Received 6 August 2024, accepted 10 September 2024, date of publication 18 September 2024, date of current version 7 October 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3462894

RESEARCH ARTICLE

Energy-Aware Microservice-Based SaaS Deployment in a Cloud Data Center Using Hybrid Particle Swarm Optimization

A. ALZAHRANI[1,2] **AND M. TANG**[1]
[1]School of Computer Science, Faculty of Science, Queensland University of Technology, Brisbane, QLD 4000, Australia
[2]Department of Computer Science, Prince Sattam Bin Abdulaziz University, Al-Kharj 16278, Saudi Arabia

Corresponding author: A. Alzahrani (amalsaleha.alzahrani@hdr.qut.edu.au)

This work was supported by Saudi Cultural Mission and Prince Sattam Bin Abdulaziz University, Saudi Arabia.

ABSTRACT The deployment of software as a service (SaaS) using a microservice architecture offers several benefits, including scalability, flexibility, and ease of maintenance. One of the most important advantages of the new microservice-based SaaS deployment is that the increase in energy consumption incurred by the deployment of a new microservice-based SaaS can be considered. With the aim of reducing the increase in energy consumption, this paper proposes a new method, namely Hybrid Particle Swarm Optimization (HPSO), to solve the microservice-based SaaS deployment problem. The HPSO incorporates adaptive inertia weight, cognitive, and social parameters to balance the trade-off between exploration and exploitation during the optimization process. Furthermore, the HPSO incorporates a local optimizer to improve the best global solution within the swarm, with a specific emphasis on improving energy efficiency. To evaluate the performance of the HPSO, we have implemented it and compared it with a GA method by experiment. The experimental results have shown that the HPSO can further reduce the increase in energy consumption by 3.68% compared to GA.

INDEX TERMS Cloud computing, data center, deployment, energy consumption, hybrid particle swarm optimization, microservice, optimization, software as a service.

I. INTRODUCTION

Cloud computing services are broadly classified into Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [1]. Of these, SaaS has garnered substantial attention from both providers and consumers in recent years. SaaS involves the deployment of services that allow users to access off-premise software via the Web. This model fundamentally alters the dynamics of software ownership and usage. In traditional on-premise software models, customers purchase software licenses, install the software locally, and bear upfront setup and installation costs. In contrast, in the SaaS model, users subscribe to or pay on a usage basis, with ownership and management transferred to service providers.

The introduction of microservice architecture has significantly changed the landscape of software development and deployment. This architectural paradigm empowers developers to conceptualize software as a group of loosely coupled microservices, each of which has distinct functionalities and interacts with other microservices through Application Programming Interfaces (API) [2]. The autonomy and scalability inherent to microservices facilitate the continuous delivery of intricate and expansive software solutions. In particular, cloud computing has emerged as the predominant environment for microservice deployment, providing users with unparalleled access to resources on a pay-per-use basis [3].

The microservice architecture has many significant advantages. One of the advantages is its scalability. Scalability refers to the ability of a system to handle an increasing workload by adding resources. The microservice architecture improves scalability in several ways. First, each microservice

The associate editor coordinating the review of this manuscript and approving it for publication was Mu-Yen Chen.

can be scaled independently based on its specific resource requirements. This allows for more efficient use of resources, as only the microservices that experience increasing workload need to be scaled. In addition, microservices can be scaled by adding more instances of the microservice. Another advantage of microservice is its modularity, which provides a multitude of benefits that enhance overall efficiency, maintainability, and scalability of applications. A third advantage of microservice is its flexibility for deployment. Microservices can be deployed continuously, enabling faster delivery of new features and fixes.

Cloud data centers serve as the foundation for modern computing infrastructure, facilitating a wide range of online services, applications, and storage solutions that underpin our digital lives. This technological advance comes with a pressing challenge, the exponential growth of energy consumption within these sprawling data centers. The expansion of data centers to meet increasing digital demands results in significant electricity consumption to power servers, communication networks, and other critical equipment. This surge in energy consumption raises substantial concerns about sustainability, environmental impact, and operational costs. The energy consumed by data centers not only impacts their operational efficiency, but also contributes significantly to the global carbon footprint.

Energy usage within a data center can be broadly classified into two primary segments: the energy expended by IT equipment, encompassing servers, communication networks, and storage; and the energy employed by infrastructure amenities, encompassing cooling and power conditioning systems. According to data published by the Info-tech group [4], [5], servers, storage devices and networking collectively constitute 36% of the total energy consumption in a data center, as illustrated in Figure 1.

The impact of deploying microservice-based SaaS within a cloud data center extends beyond operational efficiency to considerations of energy consumption. The compute servers that host the microservices deployed have a substantial increase in energy consumption. Furthermore, the energy consumed by network devices that facilitate communication between dispersed microservices residing on disparate compute servers contributes significantly to the increase in energy consumption in the data center.

Traditionally, SaaS deployment is done by SaaS providers on virtual machines (VMs), which are provided by a cloud data center as an IaaS. Therefore, SaaS providers do not have the privilege of identifying the specific compute servers that host these virtual machines. As a result, they cannot do anything with the increase in energy consumption associated with the deployment of new microservice-based SaaS. To this end, in our previous study, we proposed a paradigm shift that transfers the responsibility for microservice-based SaaS deployment from SaaS providers to the cloud provider, and proposed a genetic algorithm (GA) to solve the new SaaS deployment problem [6]. The objective was to minimize the increase in energy consumption associated with a new microservice-based SaaS deployment.

To explore alternative computational intelligence approaches, this paper presents a hybrid particle swarm optimization (HPSO) approach. It is generally believed that optimization problems where the objective function is relatively smooth and does not have many sharp discontinuities or steep gradients can be more effectively tackled by PSO. In addition, PSO is suitable for dynamic and time-varying optimization problems, as PSO can adapt more readily to changes in dynamic and time-varying environments.

The HPSO integrates a local optimizer into the traditional PSO framework, specifically designed to enhance the best global solution within the swarm. This hybridization allows us to harness the strengths of PSO's global exploration capabilities while fine-tuning solutions locally to achieve improved energy efficiency. In this paper, where optimizing energy consumption in cloud data centers is critical, the HPSO's ability to balance exploration and exploitation effectively aligns with our objective of achieving sustainable and efficient SaaS deployments.

To evaluate the performance of the HPSO, we have implemented it and compared it with the GA by experiment. The experimental results have shown that the performance of this HPSO outperforms that of the GA for all randomly generated test problems.

The remainder of the paper is arranged as follows: Section II discusses related work. Section III presents the formulation of the new SaaS deployment problem. The HPSO approach is detailed in Section IV. Section V shows the design of the experiments and analyzes the experimental results. Finally, Section VI concludes the paper.

II. RELATED WORK

The initial attempt to address the novel challenge of deploying microservice-based SaaS in cloud data centers was made by [6]. In the research, the microservice-based SaaS deployment was formulated as a multi-objective combinatorial optimization problem and a genetic algorithm (GA) was proposed to solve the problem. The objective was to reduce the increase in energy in compute servers and network devices in the data center resulting from the deployment of the microservice-based SaaS. The proposed GA achieved a 37.55% reduction in energy usage compared to conventional deployment methods. The deployment of composite SaaS was investigated by [7] who divided SaaS into interconnected application components (AC) and data components (DC). They used a Genetic Algorithm (GA) to solve the complex problem and developed a way to resolve infeasible solutions, and later improved the efficiency of GA using an iterative and parallel Cooperative Co-Evolutionary Algorithm (CCEA) [8], [9]. A subsequent study by [10] addressed the placement of composite SaaS using an Ant Colony System (ACS). Their experiments demonstrated that

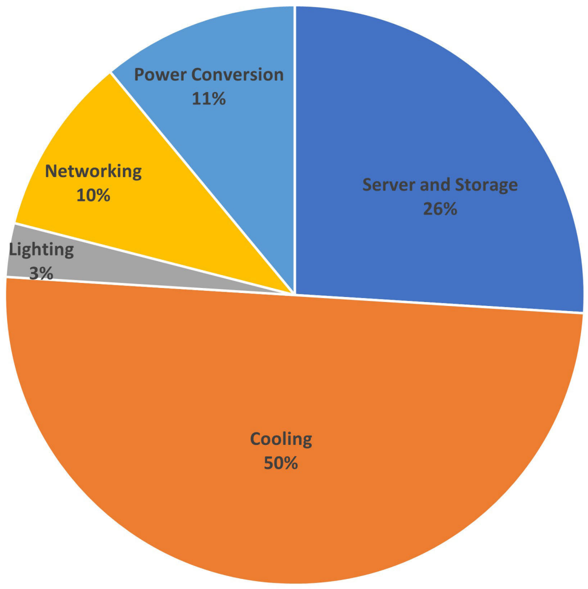

FIGURE 1. **Data centre energy consumption.**

the ACS outperformed the classical GA in terms of Estimated Total Execution Time (ETET).

In [11] a security-aware SaaS placement method was proposed which broke down the SaaS placement problem into two interactive subproblems, and a multiswarm PSO was proposed to tackle the SaaS placement problem. A cooperative learning strategy was also integrated with the PSO to allow the exchange of information between subswarms. The solution developed by the multiswarm PSO exhibited superior quality and scalability compared to those of existing state-of-the-art approaches. The study by [12] focused on the security-centered deployment, exploring the positioning of an application composed of interconnected components. The goal was to find a deployment method that deals with both the placement of the application and the configuration of the security controls. They considered three types of security controls: hardware-based; software-based; and a combination of hardware- and software-based security controls. This study investigated the impact of integrating specific security measures on the resource demands of components or connectors. The deployment challenge was addressed using a mixed-integer quadratic program. Through experimentation, their approach showcased the ability to automatically discover various deployment solutions that meet consistency, capacity, and security requirements. In [13], the researchers proposed a new Privacy-Preserving Reputation Updating (PPRU) scheme for cloud-assisted vehicular networks based on the Elliptic Curve Cryptography (ECC) and Paillier algorithms, in which the reputation feedbacks are collected and preprocessed by the honest but serious Cloud Service Provider (CSP) in a privacy-preserving manner.

