I.J. Information Engineering and Electronic Business, **2026, 1, 105-125**
Published Online on February 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2026.01.07

Exploring and Implementing Container Scheduling Methods: A Comparative Review and Practical Approach

Kanika Sharma*
Lovely Professional University/School of Computer Applications, Phagwara, 144411, India
E-mail: kanusharma0503@gmail.com
ORCID iD: https://orcid.org/0000-0002-5058-971X
*Corresponding Author

Parul Khurana
Lovely Professional University/School of Computer Applications, Phagwara, 144411, India
E-mail: parul.khurana@lpu.co.in
ORCID iD: https://orcid.org/0000-0002-1690-7972

Received: 16 January, 2025; Revised: 25 April, 2025; Accepted: 10 June, 2025; Published: 08 February, 2026

Abstract: Container-based virtualization has become prominent as lightweight virtualization due to its scalability, resource utilization, and portability, especially in microservices. Container scheduler plays an essential role in Container services to optimize performance to reduce the overall cost by managing load balancing. Although Containers serve a lot of benefits, resource allocation is one of the major concerns associated with Container technology. This paper systematically reviews the distinct scheduling techniques used for Containers in a cloud environment, where existing scheduling techniques and their shortcomings have been discussed in detail. In the review process, the various issues and challenges in the Container technology have been identified, and the same has been discussed in this paper. Based on crucial elements including different performance metrics like CPU utilization, memory utilization, load balancing, and many others, it gives an in-depth comparison of the existing scheduling techniques like Ant Colony Optimization(ACO), Particle Swarm Optimization(PSO), Bee Colony Optimization(BCO), Chicken Swarm Optimization(CSO) and Genetic Algorithm(GA) outlining their benefits and drawbacks. This study also proposes a hybrid framework for secure and efficient Container scheduling. This framework can be implemented in the future to provide better results compared to existing approaches.

Index Terms: Cloud Computing, Containerization, Isolation, Resource allocation, Scheduling, Virtualization, Virtual Machines

1. Introduction

Cloud computing deals with essential services like on-demand access to virtual resources, broad network access, resource pooling, rapid elasticity, and measured services, and it offers a shared pool of configurable resources that can be utilized with less involvement of the resource providers. These resources can be accessed by the user at any time as well from any global location. Virtualization is exploited to render these resources flexible to accommodate on-demand resource provisioning, Cloud providers mix physical servers into several virtual instances of Virtual Machines (VMs). Scheduling is the most important feature of a cloud system owing to its close connection to the effective performance of the cloud. This allows cloud providers to maximize resource consumption by executing several tasks concurrently on the same physical hardware.[1].

Virtual Machine (VM) scheduling refers to the process of distributing and controlling resources like as CPU, memory, and storage among several VMs that are operating on a single physical computer. As a result, all VMs will operate smoothly and with effective resource use. Virtual Machine scheduling is significantly influenced by the hypervisor, a thin layer of software that resides between the VMs and the real hardware. Several scheduling techniques like Best Fit, Round robin scheduling, etc. are used to manage the allocation of physical CPU cores to virtual CPUs (vCPUs) within various VMs. Although Virtual Machines offer advantages like resource isolation, flexibility, and scalability still, there are some drawbacks as well. Each VM runs its operating system, consuming additional resources

like CPU, memory, and storage. Moreover, sharing the physical hardware among multiple VMs introduces potential security risks. This overhead can lead to decreased resource utilization and higher hardware costs [2].

Due to emerging transformations in virtualization, like lightweight application management Containers are gaining more popularity. Container scheduling involves assigning lightweight Containers isolated units of software, to host machines within a cluster or platform. Unlike virtual machines, Containers share the host operating system, making them much more agile and resource-efficient. Application deployment is made simple by Containers as a Service (CaaS), which offers Container-based virtualization. With their increased flexibility as an application management framework and improved resource efficiency as compared to Virtual Machines (VMs), Containers play a significant role in cloud computing to accomplish virtualization in high-performance computing [3]. Therefore, many cloud service providers like Google, Amazon have started to offer container-based services and orchestration tools to manage these services, like Google Kubernetes and Docker Swarm.

