE**, which accounts for the total processing time of all tasks, is closely linked to **MS** but differs in that it considers the overall workload rather than just the longest execution path. A well-balanced resource allocation strategy can reduce TE while maintaining an optimal **MS**. However, inefficient scheduling can lead to bottlenecks, increasing both TE and RT.

In summary, optimizing these performance metrics requires balancing efficiency, energy consumption, and responsiveness. Reducing makespan and response time improves speed but often increases energy consumption. Higher resource usage improves efficiency but may lead to congestion if over-utilized. Energy-efficient scheduling reduces costs but can slow down task execution. Thus, finding the right trade-off between these factors is essential for designing effective and sustainable scheduling algorithms.

5.2 PSO Algorithm Parameters

To illustrate the setup for the PSO phase in our methodology, we configured the algorithm with the parameters taken from [41], as detailed in Table 3. We conducted 10 experiments, each comprising 100 iterations. The swarm size was set to 100 particles, ensuring a diverse population for efficient exploration. To enhance the algorithm's adaptability during optimization, we employed an inertia weight that dynamically adjusted between 0.9 and 0.5. Additionally, we incorporated cognitive and social knowledge factors. The cognitive factor was set to 2, allowing each particle to consider its own historical best position. Meanwhile, the social factor was also set to 2, enabling particles to learn from the best position achieved by the entire swarm. These factors play a crucial role in balancing exploration and exploitation, leading to effective task-to-VS mappings.

5.3 System Configuration Environment

To ensure a comprehensive evaluation, our simulation setup includes a detailed configuration of VSs, cloudlets, and datacenters. The specific configurations used in our experiments are organized into Table 4 to provide a structured overview of the system's environment, facilitating a clearer understanding of the experimental context and the scalability of CHPSO. This configuration includes details on the number and type of VSs, the properties of cloudlets, and the characteristics of the data centers involved.

5.4 Results with Different Metrics

In our experimentation, we compare the performance of our proposed CHPSO algorithm against other algorithms across various scenarios tailored to assess their effectiveness in cloud computing environments. These scenarios differ in terms of the number of physical servers (PS), VS, and tasks, as outlined in Table 5. This comprehensive assessment covers key performance metrics such as makespan, energy consumption, resource utilization, response time, and execution time. The aim is to highlight the strengths and potential limitations of CHPSO in contrast to existing strategies under different operational conditions.

5.4.1 Makespan

In this subsection, we analyze the time taken by each algorithm to complete all assigned tasks across various scenarios.

Figure 3 demonstrates that CHPSO consistently achieves the lowest makespans across diverse workload scenarios. This superior performance stems from two key aspects of our strategy. First, the chi-squared-based initialization effectively reduces the load variance from the outset, ensuring a more balanced distribution of tasks among virtual servers. This leads to more efficient resource utilization and, consequently, a lower makespan. Second, the integration of a heuristic workload balancing strategy within the PSO framework enables dynamic fine-tuning of task assignments. This combined approach allows CHPSO to adapt to changing conditions and avoid the pitfalls of premature convergence that affect standard PSO and other methods like G-PSO, which shows scalability issues in task-intensive scenarios. For example, in the scenario with 20 physical servers, 80 virtual servers, and 1000 tasks, CHPSO achieves a makespan of only 315.16 ms compared to FCFS's 1200 ms. These results highlight that the enhanced initialization and iterative balancing not only reduce the number of iterations required for convergence but also improve overall efficiency.

5.4.2 Energy Consumption

Energy consumption signifies the amount of energy each algorithm uses during task execution. Energy efficiency, in this context, translates to cost-effectiveness.

Figure 4 reveals that G-PSO consistently exhibits the highest energy usage, potentially reflecting its intensive computational demands or inefficiencies in managing larger workloads. Both HSGA and PSO maintain moderate energy use; however, PSO shows the highest energy consumption in scenarios with extensive tasks, suggesting potential inefficiencies in resource allocation or increased computational overhead. In contrast, FCFS employs a simpler scheduling strategy and exhibits a relatively consistent pattern of energy use, indicating less optimized energy management compared to the adaptive CHPSO strategy. This analysis underscores CHPSO's capacity for reducing operational costs and enhancing sustainability in cloud computing environments, making it a favorable option for energy-conscious deployments, especially in demanding scenarios. For instance, in the scenario PM = 20/VM = 80/Task = 1000, CHPSO achieved an energy consumption of 1,236,148.606 w/s, and in PM = 3/VM = 30/Task = 1000, it was 1,116,374.23 w/s.

5.4.3 Resource Utilization

Resource utilization measures how effectively each algorithm uses computing resources during task execution. An algorithm that can achieve high resource utilization is desirable as it means that the computing resources are being used to their full potential, leading to improved efficiency and potentially lower costs.

Figure 5 illustrates a comparative analysis of resource utilization across different workload scenarios. CHPSO consistently shows high resource utilization, excelling particularly in complex scenarios with many VSs and tasks—such as the one with 20 PSs and 80 VSs handling 1000 tasks, achieving near maximum utilization at 98.66%. This indicates CHPSO's effective resource management and task distribution in dense environments. HSGA also performs robustly in similar environments, maintaining high utilization without sacrificing efficiency even as task complexity increases. In contrast, PSO demonstrates lower utilization rates in several scenarios, highlighting potential inefficiencies in matching tasks to VS capabilities, particularly under heavy loads. FCFS, while less sophisticated, still maintains moderate to high utilization, reflecting its straightforward approach that generally results in decent resource usage.

5.4.4 Response Time

The response time examines the latency from task arrival to commencement of processing by each algorithm. This metric reflects the algorithms' responsiveness in dynamic environments.

The response time analysis, depicted in Figure 6, highlights CHPSO's consistent delivery of low response times across all scenarios, affirming its rapid task-handling capability. This is markedly noticeable in scenarios with high task density, such as PS = 20/VS = 80/Task = 5000, where CHPSO significantly outperforms G-PSO, the latter showing the highest response times

Table 3 PSO configuration

Experiment	Iterations	Swarm Size	Inertia Weight	Cognitive Factor	Social Factor
10	100	100	0.9—0.5	2	2

under comparable conditions. Notably, in the PM = 20/VM = 80/Task = 1000 scenario, CHPSO excels with a response time of approximately 64 ms, showcasing its superior efficiency. This is due to its ability to dynamically adapt to real-time task arrival variations and enhance diversity in particle positions, leading to more optimized scheduling decisions. In contrast, G-PSO and PSO register much slower response times at about 4535 ms and 2202 ms respectively. Even the simpler FCFS method achieves a reasonable response time of 624 ms, demonstrating decent efficiency. Such performance underlines CHPSO's enhanced optimization capabilities, particularly suitable for environments requiring swift task processing.

5.4.5 Execution Time

Finally, the Execution Time subsection discusses the total time each algorithm takes to execute all tasks. We compare the thoroughness of each algorithm in handling extensive workloads. Shorter execution times are indicative of algorithms that can effectively manage and expedite workload processing.

HSGA and FCFS show consistent execution times across varying scenarios in Fig. 7, reflecting their straightforward, but less adaptable, task handling strategies. In contrast, G-PSO's execution times are highly variable, with notably long durations in complex scenarios like PS = 20/VS = 80/Task = 5000, suggesting inefficiencies under heavy loads. PSO, while variable, maintains moderate execution times, indicating a balanced approach to task distribution and processing speed. Particularly, CHPSO excels in dense environments, dramatically lowering execution times in scenarios with 1000 and 3000 tasks to 4,078.33 ms and 1,951.62 ms, respectively, far surpassing G-PSO and PSO. This underscores CHPSO's capability for dynamic task allocation and computational efficiency, highlighting its potential in challenging cloud computing environments.

