SAKARYA UNIVERSITY JOURNAL OF COMPUTER AND INFORMATION SCIENCES
http://saucis.sakarya.edu.tr/
e-ISSN: 2636-8129
Publisher: Sakarya University
Vol. 7, No. 2, 2024, 173-186
DOI: 10.35377/saucis...1452049
SAUCIS
SAKARYA UNIVERSITY

RESEARCH ARTICLE

Optimization Planning Techniques with Meta-Heuristic Algorithms in IoT: Performance and QoS Evaluation

Murat Koca[1], İsa Avcı[2]

[1]Department of Computer Engineering, Faculty of Engineering, Van Yüzüncü Yıl University, Van, Türkiye
[2]Department of Computer Engineering, Faculty of Engineering, Karabük University, Karabük, Türkiye

Corresponding author:
Murat Koca, Department of Computer Engineering, Faculty of Engineering, Van Yüzüncü Yıl University, Van, Türkiye
muratkoca@yyu.edu.tr

Article History:
Received: 13.03.2024
Accepted: 28.06.2024
Published Online: 23.08.2024

ABSTRACT
Big data analysis used by Internet of Things (IoT) objects is one of the most difficult issues to deal with today due to the data increase rate. Container technology is one of the many technologies available to address this problem. Because of its adaptability, portability, and scalability, it is particularly useful in IoT micro-services. The most promising lightweight virtualization method for providing cloud services has emerged owing to the variety of workloads and cloud resources. The scheduler component is critical in cloud container services for optimizing performance and lowering costs. Even though containers have gained enormous traction in cloud computing, very few thorough publications address container scheduling strategies. This work organizes its most innovative contribution around optimization scheduling techniques, which are based on three meta-heuristic algorithms. These algorithms include the particle swarm algorithm, the genetic algorithm, and the ant colony algorithm. We examine the main advantages, drawbacks, and significant difficulties of the existing approaches based on performance indicators. In addition, we made a fair comparison of the employed algorithms by evaluating their performance through Quality of Service (QoS) while each algorithm proposed a contribution. Finally, it reveals a plethora of potential future research areas for maximizing the use of emergent container technology.

Keywords: IoT micro-services, Container method, Optimization algorithms, Scheduling methods, Meta-heuristic algorithms

1. Introduction

We envision the Internet of Things (IoT), the Internet of Everything or the Industrial Internet, as a network of globally interconnected instruments and devices [1]. We expect an increase in IoT devices in the coming years, which will impact supply chain partners' data access and supply chain operation. Gartner 2014 forecasted that the Internet of Things (IoT) will expand to 26 billion units by 2020, up from 0.9 billion in 2009. Similarly, Gartner 2021 projects that FinFET will generate semiconductor device revenue of $138.6 billion in 2025, up from $87.6 billion in 2020, from production lines and warehouses to retail delivery and store shelves [2]. The demand for IoT-connected devices, machine learning (ML) applications, audio or video streaming services, and cloud storage has increased. Therefore, as microservices gain popularity, we anticipate further expansion of cloud services [3]. Cloud computing is based on virtualization technology, which enables the sharing of resources (CPUs, memory, and networks) to execute distinct programs [4].

The Docker container is a standard software entity that encapsulates code and all its dependencies so that an application can operate swiftly and reliably in any computing environment. Containers allow developers to deploy applications in isolated environments, making them ideal for microservice architecture and modern applications[5]. There are several characteristics enjoyed by containers that enable them to displace traditional Virtual Machines (VMs), and among these characteristics are the common host operating system, fast launch, and the possibility of transfer, expansion, and rapid deployment [6], [7] Containers give great flexibility to programs to create an independent runtime on the platform, and by attaching all necessary dependencies, such as instructions, the software runs and manages the system [8].

Due to the variety of duties and available cloud resources, container scheduling has emerged as a crucial aspect of the cost-effective operation of modern cloud applications [9]. Figure 1 depicts a variety of cutting-edge scheduling algorithms devised by scientists that can yield various results.

This work is licensed under Creative Commons Attribution-NonCommercial 4.0 International License

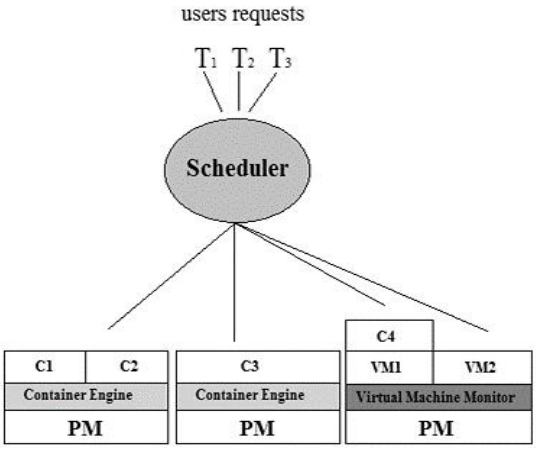

Figure 1. Sample of a Figure Caption

Other considerations include scalability, responsiveness, energy usage, cost and load balancing, resource availability, and resource efficiency, among others. Only a few studies have examined resource management in systems based on containers, although many have examined the scheduling of virtual machines[10], [11]. For large-scale container scheduling problems, no polynomial complexity algorithm exists because the problem is NP-hard. Consequently, most reported algorithms use heuristic techniques to achieve local solutions to the task [12]. Low-complexity, time-saving heuristic algorithms are typically used to generate a workable schedule. Machine learning is a hot topic regarding container scheduling that has had great success in many applications across a wide range of disciplines [13].

Machine learning algorithms rely on big data for their success, and our study's focus on meta-heuristics, a prominent class of population-based optimization algorithms, draws inspiration from the natural development of intellectual activities and actions in our environment. Two essential characteristics of these algorithms are selection and adaptation to the environment [14]. Meta-heuristics solve optimization problems in many fields.