In [14] a particle swarm optimization technique called composite particle (PSO-CP) was proposed to improve the performance of composite SaaS. The PSO-CP algorithm showed better performance in terms of resource utilization and execution time compared to other algorithms. The Hybrid Genetic Simulation Annealing Algorithm (GASA) presented in [15] successfully reduced costs related to the hardware and communication of virtual machines while demonstrating excellent solution quality. However, the performance of PSO-CP and GASA was similar when it came to small SaaS components. Neither of these methods took into account the interdependencies between servers and the interdependencies between connected SaaS components.

A multi-swarm PSO was also adopted by [16] with the notion that a separate swarm is responsible for being close to local optima to do further exploration. Empirical findings indicated that the multi-swarm adopted technique resulted in an average total execution time that was up to 83% better than the solution produced by a GA and up to 40% compared to the parallel CCEA. A PSO-based approach for the placement of a cost-effective SaaS was presented by [17] whose goal was to reduce the total costs of the SaaS provider during the deployment of a SaaS. During experiments, the PSO-based approach was compared with the greedy approach for various SaaS components, and the results obtained indicated that the PSO approach outperformed the greedy algorithm.

To maintain a balance between the isolation level of application components and resource sharing, [18] proposed a model-based algorithm together with meta-heuristic solutions to solve the cloud-based application component deployment problem. As an initial step, an open multiclass queuing network (QN) model was proposed to calculate the average number of requests permissible for accessing the component, while adhering to the specified degree of isolation. The metaheuristic search was then executed to achieve the optimal solution for deploying the component, ensuring the utmost degree of isolation and the maximum permissible number of requests. Experiments demonstrated that the model-based algorithm exhibits enhanced robustness and superior quality when combined with a metaheuristic starting from an initial greedy solution SA (Greedy), as opposed to one beginning with random solutions SA (Random). In situations where there are time and resource limitations, such as dynamic real-time environments or resource-constrained environments, simulated annealing is more effective than hill climbing in generating solutions that are resilient and consistent.

In [18], the researchers argued that their work was more valuable as a decision support system for making crucial considerations about scalability, provisioning of necessary components, and decommissioning of unused components. This paper stood out because it specifically targeted the initial deployment of microservice-based SaaS, with the aim of reducing the increase in energy, subject to resource and communication constraints.

The research in [19] explored the challenge of deploying new service-based applications by assigning specific processes to a private cloud and others to a public cloud. Their primary objective was to save costs and provide the best service placement options with the shortest execution time.

An enhanced Binary Particle Swarm Optimization (E-BPSO) algorithm was proposed to integrate modifications influenced by continuous PSO to update the equations that determine the positions of the particles. The experiments showed that the E-BPSO algorithm performed exceptionally in terms of both cost effectiveness and execution speed. The deployment strategy introduced by [19] involves assigning each service to a public or private cloud, although the specifics of how these services would be deployed within the selected cloud were not specified.

The study by [20] approached the service placement problem as a budget-constrained optimization problem to minimize service response time in distributed clouds, proposing a repair-based GA. Whenever a constraint violation was detected, the repair process was initiated to identify noncritical services and relocate them to more cost-effective virtual machines (VMs) or VMs with lower resource requirements. This approach helped mitigate constraint violations and maintain service performance. The experimental results demonstrated that the GA outperformed the benchmark algorithm in terms of total response time, indicating its effectiveness in optimizing service placement in distributed clouds. It is important to note that the work of [20] only considered the budget constraint and did not address violations of the resources and communication constraints. The repair algorithm was not specifically designed to detect and resolve such violations.

The authors of [21] addressed the problem of Microservice-based Application Workflow Scheduling problem for a minimum end-to-end delay under a user-specified Budget Constraint in public clouds (MAWS-BC) and developed a heuristic scheduling method to solve the problem. Their research focused only on assigning functions of the application to microservices, and the placement of microservices on the virtual machine has not been addressed. Reference [22] addressed the microservice placement problem by considering complex dependencies between services and support for multiple instances of each service deployed on different servers. The average response time was chosen as the main optimization goal, as it plays a crucial role in both cloud and edge computing. Their research problem was converted to a quadratic sum-of-rations fraction, and two greedy algorithms were proposed to solve it. The two greedy algorithms, namely, the depth-first search (DFS) placement algorithm and the breadth-first search (BFS) placement algorithm, vary in the sequence of service deployment. The BFS-based algorithm initiated the deployment with the service that has the least computing resources and lacks predecessors. In the DFS-based algorithm, the services were deployed in the order of the function chain. These methods began by identifying the optimal server capable of hosting a particular service, ensuring that the chosen server possesses sufficient computing resources to support at least one instance of the service. Once an instance of a service is deployed on a chosen server, all the best servers for the predecessors and successors of the deployed service will be changed. Four sets of experiments were conducted to evaluate the performance of the proposed algorithms. These sets varied according to the number of users, the server resource capacities, the number of servers and microservices, and the number of functions called by the users.

The research by [23] proposed a method to enhance the parallel execution of SaaS components while reducing communication costs across services. They presented two graph structures: the Service Dependency Graph (SDG); and, the Service Concurrence Graph (SCG). The SDG measures communication cost and execution time between different services, while the SCG determines possible candidates for parallel execution. These graphs can be used to co-deploy interdependent services on the same virtual machine by applying the k-cut principle. Although this approach performed significantly well in terms of service response time, in large- scale and complex systems such as the cloud, the SaaS placement problem cannot be adequately described using only graphs. Thus, this technique may not fully accommodate the ongoing, ever-changing, and scalable characteristics of cloud services.

The study by [24] introduced a microservices deployment strategy in a cloud-edge environment that prioritizes Quality of Service (QoS). Their research examined three primary concerns about the deployment of microservices: the initial deployment process, resource contention among microservices co-located on a single server, and the uneven allocation of resources.The researchers proposed the Nautilus system as a solution to address these problems. The Nautilus system consists of a communication-aware microservice mapper, a contention-aware resource manager, and a load-aware microservice scheduler. The communication-aware microservice mapper was developed to solve the initial problem by dividing the microservices graph into partitions based on the strength of communication between microservices. This ensures that closely coupled microservices are grouped together in the same node. Within the hosted node, resource management assigns computational resources to microservices. The issue of allocation was defined as a multiobjective optimization problem, and a technique based on reinforcement learning was proposed to address it. If the server's resources are being excessively used, the load-aware microservice scheduler is programmed to transfer microservices to another server that has enough capacity to handle them. Empirical findings demonstrated that the Nautilus system effectively decreased computing resources by 23.9%. The primary objective of the study conducted by [24] was to implement microservices in a way that optimizes computational and network resources, while also guaranteeing the quality of service (QoS). The research described in this paper is different from the study conducted by [24] in that it aims to reduce the energy consumption in compute servers and network devices while considering limitations in resources and communication.

In [25], researchers investigated the deployment of composite applications on federated edge servers where services within an application are hosted on edge servers controlled by a group of providers. Each provider has the autonomy to establish its own policies concerning data protection and the preservation of privacy. The main objective of this approach was to optimize QoS for applications, specifically addressing latency and privacy concerns, while also enhancing the efficiency of the infrastructure by minimizing power consumption. A power-aware heuristic named 'Thea' was developed by [25] to address the deployment problem in which the methodology begins by prioritizing applications based on their privacy and latency requirements. Following the classification of applications by latency and privacy criteria, 'Thea' systematically iterates through the application list, selecting appropriate edge servers to host each service. In this process, 'Thea' utilizes a cost function that arranges edge servers according to the frequency of SLA violations. The technique employed by "Thea" for ensuring service privacy involves giving a higher cost to edge servers that are trusted by a greater proportion of unprovisioned services. These servers are more likely to be crucial in later placement iterations. According to their experimental results, 'Thea' achieved close-to-optimal results according to their experimental results, decreasing latency and privacy concerns, and optimizing energy consumption on federated edges. The research presented in this paper differs from [25] in two significant aspects; the deployment system and the objectives. That is, this research focuses on deploying microservice-based in centralized data centers with unlimited computation and storage resources. Second, the main objectives are both minimizing energy increase in cloud servers, and minimizing energy increase in network devices which was not undertaken by [25].

III. RESEARCH PROBLEM AND FORMULATION

Microservice-based SaaS deployment in a data center is a popular option for businesses looking to take advantage of the benefits of cloud computing. However, an important consideration when deploying microservice-based SaaS in a data center is how the deployment of new microservices would affect the energy consumption of the data center. The deployment of microservice-based SaaS in a data center can increase energy consumption in several ways. Firstly, when deploying microservice-based SaaS in a data center, this can lead to an increase in workload on servers that host newly deployed microservices, which, in turn, increases energy consumption. Secondly, microservices typically interconnect and require a data center network for communications, which may require the use of additional networking equipment, such as routers and switches. These networking devices also consume energy, contributing to the overall energy consumption of the data center.

This paper investigates the problem of deploying microservice-based SaaS with the aim of minimizing the increase in energy consumption across compute servers and network devices caused by the deployment. Given the growing focus on sustainability and energy efficiency, it is vital to understand and address the energy consumption associated with the deployment of microservice-based SaaS. In a typical SaaS deployment scenario, SaaS providers are responsible for deploying their SaaS into VMs rented from cloud providers. However, in this setting, SaaS providers have no knowledge or control over which compute servers will execute the rented VMs. Due to this random deployment scenario, the resulting increase in energy consumption from the deployment cannot be considered.

In our paper, we shift the responsibility of the SaaS deployment from SaaS providers to cloud providers. The main focus of this new SaaS deployment approach is to determine the most efficient way to allocate each microservice in a SaaS to a compute server in the data center. The goal is to minimize the overall increase in energy consumption that results from this SaaS deployment and demands careful consideration of the resource requirements of each microservice and their data dependencies to optimize the allocation and minimize the energy increase.