To summarize, CHPSO outperforms HSGA, G-PSO, PSO, and FCFS in scenarios with high-density tasks, particularly excelling in the PS = 20/VS = 80/Task = 5000 scenario where it demonstrates substantially lower response times and energy consumption. This indicates CHPSO's strong capability in managing substantial computational demands and optimizing task distribution to enhance resource utilization and system efficiency. CHPSO's performance underscores its potential to significantly improve operational responsiveness and sustainability in complex cloud computing environments.

5.5 Discussion

Our experiments have shown that CHPSO consistently delivers low makespan and energy consumption across all tested scenarios. This suggests that the algorithm efficiently utilizes available resources, regardless of the number of tasks or servers. The heuristic strategy integrated with PSO ensures balanced workload distribution across virtual servers, dynamically adjusting to minimize idle times and maximize resource utilization. This dynamic adjustment is crucial for handling large-scale operations and maintaining system efficiency.

Moreover, CHPSO's ability to adapt to varying numbers of tasks and servers demonstrates its robustness in managing dynamic cloud environments. This adaptability is vital for real-world applications where cloud infrastructures frequently scale up or down based on demand. During our scalability assessment, we identified increased communication overhead as the number of servers and tasks grew. However, CHPSO's design effectively mitigates this through efficient task scheduling and resource allocation strategies, ensuring that communication overhead does not significantly impact performance.

The integration of PSO in CHPSO enhances its load-balancing capabilities, ensuring that tasks are optimally distributed across the cloud infrastructure. This is particularly important in scenarios with a high number of tasks, where efficient load balancing can significantly impact overall system performance. Optimized resource utilization ensures that no single server is overutilized while others remain underutilized, preventing bottlenecks and reducing idle times. Additionally, effective load balancing minimizes the

Table 4 Detailed System Configuration for Simulation Experiments

Component	Attribute	Description
General	Architecture	×86
	Operating System	Linux
	VSM	Xen
Datacenter	Time Zone	10.0
	Cost (Processing)	3.0
	Cost (Memory)	0.05
	Cost (Storage)	0.001
	Cost (Bandwidth)	0.0
VS	Image Size	1024 MB (1 GB)
	Memory	1024 MB (1 GB)
	Bandwidth	1024 MB
	CPUs	1
	MIPS	range [100, 1000]
	Cloudlet Scheduler	SpaceShared
Cloudlet	Length	Randomly set between 1000 to 2000
	File Size	300 KB
	Output Size	300 KB
	Utilization Model	Full utilization for CPU, memory, and bandwidth

Table 5 Scenarios tested

Scenarios	PS	VS	Tasks
1	3	30	1000
2	2	6	1000
3	20	80	1000
4	20	80	3000
5	20	80	5000

total execution time (makespan) by allowing tasks to be processed concurrently and efficiently, leading to faster response times and improved overall performance. Energy efficiency is another significant benefit, as balanced load distribution prevents server overloads, reducing energy consumption and operational costs. These improvements contribute to a more robust and efficient cloud computing environment, ultimately enhancing the quality of service provided to end-users.

5.6 Sensitivity Analysis for Parameters

We evaluated the impact of parameter variations on CHPSO's performance using two scenarios; **Small-scale** (PM3/VM30/Task1000) and **Large-scale** (PM20/VM80/Task5000). The results are summarized in Table 6, showing variations in makespan, resource utilization, and energy consumption across different parameter settings.

To evaluate inertia weight, we tested both small-scale (1,000 tasks) and large-scale (5,000 tasks) scenarios, each with a focus on different metrics. For smaller workloads, we prioritized makespan, reflecting the needs of latency-sensitive applications like real-time analytics. In contrast, for larger workloads, we examined resource utilization and energy consumption, which are critical for optimizing sustainability in cloud data centers.

As shown in Table 6, adjusting the inertia weight significantly impacts CHPSO's performance. For small-scale workloads, a dynamic inertia weight [0.9,0.5] reduces the makespan to 3,033 ms, outperforming static inertia ($w = 0.9$) and ($w = 0.5$), which yield makespans of 3,890 ms and 3,450 ms, respectively. For large-scale workloads, dynamic inertia improves resource utilization (82.1%) and energy consumption (1.11 M) compared to static settings ($w = 0.9$), which result in lower utilization (75.3%) and higher energy consumption (1.33 M). These findings demonstrate the advantage of adaptive inertia weight in handling diverse workloads efficiently.