According to Attaoui et al., the ability of mobile network function virtualization (NFV) to decouple network functions from hardware and host services on commodity hardware makes it crucial for mobile network operators. This enhances service deployment and management, improves flexibility, and reduces costs. However, the optimal placement of Virtualized Network Functions (VNFs) poses significant technical challenges, influencing network performance, reliability, and operating costs. This study explores optimization techniques, such as heuristic, meta-heuristic, and machine learning algorithms, to address VNF placement problems, focusing on both VNFs and Container Network Functions (CNFs) in edge/fog computing environments [15].

Shubha Brata Nath et al. talk about how important fog computing is for meeting the latency needs of Internet of Things (IoT) devices. They describe some problems and suggest a framework, PTC, that aims to improve response times by assigning micro-services to fog devices more efficiently. The use of Bayesian Optimization and containerization for service isolation and migration is highlighted, with experimental results demonstrating improved performance over baseline methods, providing a promising approach to enhancing fog computing architectures[16].

Hamza Mohammed Ridha Al-Khafaji proposed widespread device connectivity via the Internet of Things (IoT) and the associated challenges related to energy and cost constraints. It introduces an improved seagull optimization algorithm (ISOA) to enhance the quality of service (QoS) in IoT networks by effectively managing traffic and packet transmission. He has demonstrated that the proposed method significantly enhances QoS by outperforming previous approaches in accuracy and efficiency [17].

Satyanarayana P. et al. describe the Mobile Ad Hoc Network (MANET), a highly mobile and decentralized wireless network, and its integration with the Internet of Things (IoT) to form a novel MANET-IoT system aimed at reducing network implementation costs and enhancing user mobility. It emphasizes the need for new routing protocols and improved security measures, proposing a security protocol that utilizes an enhanced chaotic map and three advanced optimization algorithms. Performance evaluations demonstrate the superior efficiency of the proposed approach in various metrics, particularly the ABRR-CHIO algorithm, which outperforms other techniques significantly in convergence evaluations [18].

Docker is an open-source container platform for developing, shipping, and running applications. A container is a packaging method that combines our application's necessary configurations and dependencies. We run applications in isolated boxes called containers, and by isolating these boxes, we can run different applications on multiple independent containers. Docker creates containers to run and store applications and uses virtualization. Although Docker and virtual machines use isolated virtual environments for software development, they have different structures. The most important feature distinguishing Docker containers from virtual machines is that Docker containers are lighter, faster, and more resource-efficient. A virtual machine, Docker, isolates applications using container structures on a single operating system rather than creating a separate

virtual operating system for each application. This not only reduces the size of the application but also brings a significant performance increase. A piece of hardware can have multiple containers. Containers, unlike virtual machines, are virtualized at the application level. As a result, the host computer shares the kernel and virtualizes the operating system. This reduces resource usage and provides rapid and easily configurable virtual environments.

In this work includes expanded reviews and comparisons for three classical, modified, and hybrid meta-heuristic algorithms. The study included a narrative of the natural spirit and the behavior of the algorithms, additionally the mathematical model and the adaptation of container scheduling. Conversely, the comparisons concentrated on Quality of Service (QoS) metrics that optimization processes enhance. Despite its importance, this type of research remains relatively isolated. Unlike our research, which focuses on meta-heuristic algorithms, there is only a single review in the literature that provides a brief overview of all scheduling techniques. We organize the remaining sections of this essay as follows: Section 2 discusses container scheduling using optimization methods and common performance measures. Section 3 reviews and compares optimization scheduling meta-heuristics, presenting the author's perspective. Section 4 discusses the challenges and potential avenues of research and explains possible solutions. The final section presents the results and outlines the future work.

2. Management Containers by Optimization Algorithms and Performance Measurements

This work compiles research findings published in international magazines, conferences, and organizations such as IEEE, ACM, Elsevier, and Springer between January 2017 and January 2024, including the International Conference on Machine Learning. To gather papers, we searched for several container scheduling-related keywords. The section revolves around two locations: the first, a scheduling container for optimization algorithms, and the second, performance metrics.

2.1. Motivation

Since IoT microservices typically grow to meet user needs and be highly available, redundancy is frequently required. When we need additional processing power, we scale the service. When a microservice resides within a container, it scales by replicating it. Most services have substantial resource requirements and must adhere to strict performance criteria[19]. Lack of resource management might lead to rising expenses, subpar service, and energy waste [20]. Due to high service standards, finding an ideal location is necessary to ensure each service has enough resources without wasting any. To maximize consumption, cloud customers want to plan resources [20]. This project will place each microservice in its own container before scheduling it on a virtual machine. To reduce the number of Virtual Machines (VMs) and their overall cost, a container placement problem comparable to the Virtual Machine Placement (VMP) must be addressed.

2.2. Scheduling Container Problem Model Discerption

We formulate the Container Placement (CP) issue as follows: Given a set of containers, each with a different set of resource needs, such as CPU-intensive or memory-intensive containers, try to place all containers on VMs while using the fewest number of VMs as possible. Assume that within the specified period, between the hours of t1 and t2, a set of containers arrives at the data center. When using online container allocation, the overall goal is to assign container slots to existing VMs or newly created VMs, and then assign those VMs to physical machines (PMs) to keep the total energy consumption of all PMs at its lowest possible level. The resource entities share the following characteristics (containers, virtual machines, and physical machines). CPU and memory are the two types of resources available to each entity. This study considers a data center with heterogeneous VMs of different types and homogeneous PMs, all of which share the same CPU and memory capacity. Each type of virtual machine comes pre-configured with a tuple of resources (e.g., a small VM [825 MHz, 800 MB]). Each VM has overheads in addition to its resource capacity. Virtual machine overheads represent the resources that a hypervisor uses up. A tuple of resources represents each type of VM's overhead.

Each virtual machine can run a specific operating system. A cloud provider will also specify the supported operating system categories. In the case of containers, we contemplate a one-to-one correspondence between applications and containers, as opposed to the approach of Piraghaj [21] , which employs three categories of containers for all applications. The domain of required resources for containers is defined as a number between one and the capacity of PMs. Consequently, the container is defined in a much more realistic and general manner [22]. This task necessitates the use of containers that contain virtual machines running the same operating system. There are four decisions to make. Our model includes the following procedures for the online container allocation problem: Figure 2 illustrates the selection and construction of virtual machines, represented by blue lines, and virtual machines, represented by red lines.