In this section, we formulate the new microservice-based SaaS deployment problem as a constrained combinatorial optimization problem. In addition, an energy model is presented that calculates the energy increase in compute servers and network devices.

A. DATA CENTER NETWORK

A cloud data center can be depicted as an undirected weighted graph denoted as $G = (V, E)$, where V is a set consisting of compute servers (P) and network devices (N), expressed as $V = (P \cup N)$. Compute servers are represented by a set of vertices $P = \{p_1, p_2, \ldots, p_m\}$, where each vertex p_m corresponds to a specific compute server. Similarly, network devices are denoted by a set of vertices $N = \{n_1, n_2, \ldots, n_z\}$, where each vertex n_z represents a network device within the data center. A set of edges (E) in this graph symbolizes communication links, connecting a compute server to a network device or linking pairs of network devices within the data center. The total number of such edges is n. To determine the bandwidth capacity of each communication link, a weight is assigned to each edge. The weight of an edge connecting the vertex v_i and the vertex v_j is denoted as b_{ij}, which signifies the available bandwidth for data transmission between these specific endpoints. The values of i and j are restricted within the range of 1 to m for compute servers and 1 to z for network devices, respectively. The attributes of a compute server p_i are encapsulated in a tuple representation: $p_i = \langle c_i^{cpu}, c_i^{mem}, c_i^{disk}, e_i^{min}, e_i^{max}, w_i^{cpu}, o_i \rangle$. Each attribute carries specific information. Details of these attributes are outlined in Table 1, which offers a comprehensive breakdown of the attributes of a compute server.

Attributes of a network device n_k are represented in a tuple, $n_k = \langle n_k^{type}, e_k^{min}, e_k^{max}, d_k^{rate}, l_k, p_k \rangle$.

Table 2 shows a breakdown of these attributes.

TABLE 1. **Compute server attributes.**

Attribute	Description
c_i^{cpu}	Compute capacity of the compute server p_i
c_i^{mem}	Memory capacity of the compute server p_i
c_i^{disk}	Disk space capacity of the compute server p_i
e_i^{min}	Minimum energy consumption of the compute server p_i
e_i^{max}	Maximum energy consumption of the compute server p_i
w_i^{cpu}	Current workload of the compute server p_i
o_i	Power state of the compute server p_i

TABLE 2. **Network device attributes.**

Attribute	Description
n_k^{type}	Type of n_k
d_k^{rate}	Maximum capacity of the network device n_k
e_k^{min}	Minimum energy consumption of the network device n_k
e_k^{max}	Maximum energy consumption of the network device n_k
l_k	Traffic load of n_k
p_k	Power state of the network device n_k

B. MICROSERVICE-BASED WORKFLOW

A microservice-based SaaS is modeled as an undirected graph $W = (S, D)$. In this graph, $S = s_1, s_2, \ldots, s_p$ is a set of vertices that represent the microservices in W, and D is the set of undirected edges that reflect the communication dependencies between these microservices. An edge $e_{ij} = (s_i, s_j) \in D$ is present only when there is a data dependency between microservice s_i and microservice s_j. Each vertex on the graph W corresponds to a microservice s_j defined by its CPU, memory, and disc requirements, denoted r_j^{cpu}, r_j^{mem}, and r_j^{disk}, respectively. $B(e_{ij})$ represents the amount of bandwidth needed for the communication link between the microservices s_i and s_j. Figure 2 shows a microservice-based SaaS graph, where each node represents a microservice. The tuple inside each node specifies the requirements for CPU, memory, and disc space of the microservice. The numerical value assigned to each edge in Figure 2 indicates the specific bandwidth requirement for that particular edge.

C. ENERGY INCREASE MODEL

In order to determine the amount of energy that is consumed when deploying a microservice-based SaaS in a data center, we develop an energy model that measures the increase in energy consumption for both compute servers and network devices. The energy consumption of a compute server p_i is directly proportional to its computing workload when it is powered on. However, if the server is powered off, its energy consumption is zero. Our study concentrates on quantifying the increase in energy that arises from an increase in a server's workload. The energy consumption of p_i under the current workload w_i^{cpu}, represented as $E^p(w_i^{cpu})$, can be computed using Equation (1):

$$E_i^p(w_i^{cpu}) = e_i^{min} + (e_i^{max} - e_i^{min}) \times (w_i^{cpu}) \times o_i \tag{1}$$

Therefore, when the workload of p_i is increased by Δw_i^{cpu}, the energy consumption of p_i will correspondingly rise by ΔE_i^p, as specified in Equation (2):

$$\begin{aligned}\Delta E_i^p &= E_i^p(w_i^{cpu} + \Delta w_i^{cpu}) - E_i^p(w_i^{cpu}) \\ &= (e_i^{max} - e_i^{min}) \times (\Delta w_i^{cpu}) \times o_i\end{aligned} \tag{2}$$

The energy consumption of a network device n_k is determined by adding up the energy consumption of its static and dynamic components. The static energy consumption is mainly dependent on the characteristics of the network device, which include the energy consumption of the chassis, the line card, and the port configuration [26]. The static energy consumption remains constant regardless of the increase in data rate that comes from the deployment of microservice-based SaaS [6]. Therefore, the static energy consumption of n_k is constant both before and after the deployment. Thus, the energy consumption resulting from static factors will not be included in the energy increaseÂ model for network devices. The energy consumption of n_k, as defined in Equation (3), is exactly proportional to the amount of traffic that passes through n_k. The traffic load l_k is the ratio of current traffic passing through n_K to the maximum capacity d_k^{rate}.

$$E_k^n(l_k) = (e_k^{min} + (e_k^{max} - e_k^{min}) \times l_k) \times p_k \tag{3}$$

Therefore, if the traffic load passing n_k is increased by Δl_k, the energy consumption of n_k will increase by ΔE_k^n, which is defined in Equation (4):

$$\begin{aligned}\Delta E_k^n &= E_k^n(l_k + \Delta l_k) - E_k^n(l_k) \\ &= (e_k^{max} - e_k^{min}) \times \Delta l_k \times p_k\end{aligned} \tag{4}$$

The total increase in energy E is calculated using equations (2) and (4). This calculation takes into account both the increase in energy in the compute servers and the increase in energy in the network devices, as defined in Equation (5).

$$E = \sum_{x=1}^{m} \Delta E_x^p + \sum_{y=1}^{z} \Delta E_y^n \tag{5}$$

where m and z are the number of compute servers and number of network devices, respectively.

D. MODELLING OF THE CONSTRAINTS OF A FEASIBLE SAAS DEPLOYMENT

1) RESOURCE CONSTRAINTS

A feasible deployment plan for a microservice-based SaaS must comply with the following three constraints on available resources. Multiple microservices can be deployed on a compute server p_i, where $1 \leq i \leq m$. However, the aggregate resource demands of these microservices to be created in p_i must not exceed the existing available resources of p_i. The resource needs can be categorized into three types: CPU demand, main memory requirement, and disc requirement. The three constraints on resources:

$$\sum_{j=1}^{|S^i|} r_j^{cpu} \leq p_i^{cpu} \tag{6}$$

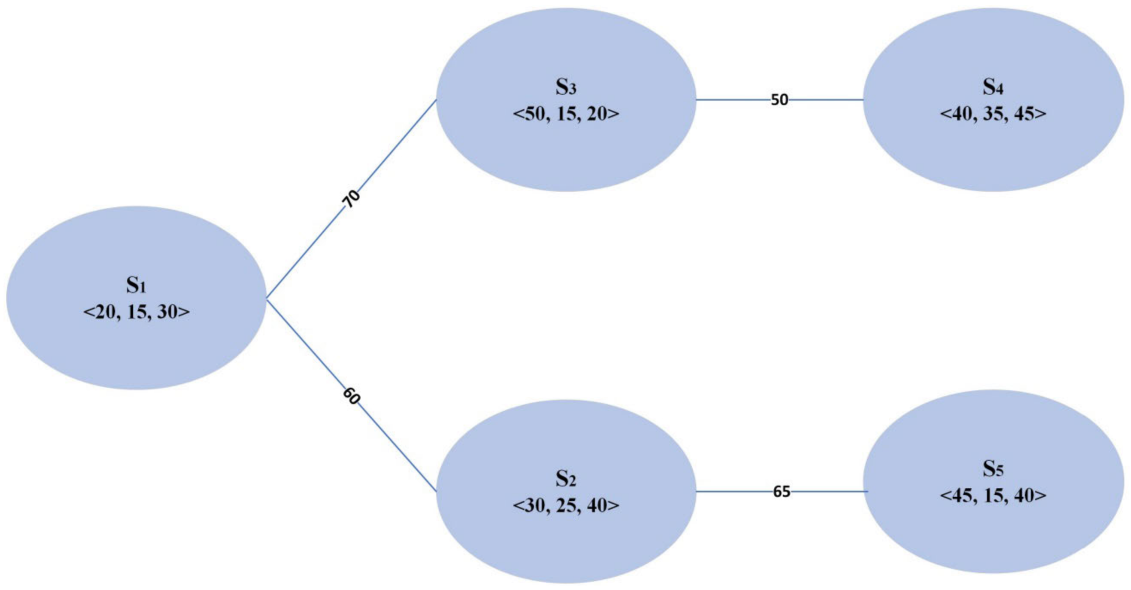

FIGURE 2. Example of microservice-based SaaS with five microservices.

$$\sum_{j=1}^{|S^i|} r_j^{mem} \le p_i^{mem} \tag{7}$$

$$\sum_{j=1}^{|S^i|} r_j^{disk} \le p_i^{disk} \tag{8}$$

where S^i is a set of microservices that are deployed on p_i, $|S^i|$ is the total number of microservices in S^i. $\sum_{j=1}^{|S^i|} r_j^{cpu}$ gives the total CPU requirement of the microservices; $\sum_{j=1}^{|S^i|} r_j^{mem}$ is the total main memory requirement of the microservices; and $\sum_{j=1}^{|S^i|} r_j^{disk}$ is the total disk requirement of the microservices. p_i^{cpu}, p_i^{mem} and p_i^{disk} are the remaining CPU, main memory, and storage capacity of p_i.