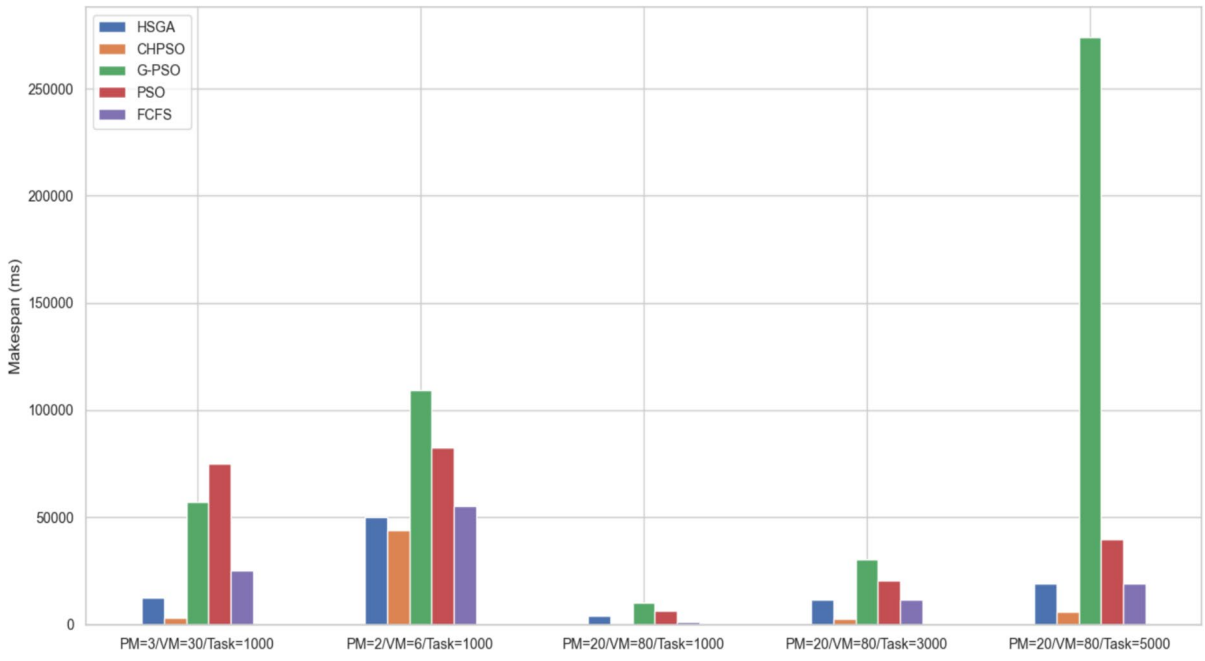


Fig. 3 Makespan Comparison across different scenarios

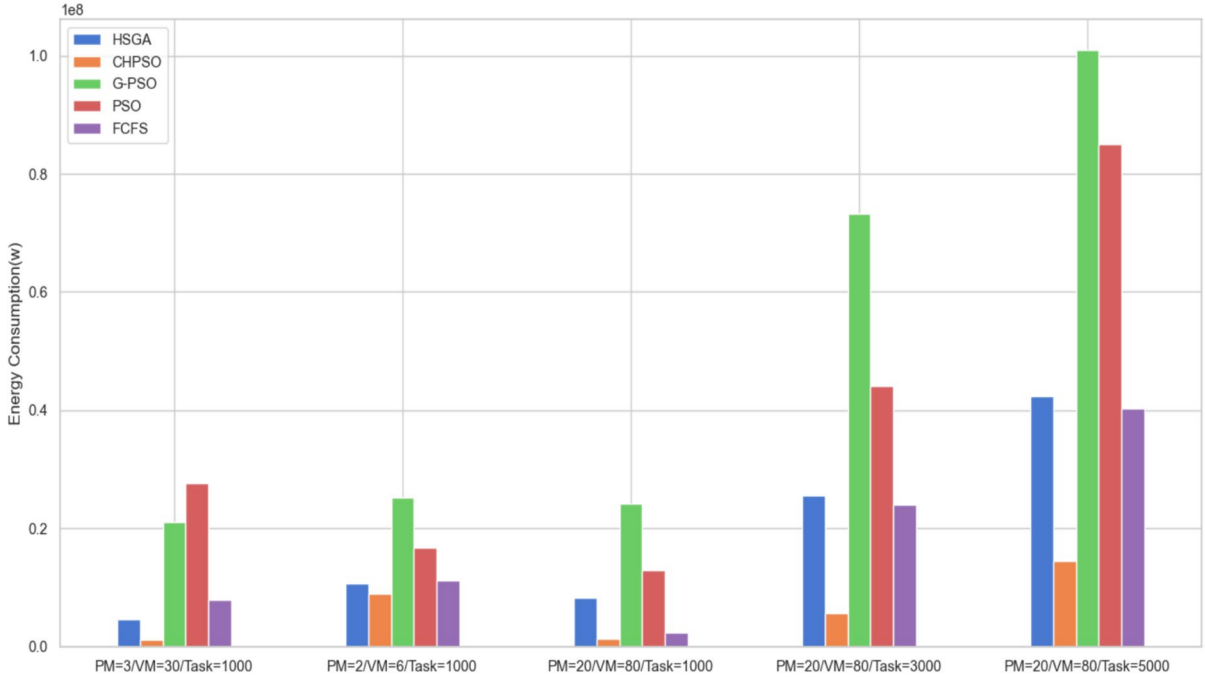


Fig. 4 Energy consumption comparison across different scenarios

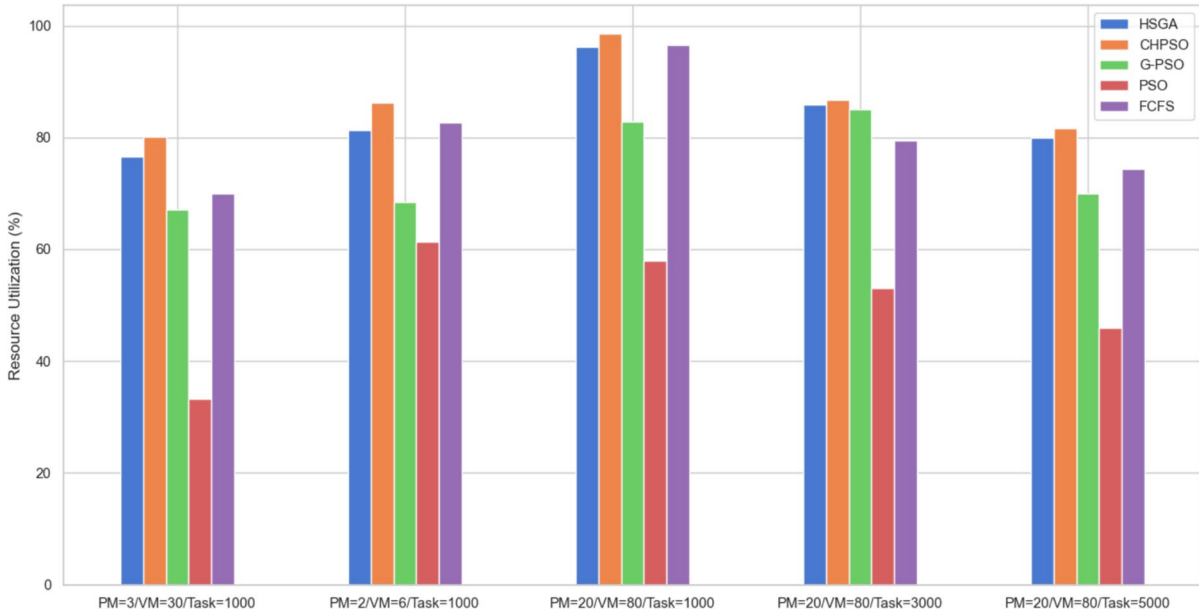


Fig. 5 Resource utilization comparison across different scenarios

Similarly, cognitive and social factor variations affect CHPSO's performance, particularly in large-scale workloads. When c1 = 2.5, c2 = 1.5, CHPSO experiences a high makespan (6,550 ms), lower resource utilization (74.8%), and increased energy consumption (1.35 M). In contrast, the baseline configuration (c1 = 2, c2 = 2) achieves better resource utilization (83.7%) and lower energy consumption (1.14 M). These results highlight the need for balanced parameter tuning to optimize task scheduling performance.

Cognitive and social factors were tested only in large-scale scenarios, as smaller workloads showed little sensitivity to these parameters. Managing 1,000 tasks across 30 servers does not require the same level of coordination as handling 5,000 tasks across 80 servers. By targeting parameter variations where they matter most, we avoid redundant testing while ensuring that our metrics align with real-world priorities—speed for smaller systems and efficiency for large-scale deployments. This streamlined approach enhances analysis without compromising valuable insights.

5.7 CHPSO Convergence

In this section, we evaluate CHPSO and examine its convergence behavior toward the optimal solution. Figure 8 illustrates the convergence pattern of CHPSO over 100 iterations across five different scenarios. As shown, CHPSO begins to stabilize around the 40th iteration, regardless of workload characteristics, indicating that beyond this point, additional iterations do not result in a significant reduction in makespan.

Figure 8 compares the convergence behavior of CHPSO and PSO across different scenarios. CHPSO consistently achieves a significantly better makespan than PSO from the very beginning, indicating that our initialization strategy and chi-squared modifications effectively guide the algorithm toward efficient solutions right from the beginning. While standard PSO gradually improves over time, it never catches up to CHPSO, reinforcing the advantage of CHPSO's balance between exploration and exploitation. In most scenarios, CHPSO converges quickly and remains stable, showing that it effectively maintains a strong solution. However, in Scenario 2, its convergence takes slightly longer to stabilize compared to the other scenarios. This suggests that the workload characteristics in this scenario introduce additional complexity, making it harder for CHPSO to quickly reach an optimal or near-optimal solution. Despite this delay, CHPSO still outperforms PSO, demonstrating its ability to adapt and eventually reach a stable makespan, albeit at a slightly slower rate.