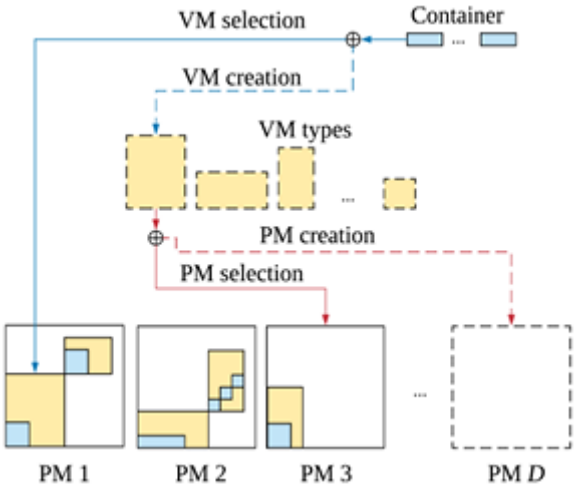

Figure 2. Container Placement Procedure

The VM selection process selects existing VMs from which to allocate containers. Another option involves creating virtual machines using cloud provider-defined virtual machine types and allocating containers to the appropriate virtual machines. The operating system requirements of the first container determine the operating system type of the virtual machine. The PM selection process chooses pre-existing PMs to assign the newly created VMs. In that it produces a kind of PM and allows the VM to use it, PM creation functions similarly to VM creation.

2.3. Scheduling Containers by Optimization Algorithms

Meta-heuristic algorithms are high-level, general optimization strategies used to solve complex problems that do not have a known solution method. We call them "meta" because they offer a higher level of abstraction than specific algorithms. Such algorithms have two important characteristics: selection of the most suitable candidates and adaptation to the environment. Meta-heuristics frequently solve optimization problems in a variety of disciplines [13].

Meta-heuristic algorithms have recently become a go-to method for tackling various industries' most challenging optimization problems. Genetic algorithms (GA) and swarm intelligence techniques like ant colony optimization, particle swarm optimization (PSO), and whale optimization are examples of meta-heuristic algorithms. Here, the meta-heuristic utilized to derive the solution has been used to categorize scheduling strategies.

2.4. Performance Metrics

We can use several performance metrics to evaluate the effectiveness of optimization algorithms:

- **Convergence**: It measures the rate at which the algorithm approaches the optimal solution.
- **Solution Quality**: It measures the algorithm's proximity to the optimal solution. The value of the objective function typically determines this.
- **Computational Time**: It measures the time the algorithm takes to find the solution.
- **Scalability**: It measures the algorithm's ability to handle larger problems.
- **Robustness**: It measures the algorithm's ability to generate effective solutions regardless of alterations to the problem or exposure to noise.
- **Repeatability**: It gauges how consistently the algorithm produces the same results for a particular problem instance each time it runs.
- **Accuracy**: It measures how closely the algorithm's solution matches the optimal solution. We recommend readers review the domain-specific optimization objectives from the most recent survey[23]. Below is a list of the most frequently used goals in the cost function specification of the container scheduling problem.

2.4.1. Energy

Regarding container deployment, the computing and storage devices hosting the containers, including servers, storage devices, switches, and other infrastructure components, consume energy. Energy consumption is an important consideration in container deployment because it can have a significant impact on the system's operating costs as well as the environment. Factors contributing to energy consumption in container deployment include the number of containers running, the utilization of computing and storage resources, the efficiency of the hardware components, and the power management policies in place. To minimize energy consumption, organizations can employ techniques such as container orchestration, resource utilization

optimization, hardware selection, and power management policies. As a result, energy efficiency serves as a critical metric for reducing carbon emissions and improving environmental sustainability [23].

2.4.2. Cost

Compute, storage, and communication costs all contribute to the overall cost of an application's execution. Compute time refers to the time it takes to run a program on the cluster's available cores. The cost increases when an application runs on a computer for a long time. Communication costs are also about how much telecommunications service providers pay to run the application. The need for more communication between nodes and clusters increases communication costs.

2.4.3. Availability

We refer to the duration of a user's access to an application as its lifetime. Users of cloud computing should have access to applications and services whenever they need them. Accessing applications and services whenever needed is an important consideration for cloud computing users.

2.4.4. Use of resources

This statistic measures the efficiency with which a worker node uses its network bandwidth, memory, and CPUs on a per-worker-node basis. According to this metric, workers are most efficient and cost-effective when used to their full capacity.

2.4.5. Load Balancing

Distributed computing ensures that no computer overburdens other idle computers. This goal affects response time, cost, and throughput. This metric is critical because of the dynamic nature of the workload in container-based applications.

2.4.6. Scalability

Additionally, container scalability is an essential measure of how well a system can handle increased demand, whether it comes from one application or a variety of different ones. These metrics measure the system's ability to adapt to changing workloads by increasing the available resources or deploying more containers on the cluster.

2.4.7. Scalability

It takes time for the application to run from start to finish. Makespan is a good scheduler's primary goal. If you're operating a real-time application that demands minimal latency or severe deadlines, this statistic is essential.

2.4.8. Throughput

The productivity of an application is determined by dividing the number of tasks by the time allotted to them. At the same time, the transfer rate provides an overview of system performance (processor, memory, and network).

2.4.9. Security

The orchestra can protect both data and services against hostile attacks or software defects by leveraging encryption and access control techniques. Online transaction processing, centralized financial services, and productivity tools enhance the performance of this metric. Because of the serious security vulnerabilities they provide, containers require special care.

2.5. The Objective Function for Performance in Algorithms

In the context of optimization or machine learning, the objective function f(x), as shown in equation 1, is a mathematical expression that you aim to maximize or minimize.The form of this function depends on the specific problem you are addressing. Here is a general example:

$$f(x) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad (1)$$

Where:

- y_i is the observed value,
- $\hat{y}_i$ is the predicted value,
- n is the number of observations.

This is a common objective function used in regression problems known as the mean squared error (MSE).