2) COMMUNICATION CONSTRAINT

Several microservices require data communication. The communication demand between two microservices is specified in the workflows of the microservice-based SaaS, and these communication demands must be fulfilled. Consider a succession of edges in graph G denoted as $L_{ij} =< e_{ij}^1, e_{ij}^2, \ldots, e_{ij}^l >$. This sequence represents a physical communication path between a pair of compute servers where the microservice s_i and the microservice s_j are deployed. Therefore, the communication constraint is expressed in the following manner:

The minimum value of the set $\{b_{ij}^1, b_{ij}^2, \ldots, b_{ij}^l\}$ must be greater than or equal to $B(e_{ij})$, which represents the communication demands between the microservice s_i and the microservice s_j.

E. OBJECTIVE FUNCTIONS

The objective of deploying a microservice-based SaaS, which is represented by a graph $W = (S, D)$, into a data center represented by another graph $G = (V, E)$, is to assign each microservice in S to a compute server in V with the aim of minimizing the energy consumption of compute servers and network devices involved in the deployment solution R. The deployment solution R is denoted as $\{S_1, S_2, \ldots, S_m\}$, where S_i represents the set of microservices deployed to p_i and $1 \le i \le m$. The objective functions can be clearly stated as follows:

$$\min \Delta E^p = \min \left(\sum_{i=1}^{|P'|} \Delta E_i^p \right) \tag{9}$$

$$\min \Delta E^n = \min \left(\sum_{k=1}^{|N'|} \Delta E_k^n \right) \tag{10}$$

where $|P'|$ and $|N'|$ represent the number of compute servers and network devices involved in a deployment solution, respectively. The term ΔE^p denotes the energy increase in the compute servers, defined as $\sum_{i=1}^{|P'|} \Delta E_i^p$. Similarly, ΔE^n refers to the energy increase in the network devices impacted by the deployment solution, calculated as $\sum_{k=1}^{|N'|} \Delta E_k^n$. The network devices affected by the deployment are determined by running a BFS algorithm between each pair of compute servers hosting connected microservices.

IV. HYBRID PSO FOR THE MICROSERVICE-BASED SAAS DEPLOYMENT PROBLEM

PSO is a popular metaheuristic optimization approach that has been widely applied in various fields, including cloud computing [27]. The main objective of PSO is to find the optimal solution by simulating the social behavior of a group of particles moving in a search space. In cloud computing, PSO has been used to solve different optimization problems, such as resource allocation, load balancing, and task scheduling [28], [29]. In addition, PSO has proven its effectiveness in optimizing data center energy consumption, making it a suitable choice for our paper [30], [31]. Using PSO, our aim is to efficiently address the new SaaS deployment problem while minimizing the increase in energy in compute servers and network devices.

In PSO, a set of candidate solutions, represented as a swarm of particles, move through the search space and update their position based on their own best solution and the global best solution found by the swarm. The particle's movement is guided by a velocity vector, which is updated at each iteration based on its previous position and the best global solution found so far. The fitness of a particle is determined by an objective function and is a measurement of the quality of the particle's position. In this paper, a higher fitness value indicates that the solution represented by a particle will incur a smaller increase in energy consumption in the compute servers and network devices in the data center.

PSO was originally designed to solve continuous optimization problems. Therefore, PSO has to be modified before it can be applied to a discrete optimization problem. To apply PSO to the microservice-based SaaS deployment problem, several modifications are necessary. In this paper, we redefine the operators and parameters used in the original PSO. Additionally, the position update strategy and the PSO velocity update strategy are adapted to comply with these adjustments. To enhance the exploitation capacity of the proposed HPSO, we design a local optimizer algorithm and use it to refine the best global solution. In the following subsections, we will address the above-mentioned issues in detail.

A. PARTICLE

In our HPSO, each particle is characterized by two variables: its position and its velocity. The position of a particle represents a potential deployment solution to the microservice-based SaaS problem. The velocity of the particle guides the direction of exploration of the particle. Each particle in the swarm maintains its own personal best position, denoted as *pbest*. This is the best solution the particle has encountered in its search history. In addition, the swarm maintains the global best position *gbest*, which represents the best solution found by any particle in the entire swarm up to the current iteration. In other words, *gbest* is the position that has the best fitness value among all the personal best positions of all the particles. *gbest* is shared among all particles and influences the movement of all particles during the optimization process.

B. POSITION

The position X_i of a particle i is defined as a vector of dimensions n, $X_i = (x_{i1}, x_{i2}, \ldots, x_{ik}, \ldots, x_{in})$, where $1 \leq k \leq n$ and n is the number of microservices in the SaaS. Each element in X_i is an integer corresponding to the compute server in which the microservice is deployed in the solution. For example, $X_i = (15, 10, 25, 45, 15)$ represents a solution to the microservice-based SaaS deployment problem where microservice 1 is deployed on the compute server 15, microservice 2 is deployed on the compute server 10, microservice 3 is deployed on the compute server 25, microservice 4 is deployed on the compute server 45, and microservice 5 is deployed on the compute server 15.

C. VELOCITY

Velocity helps the particles move towards better solutions by providing a possible adjustment to the current solution. In the proposed PSO, the velocity V_i of the particle i is defined as $V_i = (v_{i1}, v_{i2}, \ldots, v_{ik}, \ldots, v_{in})$, where $1 \leq k \leq n$, and n is the number of microservices in SaaS. The value of v_{ik} is 0 or 1. A value of 0 denotes the necessity to relocate the corresponding microservice to a different server, while a value of 1 indicates that no modifications are required.

D. SUBTRACTION ($X_I \ominus X_J$)

The subtraction operator is utilized to determine any discrepancies between the current position X_i of particle i and either the global position *gbest* or the personal best position $pbest_i$. This subtraction aims to identify the optimal dimensions within the best solutions *gbest* and $pbest_i$, guiding the particle towards improved solutions. The outcome of $(X_i \ominus gbest)$ or $(X_i \ominus pbest_i)$ for each dimension is described as follows:

- If the values are the same, the result of $\ominus$ is 1.
- If the values in the same dimension of X_i and $gbestorpbest_i$ are different, the result of $\ominus$ is 0. Through this subtraction, the disparities between the positions ($gbest$ or $pbest_i$) and X_i are examined, focusing on dimensions where the best solutions and X_i share identical values. For example, if $gbest = (1, 2, 3, 6, 9)$ and $X_i = (1, 2, 5, 8, 9)$, then $gbest \ominus X_i = (1, 1, 0, 0, 1)$. This result indicates that the first, second and fifth dimensions have identical values in both *gbest* and X_i, while there are differences in the third and fourth dimensions. These differences may serve as guidance for the particle's movement towards potentially superior solutions.

E. ADDITION ($P_I V_I \oplus P_J V_J$)

The addition operator is defined to update the velocity and produce the new velocity $V_{i,j+1}$. This operator, denoted as $p_i V_i \oplus p_j V_j$, signifies that the compute servers will be influenced by V_i with a probability of p_i and by V_j with a probability of p_j, where $p_i + p_j = 1$. For example, when we have $0.4(1, 1, 0, 1, 0) \oplus 0.6(1, 0, 0, 0, 0)$, the result is $(1, 0.4, 0, 0.6, 0)$. Here, the value 0.4 in the second dimension indicates the probability of 40% that the dimension is equal to 1, while 0.6 in the fourth dimension indicates the probability of 60% that it is equal to 1. A roulette wheel selection technique is used to perform the addition operation based on the assigned probabilities. This involves generating a random number r between 0 and 1. For example, to determine the value of the second dimension, a random value is generated.

Let us say $r = 0.3$. Since $r < 0.4$, the value in the second dimension will be 1.

F. MULTIPLICATION ($X_{I,J} \otimes V_{I,J+1}$)

The multiplication operator is used to update the current position of the particle, denoted as $X_{i,j}$. This operator modifies the current location on the basis of the value of the new velocity. If $V_{i,j+1}$ is 1, no update is made to the value at the corresponding dimension in $X_{i,j}$. Conversely, if $V_{i,j+1}$ is 0, the respective microservice will be relocated to a server from its candidate list.

For example, if the position vector $X_{i,j} = (2, 1, 4, 9, 7)$ and the velocity vector $V_{i,j+1} = (1, 1, 0, 0, 1)$, then $X_{i,j} \otimes V_{i,j+1} = (2, 1, *, *, 7)$. This indicates that microservices associated with the third and fourth dimensions need to be adjusted in their deployment plans. The modification decision is made so that the value of $*$ is randomly chosen to explore various server options that have sufficient resources to host the microservices.