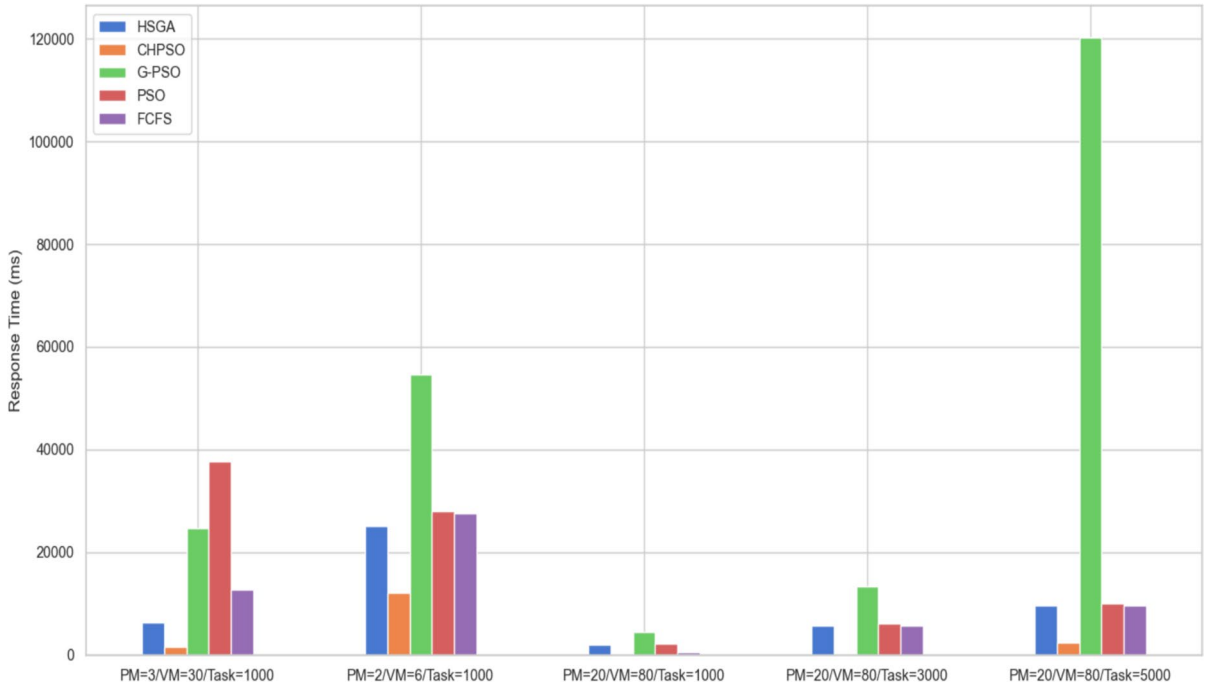


Fig. 6 Response time comparison across different scenarios

To quantify the performance gains of CHPSO over the baseline PSO, we calculate the relative improvement in makespan using the formula (16),

$$\text{Improvement } (\%) = \left(\frac{\text{PSO Makespan} - \text{CHPSO Makespan}}{\text{PSO Makespan}} \right) \times 100\% \tag{16}$$

The Table 7 shows the results for each scenario.

These results demonstrate that CHPSO significantly outperforms PSO across all scenarios, with improvements ranging from 46.73% to 95.96%. Notably, the most substantial improvement is observed in the PM3/VM30/Task1000 scenario, where CHPSO reduces the makespan by 95.96%, which means it is proven ability to explore complex, multi-dimensional search spaces and converge to high-quality solutions. Our algorithm provides a strong initial guess, and the stochastic nature of PSO refines this mapping further. Analyzing the expected decrease in the objective function across iterations shows that this hybrid approach systematically reduces the discrepancy between the actual load distribution and the ideal balanced state.

Furthermore, we compared execution times per iteration across different scenarios as illustrate in the Table 8.

CHPSO consistently outperforms standard PSO by requiring 40–60% fewer iterations to converge. For example, in the PM20/VM80/Task3000 scenario, CHPSO achieves optimal scheduling in just 5 iterations compared to PSO's 70 iterations. Although CHPSO's per-iteration runtime scales predictably with system size—about 490 ms per iteration for smaller setups like PM3/VM30 and roughly 2,094 ms for larger configurations like PM20/VM80/Task5000—PSO suffers from inefficiencies that lead to erratic scaling, such as taking 1,254 ms per iteration in the PM20/VM80/Task3000 case. Importantly, CHPSO's substantial reduction in the number of iterations more than compensates for its moderate per-step overhead, resulting in dramatic overall runtime improvements. For instance, in the PM20/VM80/

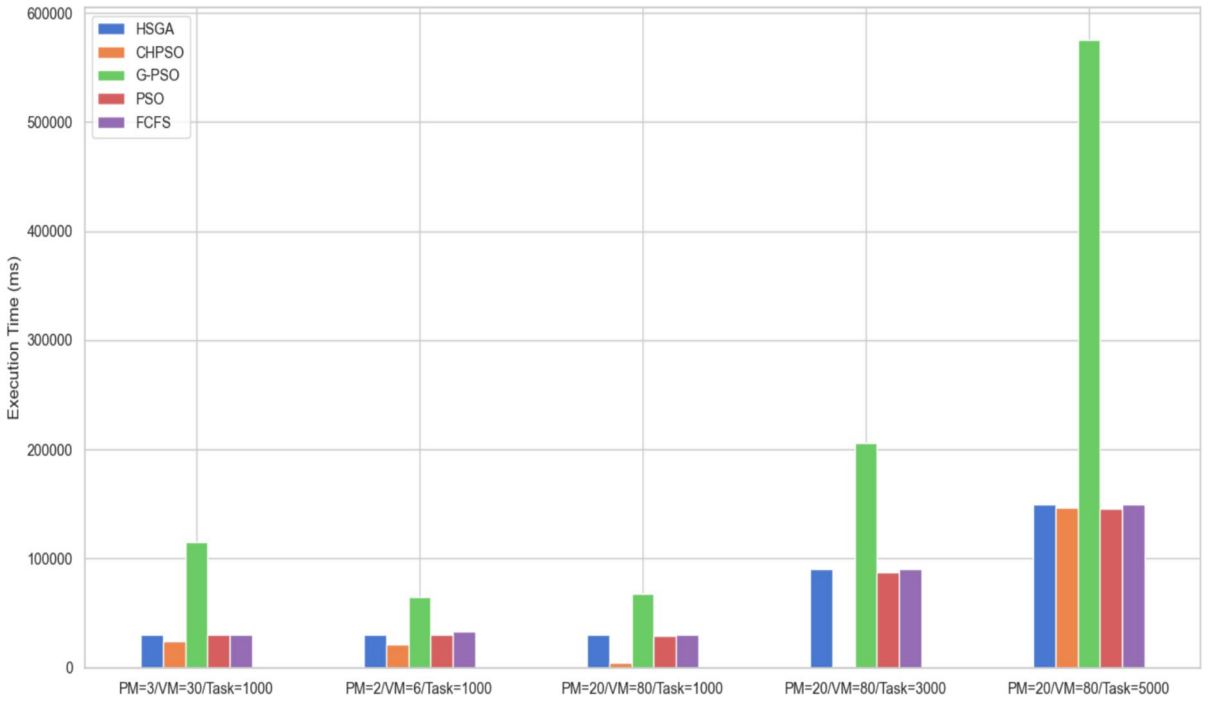


Fig. 7 Execution time comparison across different scenarios

Task1000 scenario, CHPSO completes scheduling in 4,078 ms, which is approximately 7 times faster than PSO's 29,220 ms.

5.8 Statistical Analysis

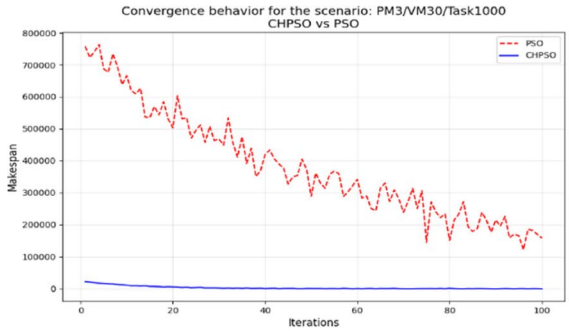
This section quantitatively assesses the performance of the CHPSO algorithm against benchmarks set by established algorithms such as HSGA, G-PSO, PSO, and FCFS. We conducted a series of statistical tests across various metrics including makespan, energy consumption, resource utilization, response time, and execution time.