Using a task scheduling system, Containerized applications are attempted to be assigned to a suitable node. Based upon receipt of the request, the data center allocates the job to a suitable host, which acquires the resources required to execute the task. Job scheduling is one of the important aspects of load balancing and achieving high performance in a distributed environment. Scheduling algorithms are categorized into two categories: Static scheduling and Dynamic scheduling. Static scheduling is used when prior information is available about the resources required by the job, and we have a node with the same number of resources available. Dynamic scheduling refers to the allocation of resources at run time to enhance resource utilization [4,23].

As a global leader in video streaming, Netflix operates a massively scaled microservice architecture. To satisfy high availability, low delay, and low cost, Netflix developed and deployed Titus, its Container management platform on top of Mesos and later in Kubernetes. New show releases and regional events meant the platform had to handle unpredictable, bursty workloads. To address the scheduling challenges, Netflix implemented priority-based scheduling for the user-facing services, bin packing strategies for maximizing resource utilization in off-peak hours, and affinity and anti-affinity rules to guarantee fault tolerance and keep workload in different zones separate. They also tried out machine learning models to predict traffic spikes to enable proactively adjusting scheduling decisions. This resulted in improved Container density, lower cloud infrastructure costs and faster service delivery. At the same time, this case shows a practical example of using intelligent Container scheduling techniques in a large-scale production environment, complementing the theoretical frameworks discussed in the paper [5].

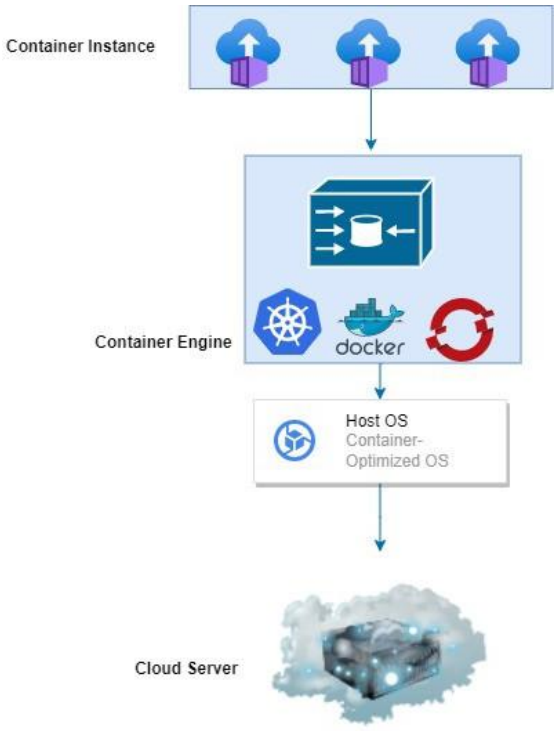

Fig. 1. Container Architecture.

Fig. 1. represents the architecture of Container. It contains Container instances, which are packed with the application that needs to be executed in the cloud environment with its required dependencies and libraries. The second layer in the Container architecture includes the Container engine, like Docker, Kubernetes, OpenShift and many others, which are responsible for providing run time environment to the applications. The third layer contains a host OS. Because all Containers can share the same operating system, they can all use the same kernel. As a result, the boot time is reduced. The last layer includes a cloud server,where finally the Container application is deployed [6].

1.1 Scope of survey

This paper presents a critical survey of Container scheduling techniques rather focusing on the main areas, namely a review of the most popular Container Scheduling techniques concerning ACO, PSO,BSO,CSO, and GA, will be provided, together with an overview of the algorithms and their mathematical formulation. Examining the core of the scheduling algorithms, for instance, bin-packing, load-balancing, and energy-efficient approaches, this work conducts an assessment against key performance metrics such as throughput, latency, resource utilization, and fault tolerance. This not only presents the heterogeneity of resources, scalability challenges, and overcoming network constraints but also new trends in the area of machine learning-based scheduling, multi-cluster, hybrid cloud scheduling, and how they are going to influence modern computing environments. The survey bridges the gaps between theory and practice by discussing various cases, highlighting real-world applications and future research directions in these domains.