G. PARTICLE EVALUATION

The fitness function descried in in Equation (11) plays a crucial role in evaluating both individual particles and the overall swarm in the HPSO algorithm. Essentially, it acts as a representation of the optimization problem, guiding particles towards better solutions. Through the process of evaluating the fitness of each particle, the HPSO algorithm is able to identify which particles are better and how their positions can be adjusted to converge towards the most optimal solution. Since our optimization goal is to minimize the increase in energy in compute servers and network devices, the fitness function used to evaluate the particle p_i is directly defined as the following:

$$F(p) = w_1 * f_1(p) + w_2 * f_2(p) \tag{11}$$

where w_1 and w_2 represent the weights assigned to each fitness function, reflecting their relative significance in the problem, and $w_1 + w_2 = 1$.

$$f_1(p) = \frac{\Delta E^s_{max} - \Delta E^s(p)}{\Delta E^s_{max}} \tag{12}$$

where p is the particle that needs to be evaluated, ΔE^s_{max} is the highest increase in energy of the compute servers in the possible deployment solutions in the swarm, $\Delta E^s(p)$ is the increase in energy consumption in the compute servers resulting from the deployment solution represented by particle p, $\Delta E^s(p) = \sum_{i=1}^{|P'|} \Delta E^s_i$; where $|P'|$ is the number of compute servers involved in the deployment solution p. The increase in energy in a compute server s_i is defined in Equation (13) below:

$$\Delta E^s_i = (e^{max}_i - e^{min}_i) \times \Delta w^{cpu}_i \times o_i \tag{13}$$

The variables in the equation are defined as follows: e^{min}_i represents the minimal energy consumption of s_i while it is idle, e^{max}_i represents the highest energy consumption of s_i when it is fully used, w^{cpu}_i represents the current workload of s_i, and o_i indicates whether the power of s_i is currently on or off.

$$f_2(p) = \frac{\Delta E^n_{max} - \Delta E^n(p)}{\Delta E^n_{max}} \tag{14}$$

where ΔE^n_{max} represents the maximum energy increase of network devices in the potential deployment solutions within the swarm. On the other hand, $\Delta E^n(p)$ denotes the energy consumption increase in network devices caused by the deployment solution represented by particle p. Specifically, $\Delta E^n(p)$ is calculated as the sum of the energy increases ΔE^n_k for each network device affected by p, where $|N'|$ represents the total number of network devices impacted by p. The energy increase in network device n_k is determined by Equation (15):

$$\Delta E^n_k = (e^{max}_k - e^{min}_k) \times \Delta l_k \times p_k \tag{15}$$

The variables e^{min}_k and e^{max}_k represent the minimum and maximum energy consumption of a network device, respectively. The variable l_k denotes the load factor of n_k, and the variable p_k shows whether the device n_k is turned on or off.

H. PARTICLE UPDATE

In the HPSO, each particle starts by randomly initializing its position and velocity. The position is initialized by deploying each microservice in a SaaS to a compute server randomly selected from those with adequate resources. Meanwhile, the velocity vector is initialized by creating a vector of the same size as the number of microservices in the SaaS. Each dimension in this vector is randomly assigned a binary variable with a value of either 0 or 1. Subsequently, the position and velocity of each particle are iteratively updated according to a set of equations designed to enhance the exploration and exploitation of the search space. The modified equations to update the position and velocity of a particle i are defined in Equations (16) and (17).

$$\begin{aligned} V_i(t+1) &= p_1 V_i(t) \oplus p_2(pbest_i(t) \ominus X_i(t)) \\ &\quad \oplus p_3(gbest(t) \ominus X_i(t)) \qquad (16) \\ X_i(t+1) &= X_i(t) \otimes V_i(t+1) \qquad (17) \end{aligned}$$

where $V_i(t+1)$ and $X_i(t+1)$ denote the new velocity and the new position of the i^{th} particle, respectively. $pbest_i(t)$ is the personal best position of i and $gbest(t)$ is the global best position among the swarm. The first equation updates the velocity of i, while the second equation updates its position.

p_1, p_2, and p_3 denote the inertia weight, cognitive, and social parameters, respectively. At the beginning the values of p_1, p_2, p_3 are generated randomly under the condition that $p_1 > p_2 + p_3$ and $0 \leq p_1, p_2, p_3 \leq 1$. As the number of iterations t increases, the values of p_1, p_2, p_3 are updated based on the following equations:

$$p^t_1 = p_1 * k_1 * (\frac{T-t}{T}) \tag{18}$$

$$p_2^t = p_2 + k_2 * (\frac{T+t}{T}) \tag{19}$$

$$p_3^t = p_3 + k_3 * (\frac{T+t}{T}) \tag{20}$$

where p_1, p_2, and p_3 are the initial probabilities. The t and T represent the current and the maximum number of iterations, respectively. The update rule for p_1 suggests that its new value p_1^t at iteration t is calculated based on the initial value of p_1, and it decreases as t increases. The term $\frac{T-t}{T}$ implies a linear decrease as t goes from 1 to T. The constant k_1 controls the rate of the decrease. A smaller k_1 would result in a faster decrease. k_2 is associated with the update of p_2. The update rule for p_2 suggests that its new value p_2^t at iteration t is calculated based on the initial value of p_2, and increases as t increases. The term $\frac{T+t}{T}$ implies a linear increase as t goes from 1 to T. The constant k_2 controls the rate of this increase. A larger k_2 would result in a faster increase. Similar to k_2, k_3 is associated with the update of p_3. The update rule suggests that its new value p_3^t in iteration t is calculated based on the initial value of p_3, and increases as t increases. The term $\frac{T+t}{T}$ also implies a linear increase as t goes from 1 to T. The constantk_3 controls the rate of this increase. A larger k_3 would result in a faster increase.

The goal of Equations (18), (19), and (20) are to adjust the values of p_1, p_2 and p_3 over a series of iterations to create a balance between exploration and exploitation. The value of p_1 starts with a high probability of encouraging exploration of the solution space during initial iterations, which then decreases as the number of iterations increases. The decreasing nature of p_1 reflects a transition from exploratory behavior to a more exploitative behavior. In contrast, the values of p_2 and p_3 start with lower probabilities but gradually increase with each iteration until $p_3 > p_1 + p_2$ in the final iterations. This design choice promotes the exploitation of promising regions that have been identified.

I. DESCRIPTION OF THE HPSO ALGORITHM FOR MICROSERVICE-BASED SAAS

The HPSO algorithm starts by initializing the swarm of particles with random positions and velocities in the search spaces (line 1). The algorithm then assesses the fitness of each particle using the fitness function denoted in Equation (11) and initializes the personal best for each of the particles and the global best (lines 2-8). The HPSO then continuously updates the velocity and position of each particle and updates the best global solution at the end of each iteration until the termination condition is satisfied (lines 10-17). During iterations, if there is any violation in a new particle, the HPSO repairs it using Algorithm (3) [6]. In (lines 14-15), the algorithm finds the global best particle in the swarm in the current iteration. Then, HPSO uses a local optimization algorithm to improve the new global best particle. After local optimization, the optimized particle is evaluated and compared with the current *gbest*. If the optimized particle is better than *gbest* in terms of energy savings, *gbest* is replaced by the optimized particle (line 16). The algorithm terminates and returns *gbest* when a predefined termination condition is met. Algorithm 1 is the description of the HPSO algorithm.

Algorithm 1 Hybrid Particle Swarm Optimization

Input: A microservice-based SaaS W and a data center G
Output: A deployment plan of W in G

```
1:  Randomly initialize the particles in swarm $S$ with their
    positions and velocities
2:  for each particle $p$ in the swarm $swr$ do
3:      Calculate its fitness value using Equation (11)
4:  end for
5:  for each particle $p$ in the swarm $swr$ do
6:      Initialize its personal best position $pbest$
7:      Calculate its fitness value
8:  end for
9:  Find the global best position $gbest$ and calculate its
    fitness value
10: while termination condition is not satisfied do
11:     for each particle $p$ in the swarm $swr$ do
12:         Update its velocity and position based on Equa-
            tions(16) and (17), respectively
13:         Repair violations in its position, if there is any as
            in Algorithm (3)
14:         Evaluate its fitness
15:         Update $p$ personal best position $pbest$ and fitness
            if it has a better fitness than its current
            personal best
16:     end for
17:     Find best particle in the swarm $swr$
18:     Apply the Local Optimization Algorithm (2) to
        optimize the best particle
19:     Update the global best position $gbest$ and fitness,
        if needed
20: end while
21: Return $gbest$
```

J. LOCAL OPTIMIZER

In this HPSO, we use a local optimizer to enhance the exploitation capability of the PSO. The input of this local optimizer is the current global best solution, *gbest*, and the output of this local optimizer is an improved solution to the SaaS deployment problem. This local optimizer systematically examines whether the increase in energy consumption incurred by the SaaS deployment could be further reduced by shifting a microservice from the current compute server to an alternative compute server that is used by the current solution to the SaaS deployment problem (lines 1-8). If there are multiple options, it chooses the option that will lead to the most energy decrease (lines 9-10). Algorithm 2 is the description of the local optimization algorithm.

Algorithm 2 Local Optimization

Input: Global best solution *gbest*

Output: Optimized global best solution *gbest*

```
1:  for each compute server $v_i$ used by $gbest$ do
2:      for each microservice $s_i$ assigned to $v_i$ in $gbest$ do
3:          Create an empty list, $L$
4:          for each other compute server $v_j$ used by $gbest$ do
5:              if $v_j$ has sufficient resources to accommodate $s_i$ then
6:                  Calculate energy increase if $s_i$ is moved to $v_j$
7:                  if the energy increase is less than 0 then
8:                      Add $v_j$, together with the energy increase, to $L$
9:                  end if
10:             end if
11:         end for
12:         Find the compute server $v_i$ in $L$ such that the least energy increase is the least
13:         Improve $gbest$ by moving $s_i$ from $v_i$ to $v_k$
14:     end for
15: end for
```

TABLE 3. Compute servers characteristics.

Server Type	c^{cpu} (MIPS)	c^{disk} (GB)	c^{mem} (GB)	e^{max} (W)	e^{min} (W)
1	15	30	10	300	210
2	20	30	10	350	245
3	25	40	10	400	280
4	30	45	10	450	329
5	35	50	10	500	350
6	40	60	10	550	385
7	45	70	10	600	420
8	50	80	10	650	455
9	55	90	10	700	490
10	60	95	10	750	525

V. EXPERIMENT AND DISCUSSION

To evaluate the performance of the HPSO, we implemented the HPSO and compared its performance with that of the GA proposed in [6], as the GA is the only available benchmark approach. The GA begins by generating a population of individuals, where each individual represents a potential deployment solution. This population evolves using a Probability-Based Crossover operator and mutation operator to improve the solutions over successive generations. The comparison between the GA and the HPSO was carried out with randomly generated test problems, as there were no benchmarks available for the new microservice-based SaaS deployment problem. In the following, we will introduce the generation of test problems, explain the conduction of the experiments, and discuss the experimental results.