2.6. Statistical Analysis for Algorithms

To evaluate the performance of different algorithms, we typically perform a statistical analysis. The key metrics often include the mean, standard deviation (std), execution time, and significance values (p-values). Here's a step-by-step guide:

1. **Mean**: Equation 2 displays the average performance metric (such as accuracy, error rate) over several runs or datasets.

$$Mean = \frac{1}{n} \sum_{i=1}^{n} x_. \quad (2)$$

2. **Standard Deviation (std):** Equation 3 calculates the variability, or distribution, of the performance measurement.

$$Std = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - Mean)^2} \quad (3)$$

3. **Execution Time:** The time taken by the algorithm to complete its task. This can be measured using time functions in programming languages like Python's time module.

4. **Significant Value (p-value):** Determines if the performance difference between algorithms is statistically significant. Typically, you perform a hypothesis test (e.g., t-test) to compute the p-value.

- **Mean and Std:** Provide insights into the central tendency and variability of the algorithms' performance.
- **Execution Time:** Helps assess the efficiency of the algorithms.
- **P-value:** If the p-value is below a significance threshold (e.g., 0.05), you can conclude that there is a statistically significant difference between the performance of the two algorithms.

3. Meta-heuristic Optimization Techniques for Scheduling

Here, we demonstrate the optimization algorithms used to enhance the container scheduling problem. We also discuss meta-heuristic algorithms in detail.

3.1. Ant Colony Optimization Algorithm

3.1.1. Natural Inspired

Ant Colony Optimization (ACO), "Optimization, Learning, and Natural Algorithms," was proposed by Marco Dorigo in 1992 and is the earliest and most widely used technique [24]. The ability of actual ants to locate sustenance and determine optimal routes inspired the development of the ACO. We use this population-based general search technique to address complex combinatorial optimization problems. The Ant Colony Optimization method draws inspiration from ants' ability to communicate with each other through a specific chemical known as pheromone. Self-organizing agents or acts can communicate with each other via "stigmergy," a term that refers to "indirect communication" between them.

Real ants first travel throughout their colony in search of food. On their journey from colony to food, they leave a unique chemical 'pheromone.' When they return, they also leave behind a small amount of pheromone. Pheromone concentrations on shorter routes are higher because the ants return earlier. Because this path is the shortest, it attracts more traffic after a certain amount of time. There is a special rate at which the phenols dissipate. Consequently, ants eventually erase long trails they don't use. Pheromone trails let ants find the most efficient route between colonies and food. Therefore, the ACO algorithm mimics the behavior of real ants [25], [26].

3.1.2. ACO Algorithm

To make the ant colony algorithm suitable for continuous problems, such as the traveling salesman problem, we must first program it to generate solutions using integers. Next, we initiate the pheromone update step, which entails the deposit and evaporation of pheromones, a crucial aspect of ACO[25]. At first, ants haphazardly begin their search for food. An ant uses a probabilistic equation to choose the next node to visit. Equation 4 gives the likelihood of ant k traveling to node j while ant k is on the node.

Algorithm 1. Initializing the Pheromone

```
Procedure Initializing the pheromone matrix τ for all edges in G
    For each iteration t from 1 to T:
        For each k ant from 1 to m:
            Choose the next node n in ant path k based on pheromone values and inference (e.g., using probability rule).
        Update ant path k with node n
        Update the pheromone values at the edges based on the quality of the tracks found by the ants
Determine the best path found by the ants as a result of the ACO algorithm
end procedure
```

$$P_J = \begin{bmatrix} \frac{\tau_j^{\propto}\mu_j^{\beta}}{\sum j \in N_i \tau_j^{\propto}\mu_j^{\beta}} & for\ j\ \in\ N_i \\ 0 & for\ j\ \in !\ N_i \end{bmatrix} \quad (4)$$

In addition, pheromone update comprises pheromone deposit and evaporation. As an ant moves from one node to the next, its pheromone values fluctuate. The process of pheromone evaporation involves a constant factor reducing the first pheromone value on each arc.

Each ant adds some pheromone to each node it traverses as part of the pheromone deposit. Equation 5 for pheromone evaporation is as follows:

$$\boldsymbol{\tau = (1 - p)\tau} \tag{5}$$

Where ρ is the rate of evaporation. Each ant deposits some pheromone on each node, which is known as a pheromone deposit and is given by Equation 6 below:

$$\boldsymbol{\tau = \tau + \triangle \tau} \tag{6}$$

Where m is the number of ants, Δτ is the amount of pheromone drop on the k node. It is calculated as shown in Equation 7:

$$\boldsymbol{\triangle \tau = Q/z} \tag{7}$$

The present length of the tour by k ant. It is worth mentioning that at the end, create one diminution matrix (1, the number of cites) with different values according to equation 1 divided by the summation of probabilities. The production of this equation is processed by a roulette wheel to select a numeric value between one and the city number. This operation is repeated until the estimated set, which represents by its turn the probable solution.

3.1.3. ACO for Container Adapting

The basic ant colony algorithm organized the containers perfectly, depending on the Docker algorithm; the results revealed a significant 15% improvement in performance compared to the greedy herd. However, the algorithm only considered a few optimization goals, such as maximizing the use of available resources and ensuring an even distribution of work Burvall, Multi-Objective Container Placement The presentation of Ant Colony Optimization (MOCP-ACO) transformed the traditional Ant Colony implementation by incorporating new criteria like network utilization and cost into the decision-making process [25]. Compared to Docker Swarm's spread scheduling method, MOCP-ACO outperformed the latter in all tested cases, regardless of the workload or the number of containers (16–1024) used.

3.2. Genetic Algorithm

3.2.1. Natural Inspired

Natural selection and evolution processes inspire the Genetic Algorithm (GA), a type of optimization algorithm. It is a population-based search algorithm that employs the concept of survival of the fittest to identify the optimal solution [27]. We create new populations by repeatedly applying genetic operators to members of an existing population. The essential components of GA are chromosome representation, selection, crossover, mutation, and fitness function computation. The following is GA's dispute resolution procedure: With a random sample, we start a population of chromosomes (Y). We determine the fitness of Y chromosomes by computing their fitness values. Based on their fitness, we select C1 and C2 from population Y and designate them as C1 and C2, respectively. We subject C1 and C2 to a single-point crossover operator with crossover probability (Cp), resulting in O production. We then subject the produced progeny (O) to a uniform mutation operator with a mutation probability of Mp to generate O′. It is determined where the new progeny O′ will be placed in the new population. To finalize the new population, it will be necessary to repeat the selection, crossover, and mutation processes in the current population.