1.2 Research Contribution

This article includes a thorough overview of the literature and in-depth analysis of the methods for scheduling Containers that are already in use in the cloud. The major contribution of this paper includes:

- A detailed analysis of classification of Container scheduling techniques has been done.
- A comparative analysis of the existing scheduling techniques for Containers have been done based on their performance metrics like load balancing, fault tolerance, make span and many more.
- Various issues and challenges in the existing scheduling techniques have been outlined.

1.3 Paper Orientation

The rest of this paper is organized as follows. Section 2 describes background and related research. It also explains the related survey of existing scheduling techniques and also describes the scheduling technique classification. Section 3 presents results and analysis of existing scheduling techniques and their mathematical formulation, followed by an analysis of performance metrics. Section 4 discusses the proposed hybrid approach for efficient and secure Container scheduling. Section 5 discusses the novelty of this study.

2. Background Research

Container technology has emerged as a pivotal component in cloud computing, revolutionizing the way applications are deployed, managed, and scaled across distributed infrastructures. Its lightweight, portable, and isolated nature has made Containers a preferred solution over traditional virtual machines (VMs), especially in microservices architectures. The adoption of Containerization has increased due to the efficiency and flexibility it offers in packaging and running applications. This section briefly explains the Container scheduling. This section also discusses the review technique and the electronic repository used for conducting this survey.

2.1. Container Scheduling

Container stores packaged, independently functioning, ready-to-deploy components of an application, including libraries and dependencies needed to run the programs. A piece of software known as a Container engine handles user requests, including command-line arguments, fetches images, and executes the Container from the viewpoint of the end user. It is in charge of containerizing all the dependencies and libraries needed for the application. It offers hardware abstraction to clients. In cloud computing, Containers play a significant role. For high-performance computing environments, Container-based virtualization is preferable [3].

Fig. 2 represents the Container scheduling. When a container is deployed in the cloud environment, resources need to be allocated to the container for its execution. A node is assigned to the Container, which contains all the necessary resources required to execute the Container. Firstly, the node capacity is fetched from the server. if a node is capable of handling the Container, the Container is assigned to the node. If there is no node available to execute the Container, then a search will be performed in the distributed system to search for a node with a matching capacity of node. If a node, the Container is assigned to the node and executed. Otherwise, the Containerized process is terminated. It will wait in the queue until and unless a node with desired resources is allocated to the Container [7, 8].