5.8.1 Analysis of Variance (ANOVA)

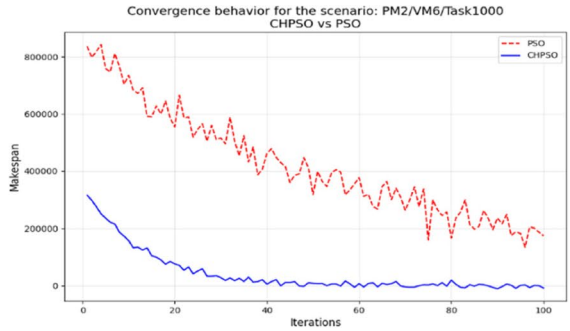
An ANOVA approach was utilized to identify significant performance differences among the algorithms. This method is particularly suited for experiments where multiple algorithms are evaluated across

Table 6 Parameters Sensitivity Analysis

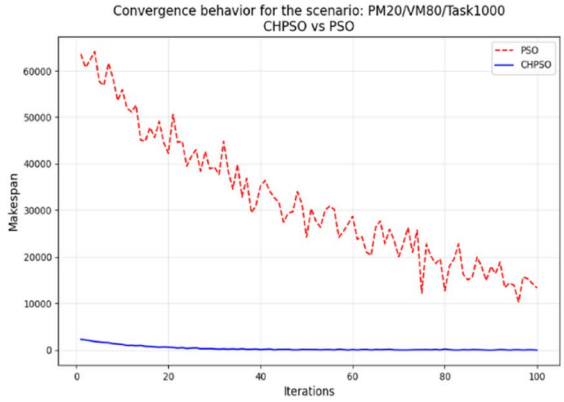
Parameter Variation	Scenario	Makespan	Resource Utilization	Energy Consumption
Dynamic Inertia [0.9,0.5]	PM3/VM30/Task1000	3,033 ± 95	-	-
	PM20/VM80/Task5000	-	82.1% ± 1.8	1.11 M ± 0.03
Static Inertia (w = 0.9)	PM3/VM30/Task1000	3,890 ± 120	-	-
	PM20/VM80/Task5000	-	75.3% ± 2.1	1.33 M ± 0.06
Static Inertia (w = 0.5)	PM3/VM30/Task1000	3,450 ± 110	-	-
	PM20/VM80/Task5000	-	79.6% ± 1.9	1.25 M ± 0.05
Baseline (c1 = 2,c2 = 2)	PM20/VM80/Task5000	5,736 ± 180	83.7% ± 1.5	1.14 M ± 0.03
c1 = 1.5,c2 = 2.5	PM20/VM80/Task5000	6,210 ± 210	76.4% ± 2.2	1.29 M ± 0.05
c1 = 2.5,c2 = 1.5	PM20/VM80/Task5000	6,550 ± 220	74.8% ± 2.4	1.35 M ± 0.06



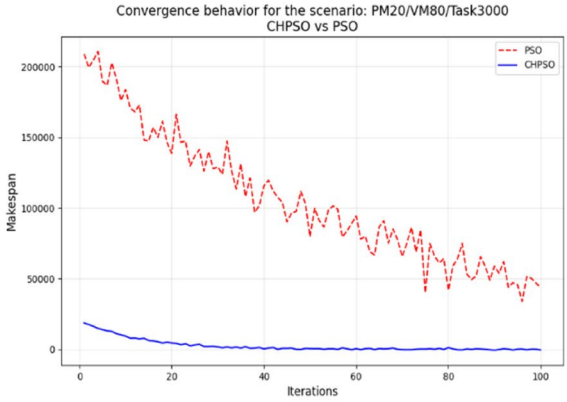
(a) Scenario 1.



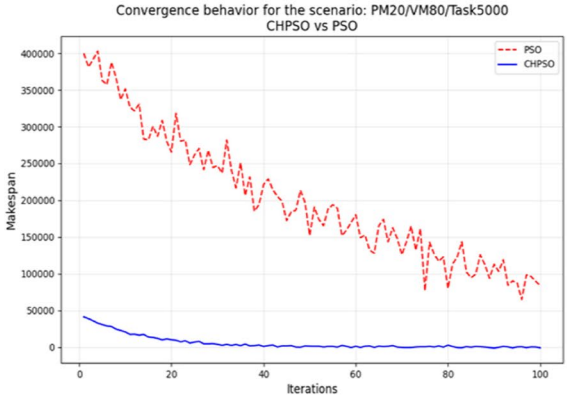
(b) Scenario 2.



(c) Scenario 3.



(d) Scenario 4.



(e) Scenario 5.

Fig. 8 Comparison of CHPSO and PSO convergence across different scenarios. (**a**) Scenario 1. (**b**) Scenario 2. (**c**) Scenario 3. (**d**) Scenario 4. (**e**) Scenario 5

various performance metrics, ensuring that any observed differences are statistically robust rather than due to random variation [24, 42]. Below are the ANOVA results for each metric:

Although our fitness function is designed to minimize makespan, our experimental results (Table 9) did not show statistically significant differences in makespan, response time, or execution time when

Table 7 Makespan Improvement

Scenario	PSO Makespan	CHPSO Makespan	Improvement (%)
PM3/VM30/Task1000	75,029.39	3,033.93	95.96%
PM2/VM6/Task1000	82,800.00	44,106.03	46.73%
PM20/VM80/Task1000	6,300.00	315.16	95.00%
PM20/VM80/Task3000	20,700.00	2,605.34	87.41%
PM20/VM80/Task5000	39,600.00	5,736.52	85.51%

Table 8 Execution times per iteration across different scenarios

Scenario	Algorithm	Execution Time (ms)	Iterations (maxEV)	Time/Iteration (ms)
PM3/VM30/Task1000	CHPSO	24,542.24	50	490.84
	PSO	30,011.71	83	361.59
PM2/VM6/Task1000	CHPSO	21,222.78	45	471.62
	PSO	30,450.00	75	406.00
PM20/VM80/Task1000	CHPSO	4,078.33	10	407.83
	PSO	29,220.00	50	584.40
PM20/VM80/Task3000	CHPSO	1,951.62	5	390.32
	PSO	87,810.00	70	1,254.43
PM20/VM80/Task5000	CHPSO	146,591.73	70	2,094.17
	PSO	145,830.00	100	1,215.25

compared with other algorithms. One possible reason is that these time-related metrics are influenced by underlying system factors—such as network latency, hardware limitations, and task dependencies—that tend to be optimized similarly across various scheduling approaches. For example, if tasks require a minimum runtime on specific VSs due to hardware limitations, improvements in scheduling logic may noticeably reduce the overall execution time.

In contrast, our method significantly improves resource utilization and energy efficiency, as evidenced by the marked differences in those metrics (with resource utilization showing an F-value of 13.076 and a p-value < 0.0001, and energy consumption approaching significance with a p-value of 0.057). For instance, evenly distributing tasks across VMs can avoid hotspots and reduce energy waste from underused servers, even if it does not shorten makespan when all VSs run at similar speeds. Overall, our approach—which integrates a workload-balancing heuristic with chi-squared-based task arrival modeling—appears to have a stronger impact on managing resources effectively rather than further reducing an already optimized makespan. This indicates that while our fitness function focuses on minimizing makespan, the true benefits of our algorithm lie in its enhanced resource allocation and energy savings, which collectively improve overall system efficiency.