3.2.2. GA Algorithm

GAs are algorithms that mimic the process of natural selection. Genetic algorithms enhance problem solutions by gradually evolving a population of potential solutions as the problem's complexity increases. Each prospective solution consists of a set of chromosomes that can be modified and altered in many ways; traditionally, solutions are represented in binary as strings of 0s and 1s. We could summarize the GA by following these steps [28]:

First: Create n by m array of population,

Check the fittest chromosome: Evaluate each chromosome to calculate fitness**.**

Generate a new population:

Selection: Select two chromosomes from the population by following the selection procedure.

Crossover: Using the two chromosomes you've chosen, perform a crossover.

Mutation: On the chromosomes obtained, perform mutations.

Replace: Substitute the new population for the current population.

Test: Check to see if the end condition is met. If this is the case, stop. If not, move to Step 2 and return to the best solution for the current population.

3.2.3. The GA Approach for Container Scheduling

Guerrero et al. proposed the first container scheduling method based on the NSGA-II standard. The National Science Foundation created the multi-objective optimization method known as NSGA-II [27]. When creating their formulation, the authors took four optimization goals into account: resource utilization, failure rate, load balancing, and network communication overhead. The suggested method outperformed the researchers by 40–60% in the bulk of the intended objectives, according to the Kubernetes scheduling algorithm. Zhang et al. suggested an enhanced evolutionary algorithm based on a non-linear energy model to minimize energy usage [29]. e have created two new mutation operators to more effectively find the best answer. Conversely, Tan et al. designed the suggested plan with a single goal in mind: energy optimization. Tan et al. have presented an energy-saving two-level hybrid algorithm. Genetic programming is a population-based evolutionary computer strategy, much like genetic algorithms [22].

In this method, VMs are initially assigned to containers before the VMs are assigned to actual computers (PMs). Compared to heuristic-only-based techniques like first-fit, best-fit, and so forth, the hybrid approach greatly lowered energy usage, according to the experimental data. Imdoukh et al. proposed the NSGA-III-based Many-Objective Genetic Algorithm. The NSGA-III, an improved version of the NSGA-I, can handle more objective functions. They evaluated the system's performance and efficiency from various angles, including load balancing, scalability, power consumption, and resource availability and use. An examination in comparison to an ACO-based scheduling methodology proved the strategy's efficiency [30]. Tan et al. built on the work they had already done to solve the two-level container allocation problem using a hyper-heuristic method based on cooperative Coevolution Genetic Programming (CCGP) [22]. This method concurrently created allocation rules for both layers (containers-virtual machines and VMs-physical machines), lowering total energy usage.

Compared to the state-of-the-art algorithms, the results of the experiments revealed a significant reduction in energy consumption. The proposal of Dhumal and Janakiram C-Balancer proposes a scheduling framework that utilizes container runtime metrics to optimize container placement in a cluster environment [31]. Specifically, the suggested technique employs a GA-based optimizer to select the most reliable and effective container for node placement by regularly collecting container runtime information. The approach's fundamental idea is to rebalance containers by distributing them over several nodes using runtime data gathered during the migration process. The Swarm Cluster experiment demonstrated that the C-usefulness Balancers improved performance in terms of resource consumption and throughput [32].

3.3. Particle Swarm Optimization Algorithm

3.3.1. Natural Inspired

We devised the PSO algorithm to address the social behavior of animals such as schooling fish, swarming invertebrates, and avian migration. Researchers use this method to systematically search for the global optimum of numerous arbitrary problems. Kennedy and Everhart presented it for the first time [33]. The initial plans for this concept included simulating the graceful and unpredictable movements of a flock of birds to discover the patterns that govern their ability to fly synchronously and abruptly change directions while regrouping into an optimal formation, all with the goal of discovering new patterns. The correct term is stochastic, population-based evolutionary computing. Individuals' ability to maintain cognitive consistency depends on social influence and social learning. Therefore, the exchange of ideas and interactions between individuals may facilitate the resolution of problems. The particle swarm simulates this social structure. Li, Huang, and Wu claim that this method seeds the multidimensional search space of an objective function with a random set of particles, each assigned a specific position and velocity [34]. The objective function measures each particle as a potential solution to the problem. We characterize a collection of particles as a "swarm". During their voyage through multidimensional space, these particles utilize two essential reasoning skills: their memory of their optimal position and their knowledge of the optimal positions around the globe or in their immediate vicinity. In a minimization problem, "best" refers to the particle's (xi) position with the minimum objective value, min fxi. Members of a swarm exchange information about advantageous positions and adjust their position and velocity accordingly. Thus, a set of design variables (xi) and the corresponding velocities (vi) represent an optimization solution for each particle.

3.3.2. PSO Algorithm

First, a brief overview of the standard PSO algorithm is provided. Assume there are m particles in the colony that are being searched for in the D-dimensional space. D-dimensional vectors are used to represent the Ith particle in the search space, which indicates that the particle is located as shown in Equation 8:

$$\boldsymbol{XI} = (\boldsymbol{X_{i1}}, \boldsymbol{X_{i2}}, \boldsymbol{X_{i3}}, \ldots, \boldsymbol{X_{id}})\ (\boldsymbol{I} = \boldsymbol{1}, \boldsymbol{2}, \ldots, \boldsymbol{m}) \quad (8)$$

Positioning each particle could be a solution to this problem. We can determine particle fitness by comparing the particle's position to a designated objective function [34] , which is "better" when the fitness level is higher. The ith particle's "flying" velocity is represented by a D-dimensional vector, as shown in Equations 9, 10, and 11:

$$\boldsymbol{i} = (\boldsymbol{vi1}, \boldsymbol{vi2}, \ldots, \boldsymbol{viD}) \quad (9)$$