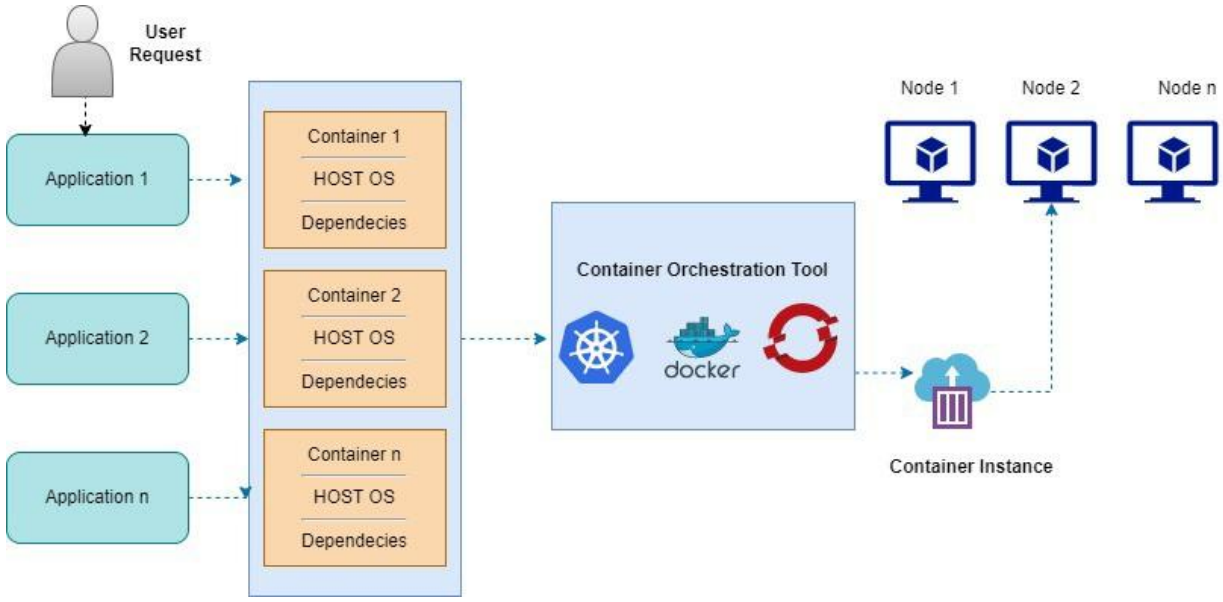

Fig. 2. Scheduling in Containers

2.2. Review Techniques

This section discusses the review techniques used to conduct the systematic literature review. The following principal methodology has been used for selecting the paper for the literature review.

- Inclusion: To conduct this systematic review, papers have been identified based on the keywords and titles. Papers have been reviewed by reading the abstract and introduction, then skimming the body of the paper to get a sense of the overall arguments.
- Exclusion: A Total of 300 papers were identified based on the search. The exclusion of a study from this systematic review is because it does not meet the inclusion criteria. Another reason for exclusion is that the papers were not relevant to the research questions.

2.3. Electronic Repositories

This section presents various keywords used for searching papers from different electronic repositories that are pertinent to the area of Container scheduling. The keywords are identified from the existing state-of-the-art research in Container scheduling.

Table 1. General keywords used for searching the articles

General keywords	**Specific keywords**	**Duration**	**Content type**
Container scheduling	Optimization,	2017- 2025	Journal, Conference, Book Chapter.
	security		
Cloud and Container	Secure		
environment	Scheduling		
Resource utilization	Load balancing		
Container orchestration	Docker		
	Swarm, Kubernetes		

Table 2. Databases used for searching information.

Database	**Link**
ACM Digital library	http://www.acm.org/dl
Google Scholar	https://scholar.google.com/
IEEE eXplore	https://ieeexplore.ieee.org/Xplore/home.jsp
Science Direct	https://www.sciencedirect.com/
Springer	https://www.springer.com/gp
Wiley	https://www.wiley.com/en-in
Docker official site	https://www.docker.com/
Kubernetes official site	https://kubernetes.io/

Table 1. shows various keywords used for searching articles. The various databases from which the journal articles, conference papers, book chapters, and technical reports are retrieved for this study are listed in Table 2. The existing research from 2017 to 2025 has been considered in this survey paper. In the initial phase, a total of 300 relevant papers were retrieved from the mentioned databases. Then, after the critical examination of the abstract, conclusion, and introduction, 42 papers were filtered out for writing this survey.

2.4. Literature Review

The following table represents the work done in the context of the Container scheduling techniques. It describes the existing scheduling techniques for the Containers. The following table lists down the name of scheduling, main contribution, advantages, limitations and performance metrics taken up by the authors.