5.8.2 Summary of Descriptive Statistics

Below is a summary table of the mean and standard deviation for each metric, showcasing CHPSO's efficiency (Table 10):

Table 9 ANOVA results of CHPSO

Metrics	F-value	p-value
Makespan	2.2053585762571744	0.10513495016460672
Resource Utilization	13.076084230343934	2.1590505794468298e-05
Response Time	2.3087634131138737	0.09337684945171885
Execution Time	1.9057938140876924	0.14883925164926384
Energy Consumption	2.7517300608480944	0.05671638186769293

Table 10 Mean and standard deviation for each metric

Metrics		HSGA	CHPSO	G-PSO	PSO	FCFS
Makespan	*mean*	19380.1	11159.39	96300.1	44885.87	22380.02
	std	17980.74	18518.10	106198.94	33346.21	20410.56
Resource Utilization	*mean*	83.96	86.67	74.69	50.33	80.58
	std	7.59	7.28	8.52	11.104	10.12
Response Time	*mean*	9775.96	3189.47	43489.80	16758.78	11245.23
	std	8998.02	5018.32	46933.415423	15288.57	10207.56
Execution Time	mean	66000	39677.34	205878	64664.34	66600
	std	53665.63	60602.90	214541.99	51844.26	53177.06
Energy Consumption	*mean*	1.831954e+07	6.273060e+06	4.894975e+07	3.732840e+07	1.713515e+07
	std	1.564876e+07	5.615100e+06	3.626457e+07	2.932114e+07	1.520005e+07

Table 10 reveals that CHPSO excels particularly in makespan, with a mean significantly lower than the others, suggesting superior efficiency in task completion times. Resource utilization for CHPSO is also impressive, showcasing efficient resource management compared to others like G-PSO and PSO, which show lesser efficiency in energy consumption and resource utilization respectively.

The high standard deviation values across most metrics, especially in execution time and energy consumption, suggest variability in algorithm performance under different conditions. Notably, G-PSO's higher mean in energy consumption and makespan points to potential inefficiencies, possibly due to its handling of larger or more complex workloads.

In summary, CHPSO's lower means in crucial metrics like makespan and energy consumption, combined with competitive resource utilization, underscore its potential for effectively managing demanding cloud computing environments. This analysis provides a clear perspective on each algorithm's strengths and weaknesses, aiding in better decision-making for task scheduling in cloud systems.

5.8.3 *Tukey's Honestly Significant Difference (HSD) Test*

Tukey's HSD test was conducted for detailed pairwise comparisons [43]. The Tukey HSD test results provide crucial insights into the comparative effectiveness of the different algorithms (HSGA, CHPSO, G-PSO, PSO, and FCFS) employed in our scenarios. Across different metrics, the outcomes suggest varying levels of performance improvement or decline, which can guide the refinement of scheduling strategies.

Figure 9 indicates significant differences in resource utilization among the algorithms. The statistical significance in resource utilization suggests that some algorithms are consistently better at managing resources than others, with CHPSO showing particularly strong performance. Further pairwise comparisons using Tukey's Honestly Significant Difference (HSD) test identified specific differences between CHPSO and other algorithms, validating the superior efficiency of CHPSO in resource management.

5.8.4 *Summary of Statistical Findings*

The review of ANOVA results shows that not all performance metrics exhibited statistically significant differences, suggesting that the advantages of CHPSO might not be a widespread approach across various scenarios. This indicates that while CHPSO holds promise, its performance may vary depending on specific operational contexts, prompting further evaluation and optimization. Nonetheless, the analysis strongly supports CHPSO's effectiveness in resource utilization, where it consistently excels compared to other algorithms. Despite similar performances in makespan, response time, and execution time, CHPSO's potential for improved energy efficiency highlights its appropriateness for energy consumption in cloud computing, emphasizing the need for continued development to fully leverage its capabilities.

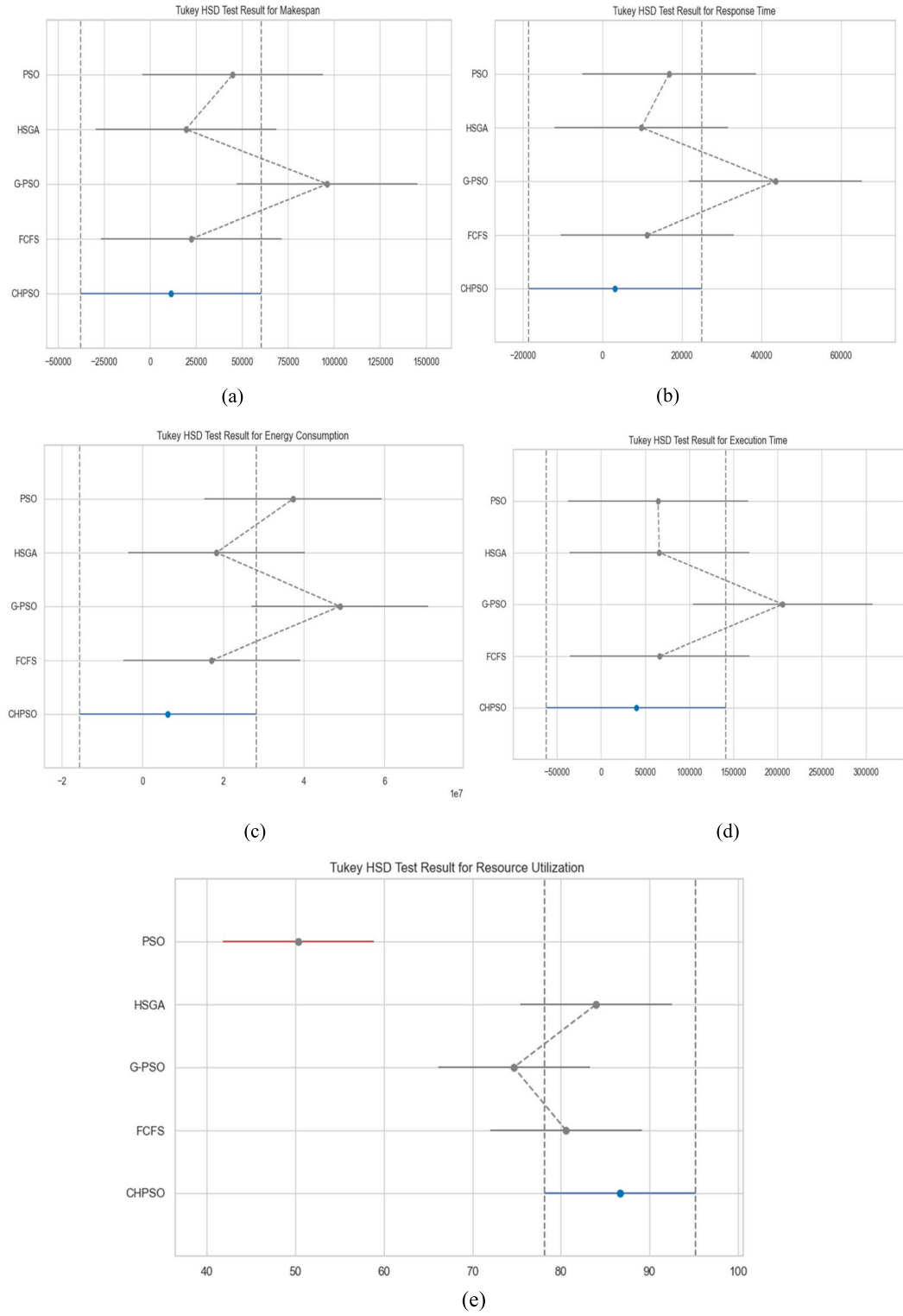


Fig. 9 Tukey HSD Test Results for Performance metrics: (**a**) Makespan- (**b**) Response Time- (**c**) Energy Consumption- (**d**) Execution Time- (**e**) Resource Utilization

6 Limitations and Future directions

While CHPSO has shown promising results, several limitations remain. Although the chi-squared distribution model generally helps avoid server overutilization, it may not capture all aspects of task arrival patterns in specific cloud environments. Additionally, the computational overhead associated with PSO optimization could be significant in extremely large-scale systems, potentially impacting real-time performance. Future work will explore alternative statistical models and optimization techniques to address these limitations. Even though our experiments demonstrated effective scalability, deploying CHPSO in highly dynamic and large-scale cloud systems may introduce further challenges, such as the need for more sophisticated communication protocols to reduce overhead and improve coordination between servers. Moreover, integrating additional load-balancing algorithms could further enhance scalability and efficiency.