A. TEST PROBLEMS

Evaluation of the proposed approaches takes place in a simulated environment. Two distinct sets of experiments,

Algorithm 3 Repair Algorithm

Input: An infeasible particle p, a data center $G = (V, E)$

Output: Particle p after repair

```
1:  Get the position $pos$ in particle $p$
2:  for each compute server $x_i$ represented by a dimension in the position $pos$ do
3:      if resource constraints violation is detected in $x_i$ then
4:          Get the set of microservices $MC$ deployed in to $x_i$
5:          for each microservice $m_i$ in $MC$ do
6:              Relocate $m_i$ to another compute server $x_j$ with sufficient resources
7:              $x_j^{cpu} = x_j^{cpu} - m_i^{cpu}$
8:              $x_j^{mem} = x_j^{mem} - m_i^{mem}$
9:              $x_j^{disk} = x_j^{disk} - m_i^{disk}$
10:             if violation is resolved in $x_i$ then
11:                 Break
12:             end if
13:         end for
14:     end if
15: end for
16: for each connected microservices $m_i$ and $m_j$ in $pos$ which are deployed into $x_i$ and $x_j$ do
17:     if communication constraint violation is detected between $m_i$ and $m_j$ then
18:         Find another shortest path between $x_i$ and $x_j$ using BFS algorithm
19:     end if
20: end for
21: Return particle $p$ after repair its position
```

TABLE 4. Switch characteristics.

Switch Type	Number of Switch	e^{max} (w)	e^{min} (w)
Core	64	100	75
Aggregate	128	70	52.5
Edge	128	50	37.5

TABLE 5. Parameter settings.

Parameter	Value
Swarm Size	150
Maximum Generation	200
Decrease Rate (k_1)	0.99
Increase Rate (k_2)	0.20
Increase Rate (k_3)	0.30
Initial Probability (p_1)	0.70
Initial Probability (p_2)	0.20
Initial Probability (p_3)	0.10

Experiment 1 and Experiment 2, are used to evaluate the performance of the HPSO in terms of energy savings, execution time, and scalability.

- Experiment 1 aims to evaluate the proposed HPSO in terms of energy savings, execution time, and scalability across varying numbers of microservices. In this set of

TABLE 6. Experimental results for the HPSO and the GA for the first set of test problems.

Test Problem	Algorithm	Energy Increase in Servers (w)			Energy Increase in Network Devices (w)			Total Energy Increase (w)
		Best	Worst	Average	Best	Worst	Average	
1	GA HPSO	394.43 390.49	430.62 414.18	406.681 394.610	1.23 0.20	2.23 1.51	1.912 0.902	408.594 395.512
2	GA HPSO	948.33 922.80	991.35 969.95	967.871 941.670	9.84 7.82	12.25 11.39	11.304 9.947	979.175 951.617
3	GA HPSO	1486.42 1467.65	1557.58 1508.47	1517.575 1484.270	23.37 18.37	26.37 25.99	25.064 23.407	1542.639 1507.677
4	GA HPSO	1898.62 1895.42	2000.20 1964.09	1956.202 1941.709	37.39 37.09	42.73 41.98	40.660 39.508	1996.861 1981.2017
5	GA HPSO	2412.02 2362.98	2500.75 2445.40	2460.155 2393.183	68.64 65.80	74.25 72.28	72.033 69.457	2532.188 2462.640
6	GA HPSO	2923.48 2817.60	3043.92 2936.28	2990.422 2877.302	103.77 100.65	109.16 109.72	107.430 106.508	3097.852 2983.810
7	GA HPSO	3654.94 3536.03	3805.76 3666.16	3716.976 3616.483	137.15 129.92	141.36 142.23	139.629 137.804	3856.605 3754.287
8	GA HPSO	4094.40 3951.42	4254.08 4124.24	4179.579 4047.179	186.34 174.71	190.87 190.76	189.090 185.808	4368.668 4232.987
9	GA HPSO	4863.75 4740.73	5049.39 4855.39	4967.928 4803.075	232.71 225.26	238.33 237.82	236.278 234.192	5204.206 5037.267
10	GA HPSO	5062.92 4933.63	5203.60 5100.96	5136.266 5016.348	288.95 284.96	297.51 298.19	293.792 293.076	5430.059 5309.424

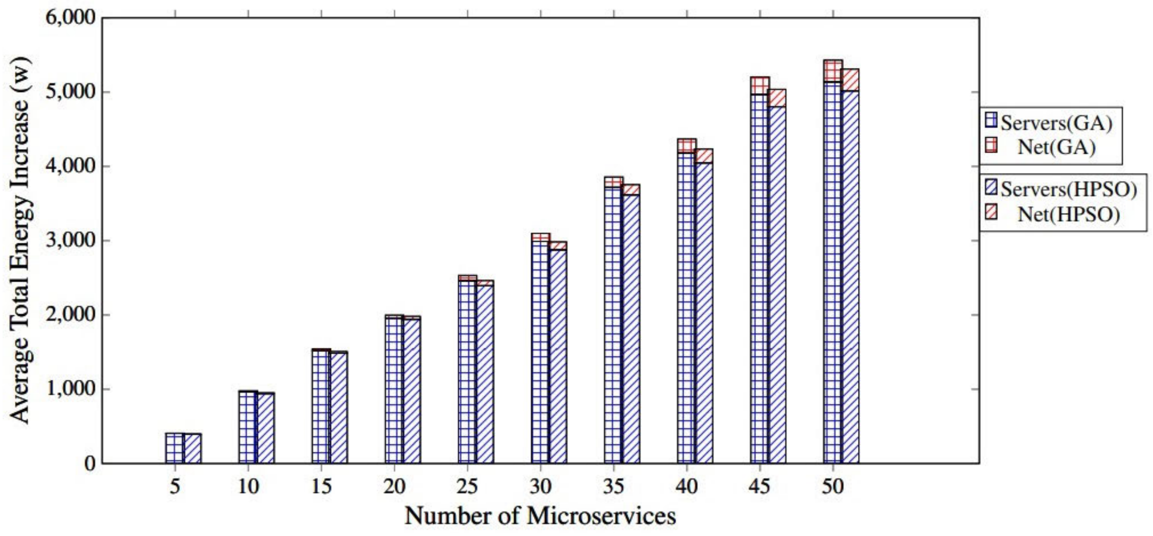

FIGURE 3. Average energy increase while the number of microservices grows for the GA and the HPSO.

TABLE 7. T-test results of the solutions found by the HPSO and the GA in the first set of test problems.

Test Problem	$p-value$
1	1.79E-07
2	1.27E-12
3	1.02E-09
4	0.01
5	1.74E-12
6	7.51E-15
7	4.67E-11
8	1.31E-14
9	2.27E-17
10	1.92E-12

experiments, 10 different test problems are used, each comprising a varying number of microservices. The range of microservices varies from 5 to 50, increasing by increments of 5 microservices. The number of compute servers is set at 500 for all the test problems.

- Experiment 2 aims to evaluate the proposed HPSO in terms of energy savings, execution time, and scalability across varying numbers of compute servers. In this set of experiments, 10 different test problems are used, each of which contains a different number of compute servers. The number of compute servers ranges from 100 to 1000, increasing by increments of 100 servers, while the number of microservices remains fixed at 25 for all the test problems.

The topology of the communication network in the data center used in the experiment was a fat tree with 1 Gbps connection capacity with 16 pods. The characteristics of the compute servers and the characteristics of the network devices are shown in Table 3 and Table 4, respectively.

SaaS workflows were created using a Java program. The number of microservices was given. The microservice

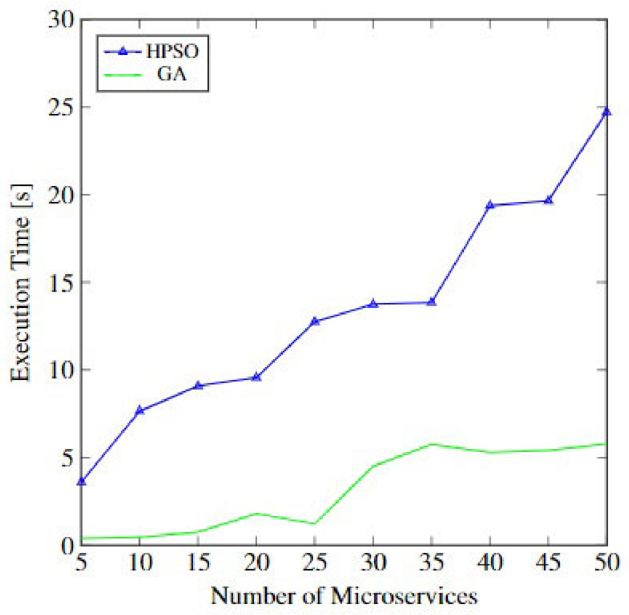

FIGURE 4. **Average execution time of the HPSO and the GA for the first set of test problems.**

dependencies were created at random. The microservice's CPU requirement was a randomly generated number between 15 and 30. The microservice's disk space and RAM requirements were also a random number between 15 and 30 and between 1.0 and 3.0, respectively. Each dependency between a pair of microservices had a randomly generated bandwidth between 30 and 100 Mbps.

B. ALGORITHM IMPLEMENTATION AND EXPERIMENTAL ENVIRONMENT

Java was used to implement the HPSO in CloudSim-3.3.3 [32]. CloudSim is a comprehensive framework and toolkit designed to model and simulate cloud computing infrastructures and services. It offers a flexible simulation environment that enables academics and developers to create and evaluate different algorithms, strategies, and architectures for cloud computing. The experiments were carried out on a desktop with an Intel(R) Core(TM) i7-8700 CPU and 16.0 GB RAM. Since both algorithms are stochastic, each experiment was reproduced 30 times. The HPSO parameters are listed in Table 5.