Table 3. Related literature of existing scheduling technique

Ref.	Year	Technique	Performance metrics	Main focus	Advantages	Limitations
Liu et al. [9]	2018	Multiopt	• CPU Utilization • Memory Management • Bandwidth • TPS	Reducing processing time.	It minimizes The overall processing time for all tasks.	This algorithm does not consider fault tolerance.
Hu Et al. [10]	2018	ECSched	• CPU Utilization • Memory Management • Make span	Better re-Source efficiency.	The proposed model offers the flexibility of handling the concur-rent request on cloud Containers.	It does not consider Container dependencies and resource dynamics.
Kambatla et al. [11]	2020	UBIS	• CPU Utilization • Memory Management • Job throughput • make span	Scheduling in a distributed environment to improve resource utilization.	The proposed technique effectively utilizes the Wasted resources while minimizing the adverse effect on the regularly scheduled jobs.	There is a need to improve memory managemet.
Lin Et al. [12]	2019	Ant Colony Algorithm	• CPU Utilization • Memory Management • Fault tolerance • bandwidth • load balancing	Reducing node selection cost and load balancing.	The proposed technique provides a better optimization of Cluster service, cluster load balancing and minimizing network transmission overhead.	The primary limitation of the proposed technique is the static allocation of the nodes.
Ungureanu et al. [13]	2019	Hybrid shared-state scheduling frame- work	• CPU Utilization • Memory Management • Fault tolerance	Efficient resource utilization.	The proposed scheduler combines the facilities of central-ized and distributed frameworks to ensure an optimal scheduling mechanism for resource utilization.	The proposed technique is not tested in rreal-time environment to validate.
Vhatkar et al. [14]	2019	WR-LA	• CPU Utilization • Memory Management • Fault tolerance	Efficient resource allocation.	The primary benefit of this technique is to achieve resource allocation optimization.	The proposed technique does not consider the computation time of task completion.
Hussein et al. [15]	2019	ACO-BF	• CPU Utilization	To improve resource utilization	The proposed technique focused on placement architecture to maximize the utilization of PMs and VMs	Performance metrics like memory, storage, and data transfer is not taken into account.
Shen et al. [2]	2019	X-Containers	• Security • Isolation	To improve Container security.	The proposed model supports various binary compatibility and multi-processing, thus making it superior to existing techniques.	The proposed model is focused only on x86 64-bit model.
Li et al. [16]	2020	Optimal service schedul- ing technique	• CPU Utilization • Memory Manage-ment	Improving Container service performance.	The primary advantage of the proposed algorithm is to offer efficient ser- vices with low execution time.	Scalability and many others factor for service performance are not include.
Abbes et al. [17]	2020	Novel replica-tion factor modeling approach	• Fault tolerance	It focus on adapting the replication factor to handle failed Containers	The primary advantage of the proposed technique to handle Containers which are not able to get the resource in the cloud.	The proposed technique needs to cover the storage overhead issue. Moreover, the scalability of nodes is not considered.
Liu etal. [18]	2021	K-PSO	• Resource utilization • Scaling	To improve Container scheduling for big data applications.	The proposed algorithm reduces the execution time of scheduling algorithm compared to the existing one, like PSO.	Only developed for Hadoop. Moreover, it is not tested for all components of Hadoop.

Menouer [19]	2020	KCSS	• Make span • power consumption	Optimize the Container scheduling.	The proposed algorithm selects the Node based on multiple criteria to improve the performance of Container scheduling.	There is a need to add more performance evaluation parameters.
Zheng et al. [20]	2021	UVPOC	• Power consumption	Container resource migration- tion	The proposed algorithm improves The global resource utilization of Containers.	More performance evaluation parameters, like fault tolerance, load balancing many others, need to be considered.
Hu etal. [21]	2021	QoS Modal	• Memory Management • CPU Utilization • Disk Utilization • Bandwidth	Improving resource utilization	The proposed technique offers the Optimal allocation of resource scheduling	A few more constraints, like security, need to be added to improve the overall quality of performance.
Acharya etal. [22]	2022	Proposed Scheduling algorithm	• CPU Utilization	Improving CPU utilization	The main goal of this algorithm is to select a machine with minimum CPU utilization so that a Container can be allocated to that machine.	No focus on memory utilization.
Muniswamy etal. [23]	2022	DSTS	• CPU Utilization	Enhance t h e cloud resource workload	The main goal of the proposed algorithm is to improve customer service level agreements. It also achieves dynamic scheduling of jobs based on priority.	Resource dynamics are not added.

2.5. Container Scheduling Techniques

In modern software development and deployment, Containerization has become a crucial technology due to its ability to bind applications and their dependencies into an isolated and portable environment. A key component of Container management is Container scheduling, which promotes effective resource utilization, scalability, high availability, and cost reduction. In this study, various popular scheduling algorithms are explored.