Looking ahead, future work should investigate alternative statistical models and optimization techniques to address these limitations. For example, integrating advanced machine learning methods to dynamically predict workload variations and adjust scheduling parameters in real time could further refine the approach. This would allow for more adaptive and intelligent decision-making, leading to improved resource utilization and system performance [44]. In addition, developing multi-objective optimization frameworks that consider not only makespan but also cost, energy consumption, reliability, and security would offer a more comprehensive solution [45]. By incorporating such diverse factors, future research can ensure that scheduling decisions align with real-world constraints in fog and cloud environments. Although the current design of CHPSO focuses on resource and energy efficiency, future iterations might incorporate time-specific heuristics—such as prioritizing critical-path tasks or enabling dynamic VM scaling—to better address makespan and response time without sacrificing its core benefits. Finally, exploring hybrid cloud architectures that combine the strengths of both cloud and edge computing could significantly enhance scalability and responsiveness, particularly in large-scale deployments. These research directions promise to pave the way for more robust, adaptive, and efficient scheduling solutions in both fog and cloud environments.

7 Conclusion

In conclusion, current cloud task scheduling algorithms often struggle with imbalanced workloads, neglect of virtual server heterogeneity, and inefficiency in dynamic environments. This work introduces a CHPSO algorithm that addresses these challenges through a dual approach: modeling workloads with a chi-squared distribution for realistic scenarios and using heuristic balancing for strategic task allocation. Additionally, the algorithm integrates PSO to refine cloudlet-to-VS mapping, ensuring optimal task placement and maximizing resource utilization. CHPSO shows promise in reducing makespan, improving energy efficiency, and decreasing response times, efficiently allocating tasks among virtual machines by dynamically adjusting assignments to minimize completion times and enhance resource usage. Moreover, our comprehensive scalability assessment indicates that CHPSO is well-suited for large-scale cloud environments, maintaining high performance and efficient resource utilization across varying scenarios. By addressing identified challenges and continuing to refine our approach, we aim to further enhance the scalability and robustness of CHPSO for broader adoption in dynamic and complex cloud systems. However, further testing under varied conditions and with larger datasets is recommended to fully ascertain its advantages and areas for improvement. Additionally, implementing machine-learning techniques to enable the algorithm to dynamically adapt to changing cloud environments may lead to even more efficient task scheduling. By addressing these future directions, this work paves the way for a more efficient and dynamic cloud task-scheduling paradigm, ultimately contributing to the optimization of cloud-based applications.

Authors' Contributions Hind Mikram: Conceptualization, Methodology, Software, Formal analysis, Writing-original draft, Visualization, Investigation, Data Curation, Implementation.

Said El Kafhali: Conceptualization, Methodology, Supervision, Visualization, Investigation, Project administration, Formal analysis, Writing -review & editing.

Funding There is no funding for this research paper.

Data Availability No datasets were generated or analysed during the current study.

Declaration

Conflict of Interest/Competing Interests The authors declare no competing interests.

References

1. Katal, A., Dahiya, S., Choudhury, T.: Energy efficiency in cloud computing data centers: A survey on software technologies. Clust. Comput. **26**(3), 1845–1875 (2023). https://doi.org/10.1007/s10586-022-03713-0
2. Mokadem, R., Hameurlain, A.: A data replication strategy with tenant performance and provider economic profit guarantees in Cloud data centers. J. Syst. Softw. **159**, 110447 (2020). https://doi.org/10.1016/j.jss.2019.110447
3. Dhaya, R., Ujwal, U.J., Sharma, T., Singh, P., Kanthavel, R., Selvan, S., Krah, D.: Energy-efficient resource allocation and migration in private cloud data centre. Wireless Comm. Mobile Comp. **1–13**,(2022). https://doi.org/10.1155/2022/3174716
4. Fé, I., Matos, R., Dantas, J., Melo, C., Nguyen, T.A., Min, D., Choi, E., Silva, F.A., Maciel, P.R.M.: Performance-cost trade-off in auto-scaling mechanisms for cloud computing. Sensors **22**(3), 1221 (2022). https://doi.org/10.3390/s22031221
5. Usman Sana, M., Li, Z.: Efficiency aware scheduling techniques in cloud computing: A descriptive literature review. PeerJ Comp. Sci. **7**, e509 (2021). https://doi.org/10.7717/peerj-cs.509
6. Mikram, H., El Kafhali, S., Saadi, Y.: Server consolidation algorithms for cloud computing: Taxonomies and systematic analysis of literature. Int. J. Cloud Appl. Comput. (IJCAC) **12**(1), 1–24 (2022)
7. Geetha, P., Vivekanandan, S.J., Yogitha, R., Jeyalakshmi, M.S.: Optimal load balancing in cloud: Introduction to hybrid optimization algorithm. Expert Syst. Appl. **237**, 121450 (2024). https://doi.org/10.1016/j.eswa.2023.121450
8. Khédiri, N., & Zaghdoud, M.: Survey of uncertainty handling in cloud service discovery and composition. Preprint at https://arxiv.org/pdf/1501.01537 (2015)
9. Gupta, S., Dileep, A.D.: Long range dependence in cloud servers: A statistical analysis based on google workload trace. Computing **102**(4), 1031–1049 (2020). https://doi.org/10.1007/s00607-019-00779-4
10. Ray, S., Alshouiliy, K., Roy, A., AlGhamdi, A., Agrawal, D. P.: Chi-squared based feature selection for stroke prediction using AzureML. 2020 intermountain engineering, technology and computing (IETC), Orem, UT, USA, 02-03 October 2020. https://doi.org/10.1109/IETC47856.2020.9249117
11. Mikram, H., El Kafhali, S., Saadi, Y.: A hybrid algorithm based on PSO algorithm and chi-squared distribution for tasks consolidation in cloud computing environment. 2023 IEEE 6th international conference on cloud computing and artificial intelligence: Technologies and applications (CloudTech) (p. 1-6), Marrakech, Morocco, 21-23 November 2023. https://doi.org/10.1109/CloudTech58737.2023.10366164
12. Yang, X.: Probability and distributions. In engineering simulation and its applications (pp. 173–186). Academic Press. Elsevier Inc (2024). https://doi.org/10.1016/b978-0-44-314084-6.00019-x
13. Mikram, H., El Kafhali, S., Saadi, Y.: Processing time performance analysis of scheduling algorithms for virtual machines placement in cloud computing environment. In international conference on big data and internet of things (pp. 200–211). Cham: Springer International Publishing (2022)
14. Mikram, H., El Kafhali, S., Saadi, Y.: Performance analysis of scheduling algorithms for virtual machines and tasks in cloud computing. In The international conference of advanced computing and informatics (pp. 278–289). Cham: Springer International Publishing (2022)
15. Pirozmand, P., Hosseinabadi, A.A.R., Farrokhzad, M., Sadeghilalimi, M., Mirkamali, S., Slowik, A.: Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing. Neural Comput. Appl. **33**(19), 13075–13088 (2021). https://doi.org/10.1007/s00521-021-06002-w
16. Krishna, B.S.R., Krishna, T.M., Sri, J.B., Priya, B.V.: Task Scheduling Using TVIW-PSO in Cloud Computing. In 2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI) (pp. 1–7). IEEE, Chennai, India, 09-10 May 2024
17. Khaledian, N., Khamforoosh, K., Akraminejad, R., Abualigah, L., Javaheri, D.: An energy-efficient and deadline-aware workflow scheduling algorithm in the fog and cloud environment. Computing **106**(1), 109–137 (2024). https://doi.org/10.1007/s00607-023-01215-4
18. Saad, M., Enam, R.N., Qureshi, R.: Optimizing multi-objective task scheduling in fog computing with GA-PSO algorithm for big data application. Front. Big Data **7**, 1358486 (2024). https://doi.org/10.3389/fdata.2024.1358486
19. Khaledian, N., Razzaghzadeh, S., Haghbayan, Z., Völp, M.: Hybrid Markov chain-based dynamic scheduling to improve load balancing performance in fog-cloud environment. Sustain. Comp.: Inform. Syst. **45**, 101077 (2025). https://doi.org/10.1016/j.suscom.2024.101077
20. Chen, Z., Hu, J., Min, G., Luo, C., El-Ghazawi, T.: Adaptive and efficient resource allocation in cloud datacenters using actor-critic deep reinforcement learning. IEEE Trans. Parallel Distrib. Syst. **33**(8), 1911–1923 (2021)
21. Chen, Z., Hu, J., Min, G., Zomaya, A.Y., El-Ghazawi, T.: Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning. IEEE Trans. Parallel Distrib. Syst. **31**(4), 923–934 (2019)
22. Chen, Z., Xiong, B., Chen, X., Min, G., Li, J.: Joint computation offloading and resource allocation in multi-edge smart communities with personalized federated deep reinforcement learning. IEEE Trans. Mob. Comput. **23**(12), 11604–11619 (2024)
23. Khaledian, N., Voelp, M., Azizi, S., Shirvani, M.H.: AI-based & heuristic workflow scheduling in cloud and fog computing: A systematic review. Clust. Comput.