99

$$\boldsymbol{I} = \boldsymbol{1}, \boldsymbol{2}, \ldots, \boldsymbol{m}) \quad (10)$$

$$\boldsymbol{Pi} = (\boldsymbol{pi1}, \boldsymbol{pi2}, \ldots, \boldsymbol{pgD}) \quad (11)$$

The particle's optimal position ( ) and the colony's optimal position ( ) are indicated. The PSO algorithm can be implemented using the following Equations 12, 13:

$$\boldsymbol{Xi}(\boldsymbol{k} + \boldsymbol{1}) = \boldsymbol{Xi}(\boldsymbol{k}) + \boldsymbol{Vi}(\boldsymbol{k} + \boldsymbol{1}) \quad (12)$$

$$\boldsymbol{Vi}(\boldsymbol{k} + \boldsymbol{1}) = \boldsymbol{wVi}(\boldsymbol{k}) + \boldsymbol{c1r1\ Pi} - \boldsymbol{xi}(\boldsymbol{k}) + \boldsymbol{c2r2\ Pg} - \boldsymbol{xi}(\boldsymbol{k}) \quad (13)$$

The inertia coefficient w, nonnegative constant learning rates c1 and c2, random generation of r1, and the inertia coefficient we are all constants in the interval [0, 1]. Pbest and fitness gbests are used to determine whether iterations should be terminated when they have reached their maximum generation or a predetermined value[35].

3.3.3. PSO for Container Scheduling

Li et al. improved resource utilization and load balancing with a PSO algorithm [34]. To get around the local minimum, the authors implemented a simulated annealing algorithm with PSO. In a series of tests, the suggested implementation of Docker Swarm outscored the spread technique by 20. As a scheduling technique based on PSO, Guo and Yao suggested employing the neighborhood division idea to fine-tune the PSO algorithm's parameters for better-quality solutions [35]. o enhance system performance, the algorithm took into account load balancing as well as reaction time. In the suggested approach, non-dependent containers were put on nearby hosts to minimize their connection [8], [36]. Compared to the existing and PSO algorithms, the results showed a 20–25 percent improvement. The increased power consumption associated with the developed solution was one of its drawbacks. With the help of a two-stage multi-type particle swarm optimization (TMPSO) algorithm, we were able to solve the container consolidation problem [21]. To increase the quality of the results, the TMPSO algorithm employed a combination of heuristic and greedy strategies. When comparing conventional and binary PSO algorithms, the experimental findings revealed a significant reduction in energy consumption. Adhikari and Srirama proposed, to reduce energy consumption and computing time, a multi-objective particle swarm method for scheduling containers for Internet of Things jobs [37].

Additionally, the suggested method accounted for lowering CO2 emissions and calculating node temperatures. For big data applications, Liu et al. proposed the Kubernetes container scheduling algorithm, K-PSO, based on particle swarm optimization [8]. The authors added a dynamic inertia weight and learning factor to the standard PSO, allowing faster convergence with higher-quality solutions. The authors added a new predicate scheduling process and priority scheduling process to the Kubernetes native scheduler to leverage the improved PSO.

The experiments revealed a 20 percent increase in resource utilization compared to Kubernetes' default scheduling strategy. Conversely, we can expand the suggested approach to include multi-criteria optimization objectives. To decide on container-based micro-service deployment in an edge computing paradigm, Fan et al. presented an edge computing-based scheduling method based on PSO [38]. The algorithm knows latency, reliability, and load balancing (LRLBAS). We demonstrated the efficacy and efficiency of the treatment by comparing trial results with PSO versions. Edge computing has demonstrated the effectiveness of the LRLBAS algorithm for micro-service scheduling. In the future, the authors intend to incorporate additional objectives into their proposal.

4. Discussion

The issue is allocating container slots to existing or new VMs and assigning those VMs to PMs while simultaneously reducing the power consumption of all PMs. The data center contains a variety of heterogeneous virtual machines and homogeneous PMs. Each virtual machine includes a set of preconfigured resources and overheads. The first container's operating system requirements determine the virtual machine's operating system type. The situation necessitates four decisions, including VM selection, VM creation, PM selection, and PM generation. The section also discusses two optimization techniques for container adaptation: ACO and GA. MOCP-ACO outperformed the Docker Swarm deployment scheduling method in every scenario tested, while the NSGA-II-based method outperformed the Kubernetes scheduling algorithm by 40–60% for many

targets. In comparison to heuristic-only techniques, the enhanced evolutionary algorithm and the two-level hybrid algorithm substantially reduced energy consumption. Compared to the most recent algorithms, the multi-objective genetic algorithm based on NSGA-III and the collaborative genetic programming approach based on the over-exploration method yielded significant results. Table 1 provides a comparison of selected works from the literature.

Table 1. Comparison of Selected Works from the Literature

Compared Algorithms	Goal								Tool	Algorithm	Limitations
	Energy	Availability	Utilizing	Balance	Explanation	Cost	Make-span	Connectivity			
ACO			*	*					Docker	-First ACO approach -Effective use of resources	-No container migration –Required parameter tuning
MOCP-ACO						*		*	Docker	-Modified ACO methodology -Be aware of network latency	-Is slow and disregards energy
MOACO	*	*	*					*	-	-Reduce network cost –No -Use resources efficiently	-Container consolidation -No energy education
NSGA-II	*	*	*					*	Kubernetes	-First GA with multiple objectives based on NSGA-II	-No container migration -No energy reduction
Improve-GA	*								-	Non-linear energy model, with an emphasis on energy savings	-Requires parameters tuning
2-level-hybrid-GA	*								-	-Genetic programming	-Single objective