The Containers can be scheduled by using different schedulers like Kubernetes, Docker Swarm. In some cases, it is desirable to modify the Container allocation process during the deployment of a Container when it will be deployed on a real cluster environment. An example would be that the nodes containing SSDs will be allocated for I/O intensive Containers. Nodes with more CPUs are expected to be assigned to computation-intensive Containers. Within a distributed system, Containers using a lot of network interfaces are expected to be assigned to nodes with lots of bandwidth.

In Fig. 3, we classify container scheduling techniques in four broad categories: mathematical modeling techniques, heuristic techniques, meta-heuristic techniques, and machine learning based techniques. The classifications here represent different ways of dealing with container placement, resource allocation, and workload distribution in cloud computing environments. Formal optimization models, such as linear programming, integer programming, and queuing theory, derive optimal or near-optimal scheduling decisions through mathematical modeling techniques. However, these methods are computationally intensive and not as effective as they could be in dynamic or large-scale systems because they are complex and take a long time to execute. On the other hand, heuristic techniques employ simplified rule-based logic to generate acceptable solutions quickly. First fit, Best fit, Round robin, and resource-aware strategies are just some examples of which. Heuristics are fast and practical to apply but tend to sacrifice optimality for speed and cannot adapt fully to complex scenarios. Traditional heuristics are enhanced with global search strategies inspired by natural phenomena using meta-heuristic techniques. Genetic Algorithms, Particle Swarm Optimization, Ant Colony Optimization, and Simulated Annealing are common approaches. Because these techniques balance exploration and exploitation to escape local optima and find better solutions over time, they are more appropriate for more complex scheduling problems. Nevertheless, they are computationally expensive, and their performance greatly relies on parameter tuning. Lastly, machine learning based approaches are a modern and increasingly popular method of using historical data and patterns to make intelligent scheduling decisions. Training and reinforcement learning mechanisms can enable these methods to adapt to varying workloads and environments. Predict resource requirements, detect anomalies, and automate decision-making techniques, including deep learning, decision trees, clustering, and reinforcement learning.