27(8), 10265–10298 (2024). https://doi.org/10.1007/s10586-024-04442-2
24. Hussain, M., Wei, L.-F., Rehman, A., Abbas, F., Hussain, A., Ali, M.: Deadline-constrained energy-aware workflow scheduling in geographically distributed cloud data centers. Futur. Gener. Comput. Syst. **132**, 211–222 (2022). https://doi.org/10.1016/j.future.2022.02.018
25. Thakur, A., Goraya, M.S.: RAFL: A hybrid metaheuristic based resource allocation framework for load balancing in cloud computing environment. Simul. Model. Pract. Theory **116**, 102485 (2022). https://doi.org/10.1016/j.simpat.2021.102485
26. Masoudi, J., Barzegar, B., Motameni, H.: Energy-aware virtual machine allocation in DVFS-enabled cloud data centers. IEEE Access **10**, 3617–3630 (2022). https://doi.org/10.1109/ACCESS.2021.3136827
27. Mukherjee, D., Ghosh, S., Pal, S., Aly, A.A., Le, D.-N.: Adaptive scheduling algorithm based task loading in cloud data centers. IEEE Access **10**, 49412–49421 (2022). https://doi.org/10.1109/ACCESS.2022.3168288
28. Zhang, B., Zeng, Z., Shi, X., Yang, J., Veeravalli, B., Li, K.: A novel cooperative resource provisioning strategy for multi-cloud load balancing. J. Parallel Distri. Comp. **152**, 98–107 (2021). https://doi.org/10.1016/j.jpdc.2021.02.003
29. Sathish, N., Valarmathi, K.: Detection of Intrusion behavior in cloud applications using Pearson's chi-squared distribution and decision tree classifiers. Pattern Recog. Lett. **162**, 15–21 (2022). https://doi.org/10.1016/j.patrec.2022.08.008
30. Kathavate, P.N.: Optimal prediction of viral host from genomic datasets using ensemble classifier. Adv. Eng. Softw. **175**, 103273 (2023). https://doi.org/10.1016/j.advengsoft.2022.103273
31. Dey, A.K., Gupta, G.P., Sahu, S.P.: Hybrid meta-heuristic based feature selection mechanism for cyber-attack detection in IoT-enabled networks. Procedia Comp. Sci. **218**, 318–327 (2023). https://doi.org/10.1016/j.procs.2023.01.014
32. Zeedan, M., Attiya, G., El-Fishawy, N.: Enhanced hybrid multi-objective workflow scheduling approach based artificial bee colony in cloud computing. Computing **105**(1), 217–247 (2023). https://doi.org/10.1007/s00607-022-01116-y
33. Harchol-Balter, M.: Open problems in queueing theory inspired by datacenter computing. Queueing Syst. **97**(1), 3–37 (2021). https://doi.org/10.1007/s11134-020-09684-6
34. Orsi, C.: New DeveloPSents on the non-central chi-squared and beta distributions. Austrian J. Stat. **51**, 35–51 (2021). https://doi.org/10.17713/ajs.v51i1.1106
35. Tarekegn Nigatu, D., Gemechu Dinka, T., Luleseged Tilahun, S.: Convergence analysis of particle swarm optimization algorithms for different constriction factors. Front. Appl. Math. Stat. **10**, 1304268 (2024). https://doi.org/10.3389/fams.2024.1304268
36. Smith, W.L.: On the distribution of queueing times. Math. Proc. Cambridge Philos. Soc. **49**(3), 449–461 (1953). https://doi.org/10.1017/S0305004100028620
37. Ghorbannia Delavar, A., Aryan, Y.: HSGA: A hybrid heuristic algorithm for workflow scheduling in cloud systems. Clust. Comput. **17**(1), 129–137 (2014). https://doi.org/10.1007/s10586-013-0275-6
38. Zhong, Z., Chen, K., Zhai, X., Zhou, S.: Virtual machine-based task scheduling algorithm in a cloud computing environment. Tsinghua Sci. Technol. **21**(6), 660–667 (2016). https://doi.org/10.1109/TST.2016.7787008
39. Gökalp, O.: Performance evaluation of heuristic and metaheuristic algorithms for independent and static task scheduling in cloud computing. 2021 29th signal processing and communications applications conference (SIU), (p. 1–4), Istanbul, Turkey, 09-11 June 2021. https://doi.org/10.1109/SIU53274.2021.9477821
40. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw.: Pract. Exper **41**(1), 23–50 (2011). https://doi.org/10.1002/spe.995
41. Tian, D., Shi, Z.: MPSO: Modified particle swarm optimization and its applications. Swarm Evol. Comput. **41**, 49–68 (2018)
42. Abid, M., El Kafhali, S., Amzil, A., Hanini, M.: Solving the 0/1 knapsack problem using metaheuristic and neural networks for the virtual machine placement process in cloud computing environment. Math. Probl. Eng. **2023**, 1–17 (2023). https://doi.org/10.1155/2023/1742922
43. Lu, C., Zhu, J., Huang, H., Sun, Y.: A multi-hierarchy particle swarm optimization-based algorithm for cloud workflow scheduling. Futur. Gener. Comput. Syst. **153**, 125–138 (2024). https://doi.org/10.1016/j.future.2023.11.030
44. Mikram, H., El Kafhali, S.: AI-Driven autonomous systems for optimal management of fog computing resources. In 2024 7th international conference on advanced communication technologies and networking (CommNet) (pp. 1–7). IEEE, Rabat, Morocco, 04-06 December 2024
45. Mikram, H., El Kafhali, S., Saadi, Y.: HEPGA: A new effective hybrid algorithm for scientific workflow scheduling in cloud computing environment. Simul. Model. Pract. Theory **130**, 102864 (2024)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.