										-Two-level optimizatio n	-No cooperative evolution
MOGAS	*	*	*	*	*				Docker	-Take into account five goals based on NSGA-III	-Slow, need parallelizatio n
Improve-PSO			*	*					Docker	-First PSO model; enhanced performanc e	-Not fault tolerant; needs settings to be adjusted
PSO-based- Scheduling				*			*	*	CloudSim	-Immediate -No migration of the container -Uses more RAM	-Improved efficiency
TMPSO	*		*						-	-Objective PSO -Fast	-No movement of containers
PSO-for-IoT tasks	*		*				*		CloudSim	-Think about CO2 emissions -Attention to temperature	-Needs settings to be tuned
Vhatkar and Bhole		*		*				*	-	-Hybrid whale-Lion model, incorporating availability into account	-Inadequate testing in a real-world environment; parameter-twisting
0C-balance-GA			*				*		Docker	-GA-based strategy - Migration of containers	-Needs settings to be tuned
PSO-based-container Scheduling			*						Kubernetes	-PSO method modification - Fast convergence	-Only a single goal

LRLBAS		*		*			*		'	-Edge computing paradigm -Objective PSO	-No energy savings, no real-world testing

5. Conclusion

With the proliferation of IoT services, the use of containers in the cloud computing community to provide cloud services has increased dramatically in recent years. Container scheduling has grown to become a significant component of cloud computing architecture to ensure proper management of cloud resources during runtime. We conduct a comprehensive analysis in this paper to examine the landscape of container scheduling methods. To start, we divided scheduling strategies into four groups depending on the optimization method that was used to create the schedule. Following that, we considered optimization goals to determine how well the created schedules performed. Then, based on performance indicators, we categorized and described current strategies for each category to understand their benefits and drawbacks. A new generation of resource management and scheduling systems will be required because of container technology, according to present trends and probable future research paths. New developments in technology, whether fuzzy or cloud computing, provide opportunities and new horizons for researchers in reducing the power and increasing the security and communications of these settings and micro-services.

References

[1] I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," *Bus. Horiz.*, vol. 58, no. 4, pp. 431–440, Jul. 2015, doi: 10.1016/J.BUSHOR.2015.03.008.

[2] W. W. W. Gartner, " Gartner says the Internet of Things will transform the data center."

[3] Y. Alahmad, T. Daradkeh, and A. Agarwal, "Availability-Aware Container Scheduler for Application Services in Cloud," *2018 IEEE 37th Int. Perform. Comput. Commun. Conf. IPCCC 2018*, Jul. 2018, doi: 10.1109/PCCC.2018.8711295.

[4] M. Alouane and H. El Bakkali, "Virtualization in Cloud Computing: Existing solutions and new approach," *Proc. 2016 Int. Conf. Cloud Comput. Technol. Appl. CloudTech 2016*, pp. 116–123, Feb. 2017, doi: 10.1109/CLOUDTECH.2016.7847687.

[5] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment", Accessed: May 10, 2023. [Online]. Available: http://www.docker.io

[6] X. Li, P. Garraghan, X. Jiang, Z. Wu, and J. Xu, "Holistic Virtual Machine Scheduling in Cloud Datacenters towards Minimizing Total Energy," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1317–1331, Jun. 2018, doi: 10.1109/TPDS.2017.2688445.

[7] M. Lin, J. Xi, W. Bai, and J. Wu, "Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud," *IEEE Access*, vol. 7, pp. 83088–83100, 2019, doi: 10.1109/ACCESS.2019.2924414.

[8] B. Liu, J. Li, W. Lin, W. Bai, P. Li, and Q. Gao, "K-PSO: An improved PSO-based container scheduling algorithm for big data applications," *Int. J. Netw. Manag.*, vol. 31, no. 2, p. e2092, Mar. 2021, doi: 10.1002/NEM.2092.

[9] F. Chen, X. Zhou, and C. Shi, "The container scheduling method based on the min-min in edge computing," *ACM Int. Conf. Proceeding Ser.*, pp. 83–90, May 2019, doi: 10.1145/3335484.3335506.

[10] A. R. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Futur. Gener. Comput. Syst.*, vol. 91, pp. 407–415, Feb. 2019, doi: 10.1016/J.FUTURE.2018.09.014.

[11] P. J. Maenhaut, B. Volckaert, V. Ongenae, and F. De Turck, "Resource Management in a Containerized Cloud: Status and Challenges," *J. Netw. Syst. Manag. 2019 282*, vol. 28, no. 2, pp. 197–246, Nov. 2019, doi: 10.1007/S10922-019-09504-0.

[12] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali, "Approximation and online algorithms for multidimensional bin packing: A survey," *Comput. Sci. Rev.*, vol. 24, pp. 63–79, May 2017, doi: 10.1016/J.COSREV.2016.12.001.

[13] K. Hussain, M. N. Mohd Salleh, S. Cheng, and Y. Shi, "Metaheuristic research: a comprehensive survey," *Artif. Intell. Rev. 2018 524*, vol. 52, no. 4, pp. 2191–2233, Jan. 2018, doi: 10.1007/S10462-017-9605-Z.

[14] S. Pouyanfar *et al.*, "A Survey on Deep Learning," *ACM Comput. Surv.*, vol. 51, no. 5, Sep. 2018, doi: 10.1145/3234150.

[15] W. Attaoui, E. Sabir, H. Elbiaze, and M. Guizani, "VNF and CNF Placement in 5G: Recent Advances and Future Trends," *IEEE Trans. Netw. Serv. Manag.*, vol. 20, no. 4, pp. 4698–4733, Dec. 2023, doi:

10.1109/TNSM.2023.3264005.
[16] S. B. Nath, S. Chattopadhyay, R. Karmakar, S. K. Addya, S. Chakraborty, and S. K. Ghosh, "PTC: Pick-test-choose to place containerized micro-services in IoT," *Proc. - IEEE Glob. Commun. Conf. GLOBECOM*, 2019, doi: 10.1109/GLOBECOM38437.2019.9013163.
[17] H. M. R. Al-Khafaji, "Improving Quality Indicators of the Cloud-Based IoT Networks Using an Improved Form of Seagull Optimization Algorithm," *Futur. Internet 2022, Vol. 14, Page 281*, vol. 14, no. 10, p. 281, Sep. 2022, doi: 10.3390/FI14100281.
[18] P. Satyanarayana, G. Diwakar, B. V. Subbayamma, N. V. Phani Sai Kumar, M. Arun, and S. Gopalakrishnan, "Comparative analysis of new meta-heuristic-variants for privacy preservation in wireless mobile adhoc networks for IoT applications," *Comput. Commun.*, vol. 198, pp. 262–281, Jan. 2023, doi: 10.1016/J.COMCOM.2022.12.006.
[19] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Multi-objective scheduling of micro-services for optimal service function chains," *IEEE Int. Conf. Commun.*, Jul. 2017, doi: 10.1109/ICC.2017.7996729.
[20] C. Kaewkasi and K. Chuenmuneewong, "Improvement of container scheduling for Docker using Ant Colony Optimization," *2017 9th Int. Conf. Knowl. Smart Technol. Crunching Inf. Everything, KST 2017*, pp. 254–259, Mar. 2017, doi: 10.1109/KST.2017.7886112.
[21] T. Shi, H. Ma, and G. Chen, "Energy-Aware Container Consolidation Based on PSO in Cloud Data Centers," *2018 IEEE Congr. Evol. Comput. CEC 2018 - Proc.*, Sep. 2018, doi: 10.1109/CEC.2018.8477708.
[22] B. Tan, H. Ma, and Y. Mei, "A Hybrid Genetic Programming Hyper-Heuristic Approach for Online Two-level Resource Allocation in Container-based Clouds," *2019 IEEE Congr. Evol. Comput. CEC 2019 - Proc.*, pp. 2681–2688, Jun. 2019, doi: 10.1109/CEC.2019.8790220.
[23] S. S. Gill and R. Buyya, "A Taxonomy and Future Directions for Sustainable Cloud Computing," *ACM Comput. Surv.*, vol. 51, no. 5, Dec. 2018, doi: 10.1145/3241038.
[24] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006, doi: 10.1109/MCI.2006.329691.
[25] B. Burvall, "Improvement of Container Placement Using Multi-Objective Ant Colony Optimization," *DEGREE Proj. Comput. Sci. Eng.*, 2019, Accessed: May 10, 2023. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-249709
[26] J. Yang, X. Shi, M. Marchese, and Y. Liang, "An ant colony optimization method for generalized TSP problem," *Prog. Nat. Sci.*, vol. 18, no. 11, pp. 1417–1422, Nov. 2008, doi: 10.1016/J.PNSC.2008.03.028.
[27] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *J. Grid Comput.*, vol. 16, no. 1, pp. 113–135, Nov. 2017, doi: 10.1007/S10723-017-9419-X/METRICS.
[28] C. Teixeira, J. A. Covas, T. Stützle, and A. Gaspar-Cunha, "Multi-objective ant colony optimization for the twin-screw configuration problem," *https://doi.org/10.1080/0305215X.2011.639370*, vol. 44, no. 3, pp. 351–371, Mar. 2012, doi: 10.1080/0305215X.2011.639370.
[29] R. Zhang, Y. Chen, B. Dong, F. Tian, and Q. Zheng, "A Genetic Algorithm-Based Energy-Efficient Container Placement Strategy in CaaS," *IEEE Access*, vol. 7, pp. 121360–121373, 2019, doi: 10.1109/ACCESS.2019.2937553.
[30] M. Imdoukh, I. Ahmad, and M. Alfailakawi, "Optimizing scheduling decisions of container management tool using many-objective genetic algorithm," *Concurr. Comput. Pract. Exp.*, vol. 32, no. 5, p. e5536, Mar. 2020, doi: 10.1002/CPE.5536.
[31] A. Dhumal and D. Janakiram, "C-Balancer: A System for Container Profiling and Scheduling," Sep. 2020, Accessed: May 10, 2023. [Online]. Available: https://arxiv.org/abs/2009.08912v1
[32] L. Li, J. Chen, and W. Yan, "A particle swarm optimization-based container scheduling algorithm of docker platform," *ACM Int. Conf. Proceeding Ser.*, pp. 12–17, Nov. 2018, doi: 10.1145/3290420.3290432.
[33] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proc. ICNN'95 - Int. Conf. Neural Networks*, vol. 4, pp. 1942–1948, doi: 10.1109/ICNN.1995.488968.
[34] L. J. Li, Z. B. Huang, F. Liu, and Q. H. Wu, "A heuristic particle swarm optimizer for optimization of pin connected structures," *Comput. Struct.*, vol. 85, no. 7–8, pp. 340–349, Apr. 2007, doi: 10.1016/J.COMPSTRUC.2006.11.020.
[35] Y. Guo and W. Yao, "A container scheduling strategy based on neighborhood division in micro service," *IEEE/IFIP Netw. Oper. Manag. Symp. Cogn. Manag. a Cyber World, NOMS 2018*, pp. 1–6, Jul. 2018, doi:

10.1109/NOMS.2018.8406285.
[36] X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, and Q. X. Wang, "Particle swarm optimization-based algorithms for TSP and generalized TSP," *Inf. Process. Lett.*, vol. 103, no. 5, pp. 169–176, Aug. 2007, doi: 10.1016/J.IPL.2007.03.010.
[37] M. Adhikari and S. N. Srirama, "Multi-objective accelerated particle swarm optimization with a container-based scheduling for Internet-of-Things in cloud environment," *J. Netw. Comput. Appl.*, vol. 137, pp. 35–61, Jul. 2019, doi: 10.1016/J.JNCA.2019.04.003.
[38] G. Fan, L. Chen, H. Yu, and W. Qi, "Multi-objective optimization of container-based microservice scheduling in edge computing," *Comput. Sci. Inf.*, vol. 18, no. 1, pp. 23–42, 2021, Accessed: May 12, 2023. [Online]. Available: http://www.doiserbia.nb.rs/Article.aspx?id=1820-02142000041F

Author(s) Contributions
Murat Koca: Algorithm generation, Optimization of Algorithms, writing, review, and editing.
İsa Avcı: Comparison of Algorithms, Writing, review, and editing.

Conflict of Interest Notice
Authors declare that there is no conflict of interest regarding the publication of this paper.

Ethical Approval
It is declared that during the preparation process of this study, scientific and ethical principles were followed, and all the studies benefited from are stated in the bibliography.

Availability of data and material
Not applicable

Plagiarism Statement
This article has been scanned by iThenticate ™.