C. EXPERIMENT 1 RESULTS AND DISCUSSION

We conducted a comprehensive comparison between the HPSO and the GA with a focus on their performance in minimizing the total increase in energy consumption through Experiment 1. The primary objective of these experiments was to examine how the number of microservices affects the energy savings, execution time, and scalability of the HPSO.

1) ENERGY SAVINGS

Table 6 shows the experimental results for Experiment 1. The experimental results comprise the increase in energy consumption within servers and network devices, as well as the total energy increase observed for the HPSO and the GA.

Figure 3 presents a visual representation of the increase in total energy consumption in all test problems. Upon a thorough investigation of the performance of the examined algorithms, a noteworthy observation emerges. In general, for all test problems, the HPSO consistently produced more energy-efficient deployment solutions, with total energy reductions ranging from 0.78% to 3.68% compared to the GA.

We conducted a two-sample paired t-test analysis for each of the 10 test problems to determine the confidence level of the experimental results of the HPSO and the GA. To account for the random nature of the HPSO and the GA, each experiment was replicated 30 times. The null hypothesis for the t-test is that there is no significant difference between the mean values of the solutions found by the HPSO and the mean values of the solutions found by the GA in terms of total energy increase. The t-test was carried out with a confidence level of 95% and a significance level α of 0.05. The p values for the 10 t-test are shown in Table 7. The table shows that the p values are significantly lower than 0.05, indicating strong evidence against the null hypothesis.

2) EXECUTION TIME

The relationship between the average execution time of the HPSO and the number of microservices in SaaS is presented in Figure 4. Figure 4 shows that HPSO begins from an execution time of approximately 3.59 s for the test problem with five microservices, and its execution time gradually increases as the number of microservices increases, ultimately reaching approximately 24.71 s when the number of microservices is 50. While the GA demonstrates a shorter overall execution time, it is important to note that the HPSO involves additional computational steps, contributing to a longer runtime. However, the justification for this extended duration is the significant improvement in energy efficiency. For example, in test problem 6, the additional 9.2 seconds consumed by the HPSO deployment result in a further reduction in energy of 3. 68% compared to GA.

D. EXPERIMENT 2 RESULTS AND DISCUSSION

We also conducted a comprehensive comparison between the HPSO and the GA with a focus on their performance in minimizing the total increase in energy consumption through Experiment 2. The primary objective of these experiments was to examine how the number of compute servers affects the energy savings, execution time, and scalability of the HPSO.

1) ENERGY SAVINGS

Table 8 shows the experimental results for the second set of test problems. The experimental results include the increase in energy consumption within servers and network devices, along with the total energy increase observed for the HPSO and the GA.

TABLE 8. Experimental results for the HPSO and the GA for the second set of test problems.

Test Problem	Algorithm	Energy Increase in Servers (w)			Energy Increase in Network Devices (w)			Total Energy Increase (w)
		Best	Worst	Average	Best	Worst	Average	
1	GA	2193.98	2259.32	2219.561	39.54	51.44	46.122	2265.683
	HPSO	2151.63	2255.82	2204.471	34.12	51.33	44.671	2249.142
2	GA	2393.88	2552.43	2482.321	51.14	65.26	58.851	2541.172
	HPSO	2339.42	2435.79	2397.682	41.24	62.66	52.218	2449.901
3	GA	2405.05	2511.02	2452.264	65.32	70.40	68.707	2520.970
	HPSO	2337.45	2438.21	2390.231	62.65	69.09	66.040	2456.272
4	GA	2413.49	2523.90	2473.461	61.87	73.01	68.498	2541.959
	HPSO	2357.16	2468.15	2400.730	57.67	70.97	65.042	2465.772
5	GA	2412.02	2500.75	2460.155	68.64	74.25	72.033	2532.188
	HPSO	2362.98	2445.40	2393.183	65.80	72.28	69.457	2462.640
6	GA	2409.65	2530.15	2460.696	68.98	74.91	72.944	2533.640
	HPSO	2354.64	2431.89	2401.458	65.37	73.49	69.424	2470.882
7	GA	2412.76	2515.06	2460.866	69.11	75.67	73.447	2534.313
	HPSO	2368.98	2436.52	2402.487	65.47	74.05	70.123	2472.609
8	GA	2412.87	2532.35	2473.046	67.43	76.75	72.386	2545.431
	HPSO	2351.52	2436.01	2393.485	61.29	72.96	68.523	2462.008
9	GA	2409.35	2505.05	2450.550	73.09	76.73	75.007	2525.556
	HPSO	2326.03	2423.70	2384.880	68.11	73.75	71.381	2456.261
10	GA	2407.92	2500.10	2457.649	73.07	76.94	75.379	2533.028
	HPSO	2346.09	2454.13	2395.050	68.59	75.01	72.461	2467.512

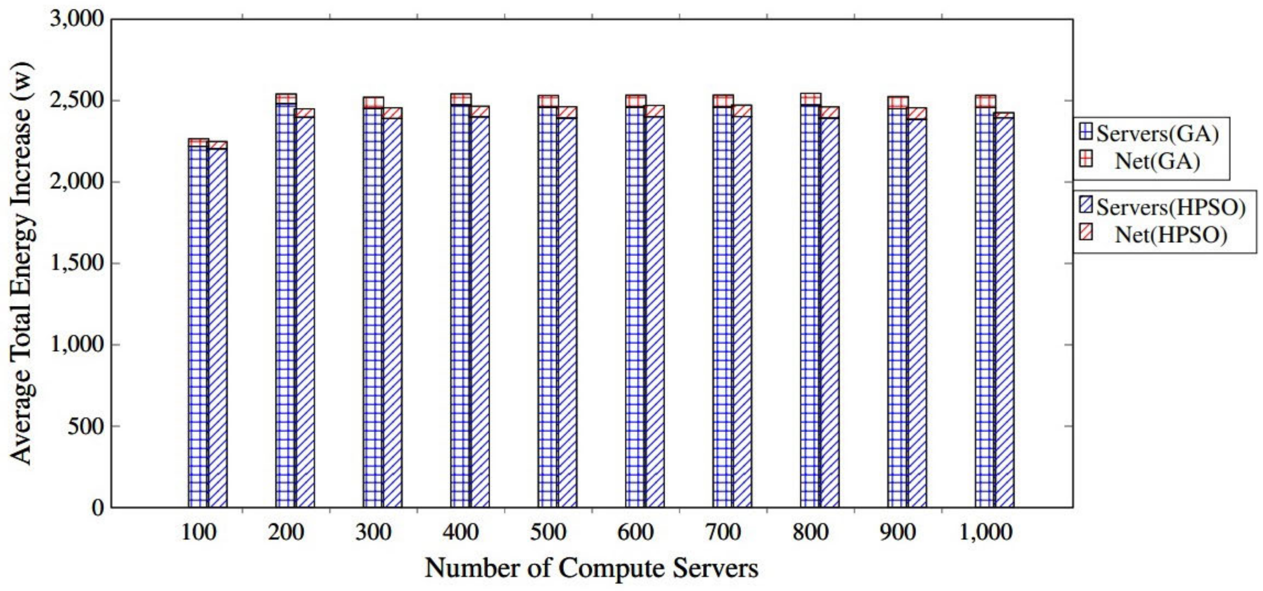

FIGURE 5. Average energy increase for the deployment plans generated by the two algorithms while the number of compute servers grows.

Figure 5 visually demonstrates the level of enhancement achieved by the HPSO in varying numbers of compute servers. HPSO-generated deployment plans can further reduce the increase in energy from 0.73% to 3.59% for test problems compared to GA-generated deployment plans.

In order to demonstrate the confidence level of the experimental results of HPSO and GA, we applied a paired two-sample t-test for means for all the 10 test problems. To account for the random nature of the HPSO and the GA, each experiment was repeated 30 times. The null hypothesis for the t test is that there is no statistically significant difference between the mean energy savings offered by the HPSO and the GA in relation to the total increase in energy consumption. The t-test was carried out with a confidence level of 95% and a significance level α of 0.05. By analyzing Table 9, it can be seen that for each test problem, the p values are significantly lower than 0.05, indicating strong evidence against the null hypothesis. Therefore, the HPSO generates significantly better results than the original GA.

2) EXECUTION TIME

Figure 6 shows that the execution time of HPSO generally increases as the number of compute servers increases. For a smaller number of compute servers (100 - 400), the

TABLE 9. T-test results of the solutions found by the HPSO and the GA in the second set of test problems.

Test Problem	$p - value$
1	0.01
2	3.05E-14
3	1.33E-09
4	2.09E-12
5	1.74E-12
6	1.05E-13
7	1.24E-12
8	6.51E-15
9	6.36E-14
10	8.98E-13

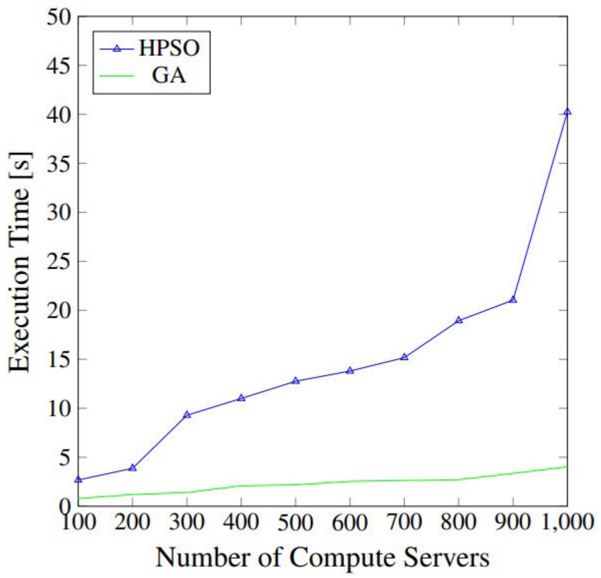

FIGURE 6. Average execution time of the HPSO and the GA for the second set of test problems.

execution time remains relatively low, ranging from about 2.68 to 11.00 seconds. However, it escalates to approximately 40.26 seconds for the test problem with 1000 compute servers. Although the GA has a lower overall execution time, the extra time consumed by the HPSO results in an approximate 3.59% improvement in energy savings.