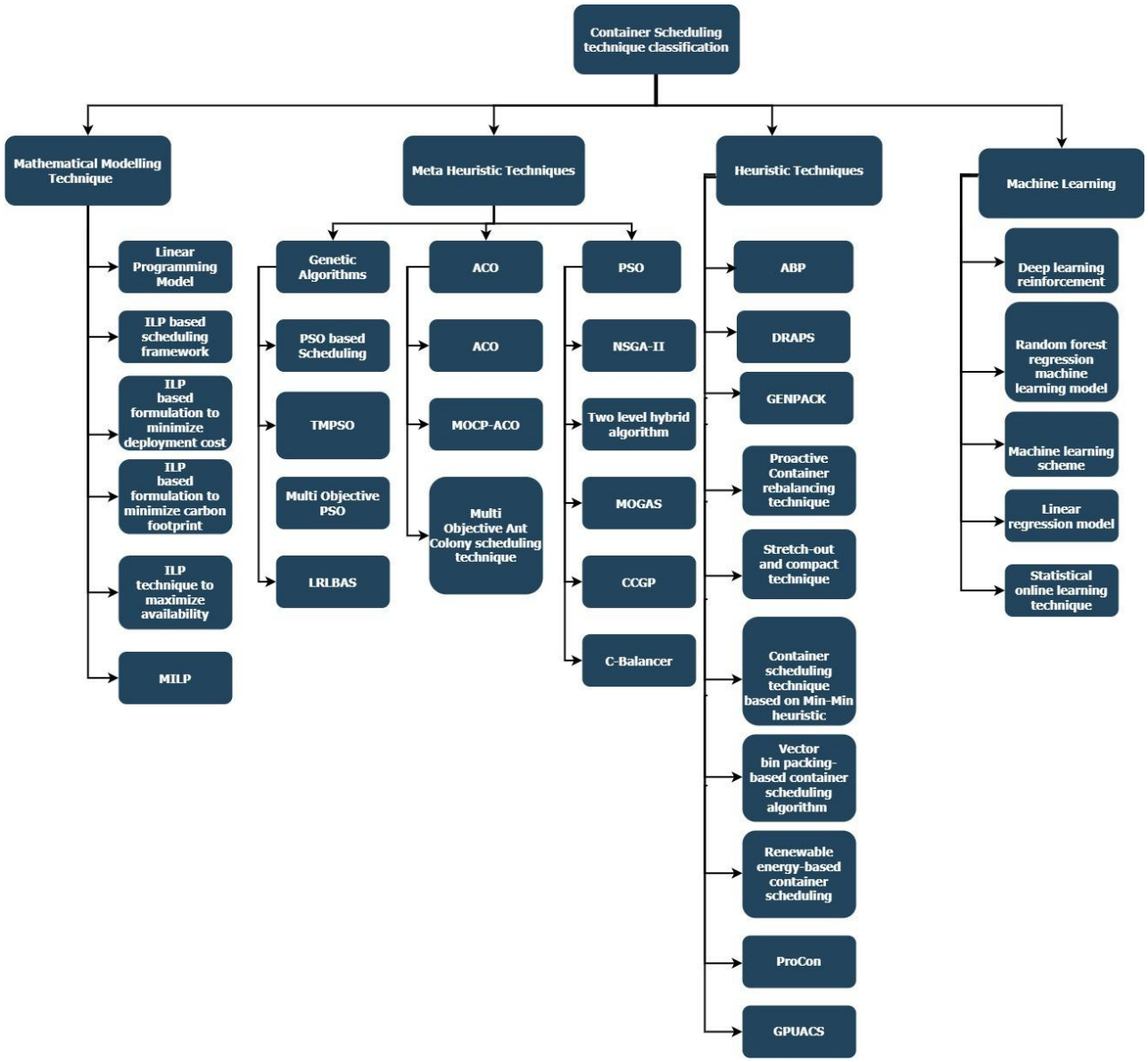

Fig. 3. Scheduling technique classification.

- **Mathematical modelling techniques:** These techniques generally solve the problem by constructing a mathematical equation with a set of limited constraints. After that, they use standardized techniques to find the optimal solution for the given problem. However, these techniques are more suited for small-scale problems.

Integer Linear Programming (ILP)-based Container scheduling is an optimization approach that formulates the scheduling and integer decision variables. Scheduling Containers to satisfy resource capacity, dependencies, and task priorities with minimum resource usage is known as Container scheduling. Table 4. Presents the various proposed/implemented mathematical modelling techniques for Container scheduling.

- **Heuristics:** Container heuristic scheduling algorithms are a class of algorithms that use heuristics or rule-based approaches to efficiently allocate resources and schedule containerized tasks within a computing environment. These algorithms aim to find near-optimal solutions without guaranteeing global optimality. Heuristic algorithms gained their popularity for finding approximate solutions. These algorithms generally have low complexity and provide better solutions within a low response time. Table 5. Represents the heuristic techniques implemented for better Container scheduling.

Table 4. Mathematical modelling techniques

Ref.	Year	Techniques	Parameters ered Consid-	Observations
[24]	2018	ILP-based scheduling framework	Resource Cost Utilization,	The proposed technique is robust and helps in im- proving the fault toler- ance.
[25]	2020	ILP-based for-Mulation to minimize carbon footprint	Energy, Resource Uti- lization, Load Balanc- ing, Makespan, Network	Reduces energy con- carbon sumption and footprint.
[24]	2019	Offline and on- line scheduling techniques	Load Make span Balancing,	Reduces total task interruption overhead.
[26]	2019	ILP technique to maximize availability	Availability, tion, Utiliza- Balancing, Load Makespan	High application availability compared to Docker scheduling strategies (binpack, spread, and random).
[27]	2021	MILP	Energy, Resource Utilization, Load Balancing, Network	Minimizes energy consumption.

Table 5. Heuristic Algorithms

Ref.	Year	Techniques	Parameters Considered	Observations
[28]	2019	Min-Min heuristic-Based on Container scheduling	Energy consumption	56% reduction in energy consumption compared to the First-Fit algorithm.
[29]	2018	Renewable energy-based Container scheduling	Energy consumption	15% energy saving compared to First-Fit or Random algorithms.
[30]	2020	GPUACS	QoS, Scalability	Able to schedule large-scale requests on data centers with over 20,000 servers in less than 3.5 seconds.