E. SCALABILITY EVALUATION

Scalability refers to the ability of an algorithm to handle increasing problem size effectively without compromising performance or stability. In this microservice-based SaaS problem, there are two parameters that characterize the size of the problem. One is the number of microservices in the SaaS; another is the number of compute servers in the data center. Thus, when evaluating the scalability of the HPSO, we need to look at how the computation time of the HPSO algorithm implementation increases as the number of microservices increases and the number of compute servers increases.

Figures 4 and 6 show the relationship between execution time and the number of microservices and the number of compute servers, respectively. It can be seen in Figure 4 that the execution time of the HPSO implementation increases linearly as the number of microservices increases. It can be seen in Figure 6 that the execution time of the HPSO implementation increases almost linearly as the number of compute servers increases. These have demonstrated that the HPSO is scalable.

VI. CONCLUSION AND FUTURE WORK

In this paper, a scalable Hybrid Particle Swarm Optimization (HPSO) was proposed for the microservice-based SaaS deployment problem. To evaluate the performance of the HPSO, we implemented it and compared it with a GA. The simulation results demonstrated that the HPSO achieved an additional reduction of 3. 68% in total energy increase in a data center compared to GA, and it did so more efficiently in terms of execution time.

In this paper, we focus on minimizing the increase in energy consumption incurred by the deployment of a microservice-based SaaS in the data center. In the future, we will extend the HPSO to take into account security, privacy, quality of service, and performance.

REFERENCES

[1] R. Buyya, J. Broberg, and A. Goscinski, *Cloud Computing: Principles and Paradigms*, vol. 87. Hoboken, NJ, USA: Wiley, 2010.

[2] C. T. Joseph and K. Chandrasekaran, "Straddling the crevasse: A review of microservice software architecture foundations and recent advancements," *Software: Pract. Exper.*, vol. 49, no. 10, pp. 1448–1484, Oct. 2019.

[3] P. Mell and T. Grance, "The NIST definition of cloud computing," Nat. Inst. Sci. Technol., Special Publication 800.2011, 2011, p. 145.

[4] F. Abdessamia, W.-Z. Zhang, and Y.-C. Tian, "Energy-efficiency virtual machine placement based on binary gravitational search algorithm," *Cluster Comput.*, vol. 23, no. 3, pp. 1577–1588, Sep. 2020.

[5] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 732–794, 1st Quart., 2016.

[6] A. Alzahrani and M. Tang, "A microservice-based SaaS deployment in a data center considering computational server and network energy consumption," in *Proc. IEEE 16th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2023, pp. 505–515.

[7] Z. I. M. Yusoh and M. Tang, "A penalty-based genetic algorithm for the composite SaaS placement problem in the cloud," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 2010, pp. 1–8.

[8] Z. I. M. Yusoh and M. Tang, "A cooperative coevolutionary algorithm for the composite SaaS placement problem in the cloud," in *Proc. Int. Conf. Neural Inf. Process.* Berlin, Germany: Springer, 2010, pp. 618–625.

[9] M. Tang and Z. I. M. Yusoh, "A parallel cooperative co-evolutionary genetic algorithm for the composite SaaS placement problem in cloud computing," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Berlin, Germany: Springer, 2012, pp. 225–234.

[10] Z. W. Ni, X. F. Pan, and Z. J. Wu, "An ant colony optimization for the composite SaaS placement problem in the cloud," *Appl. Mech. Mater.*, vols. 130–134, pp. 3062–3067, Oct. 2011.

[11] H. Mezni, M. Sellami, and J. Kouki, "Security-aware SaaS placement using swarm intelligence," *J. Softw., Evol. Process*, vol. 30, no. 8, p. e1932, Aug. 2018.

[12] Z. Mann, "Secure software placement and configuration," *Future Gener. Comput. Syst.*, vol. 110, pp. 243–253, Sep. 2020.

[13] Z. Liu, L. Wan, J. Guo, F. Huang, X. Feng, L. Wang, and J. Ma, "PPRU: A privacy-preserving reputation updating scheme for cloud-assisted vehicular networks," *IEEE Trans. Veh. Technol.*, early access, Dec. 8, 2023, doi: 10.1109/TVT.2023.3340723.

[14] M. A. Hajji and H. Mezni, "A composite particle swarm optimization approach for the composite SaaS placement in cloud environment," *Soft Comput.*, vol. 22, no. 12, pp. 4025–4045, Jun. 2018.

[15] B. Qian, F. Meng, and D. Chu, "A cost-driven multi-objective optimization algorithm for SaaS applications placement," in *Proc. IEEE Int. Conf. Smart City/SocialCom/SustainCom (SmartCity)*, Dec. 2015, pp. 1086–1091.

[16] W. Chainbi and E. Sassi, "A multiswarm for composite SaaS placement optimization based on PSO," *Softw., Pract. Exper.*, vol. 48, no. 10, pp. 1847–1864, Oct. 2018.

[17] S. Bhardwaj and B. Sahoo, "A particle swarm optimization approach for cost effective SaaS placement on cloud," in *Proc. Int. Conf. Comput., Commun. Autom.*, May 2015, pp. 686–690.

[18] L. C. Ochei, A. Petrovski, and J. M. Bass, "Optimal deployment of components of cloud-hosted application for guaranteeing multitenancy isolation," *J. Cloud Comput.*, vol. 8, no. 1, pp. 1–38, Dec. 2019.

[19] W. Abbes, Z. Kechaou, A. Hussain, A. M. Qahtani, O. Almutiry, H. Dhahri, and A. M. Alimi, "An enhanced binary particle swarm optimization (E-BPSO) algorithm for service placement in hybrid cloud platforms," *Neural Comput. Appl.*, vol. 35, no. 2, pp. 1343–1361, Jan. 2023.

[20] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 8, pp. 1954–1969, Aug. 2020.

[21] L. Bao, C. Wu, X. Bu, N. Ren, and M. Shen, "Performance modeling and workflow scheduling of microservice-based applications in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 9, pp. 2114–2129, Sep. 2019.

[22] X. He, Z. Tu, M. Wagner, X. Xu, and Z. Wang, "Online deployment algorithms for microservice systems with complex dependencies," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1746–1763, Jun. 2023.

[23] K.-C. Huang and B.-J. Shen, "Service deployment strategies for efficient execution of composite SaaS applications on cloud platform," *J. Syst. Softw.*, vol. 107, pp. 127–141, Sep. 2015.

[24] K. Fu, W. Zhang, Q. Chen, D. Zeng, X. Peng, W. Zheng, and M. Guo, "QoS-aware and resource efficient microservice deployment in cloud-edge continuum," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2021, pp. 932–941.

[25] P. Souza, C. Kayser, L. Roges, and T. Ferreto, "Thea—A QoS, privacy, and power-aware algorithm for placing applications on federated edges," in *Proc. 31st Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Mar. 2023, pp. 136–143.

[26] O. Popoola and B. Pranggono, "On energy consumption of switch-centric data center networks," *J. Supercomput.*, vol. 74, no. 1, pp. 334–369, Jan. 2018.

[27] E. H. Houssein, A. G. Gad, K. Hussain, and P. N. Suganthan, "Major advances in particle swarm optimization: Theory, analysis, and application," *Swarm Evol. Comput.*, vol. 63, Jun. 2021, Art. no. 100868.

[28] X. Huang, C. Li, H. Chen, and D. An, "Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies," *Cluster Comput.*, vol. 23, no. 2, pp. 1137–1147, Jun. 2020.

[29] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends," *Swarm Evol. Comput.*, vol. 62, Apr. 2021, Art. no. 100841.

[30] A. Ibrahim, M. Noshy, H. A. Ali, and M. Badawy, "PAPSO: A power-aware VM placement technique based on particle swarm optimization," *IEEE Access*, vol. 8, pp. 81747–81764, 2020.

[31] V. Dinesh Reddy, G. R. Gangadharan, and G. S. V. R. K. Rao, "Energy-aware virtual machine allocation and selection in cloud data centers," *Soft Comput.*, vol. 23, no. 6, pp. 1917–1932, Mar. 2019.

[32] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *Proc. Int. Conf. High Perform. Comput. Simul.*, Jun. 2009, pp. 1–11.

A. ALZAHRANI received the B.S. degree in information technology from King Saud University, Riyadh, Saudi Arabia, and the M.S. degree in computer science from Imam Mohammed Ibn Saud University, Riyadh. She is currently pursuing the Ph.D. degree in computer science with Queensland University of Technology, Brisbane, QLD, Australia. She is a Lecturer with the College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj, Saudi Arabia. Her current research interests include cloud computing, microservice deployment, SaaS deployment, computational intelligence, and its applications in optimization and deployment.

M. TANG received the B.E. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, the M.E. degree in computer science from Chongqing University, Chongqing, China, and the Ph.D. degree in computer systems engineering from Edith Cowan University, Perth, WA, Australia.

He is currently a Faculty Member with the School of Computer Science, Faculty of Science, Queensland University of Technology, Brisbane, QLD, Australia. He has published over 100 papers in prestigious journals and international conference proceedings, including IEEE Transactions on Intelligent Transportation Systems, IEEE Transactions on Industrial Electronics, IEEE Transactions on System, Man and Cybernetics—Part B, IEEE Transactions on Cybernetics, IEEE Transactions on Services Computing, and IEEE Transactions on Cloud Computing. His current research interests include evolutionary computation and nature-inspired computation and their applications in optimization, planning, placement, and scheduling.