- **Meta-heuristics:** The main purpose of meta-heuristic algorithms is to find the optimal usage of nodes. These algorithms belong to the class of population-based optimization techniques. In general, each node is allotted a single task, and this allotting process is determined by a single criterion function. Assigning several tasks to a single node is possible only by enlarging the problem to include fictitious tasks or duplicate nodes. The generalized assignment problem accurately represents this situation, where tasks (like software development) are assigned to workers or computers in a network. Table 6 provides the detailed information of the Meta heuristic algorithm, which further includes ACO and PCO-based algorithms.

Table 6. Meta-heuristic

Ref.	Year	Techniques	Parameters Considered	Observations
[31]	2019	Two-level hybrid algorithm	Energy consumption	Significantly reduced energy consumption compared with First Fit, Best-Fit, etc.
[32]	2019	MOGAS	Resource Utilization, Energy consumption, Scalability, Availability	Outperforms ACO-based scheduling in achieving defined objectives.
[33]	2020	CCGP	Resource Utilization, Throughput	Provides efficient re- source utilization based on Swarm cluster experiments.
Ant Colony Optimization				
[34]	2019	MOCP-ACO	Network usage, Cost	Better handling of Container workload compared with the Spread strategy.
[12]	2019	Multi-objective Ant Colon y scheduling	Resource utilization, Network transmission, and Fault tolerance	Improved resource balancing and service reliability compared with Spread and GA-based techniques.
Particle Swarm Optimization				
[35]	2019	TMPSO	Energy consumption	Significant energy sav- ings compared with stan- dard and binary PSO.
[8]	2019	Multi-objective PSO	Energy consumption, Throughput	Reduces power con- consumption compared to traditional PSO.
[36]	2020	LRLBAS	Resource utilization	Demonstrates effective resource utilization compared to other PSO-based scheduling techniques.

- **Machine Learning:** It is one of the emerging techniques to deal with the scheduling of Containers. Machine learning-based Container scheduling algorithms use ML techniques to make intelligent scheduling decisions based on historical data, real-time information, and workload patterns. These algorithms aim to optimize resource allocation, improve task performance, and adapt to dynamic environments. However, these techniques are still under-explored in the context of Container scheduling. Table 7 depicts the machine learning based algorithms proposed year wise.

Table 7. Machine Learning Techniques

Ref.	Year	Techniques	Parameters Considered	Observations
[37]	2019	Random forest regression model	Time, Accuracy	Improved time and accuracy for Container need prediction compared to standard algorithms.
[38]	2019	Machine learning scheme	Energy consumption	Reduced number of active nodes and overall energy usage.
[39]	2020	Linear regression model	Load balancing, Energy consumption	Accurate resource utilization prediction leading to power savings.
[40]	2020	Statistical online learning technique	Power consumption	Better energy performance than binpack, spread, and random scheduling.

3. Results and Discussion

The comparative analysis of existing Container scheduling, such as Ant Colony Optimization (ACO), Bee Colony Optimiza- tion (BCO), Chicken Swarm Optimization (CSO), Particle Swarm Optimization (PSO), and Genetic Algorithm (GA), have been compared in the literature, showing different strengths and limitations that depend on the used performance metrics in any cloud computing environment. ACO and BCO, inspired by natural behaviors of ants and bees, respectively, show strong capabilities in finding near-optimal resource allocation paths, thus eventually reducing latency while achieving a fine-grained balance of resource utilization. These approaches excel in distributed environments where communication overhead and adaptability matter. However, these usually tend to have higher computational complexity that hampers their performance on large-scale deploy- ments. The basis of CSO is the social behavior of chickens. It has been noted to perform well during fluctuating workloads, making it also suitable for dynamic Containerized environments. Convergence speed may be much slower in comparison to other methods, therefore adding potential delays to scheduling decisions. PSO and GA represent a class of evolutionary algorithms: their major advantages are the efficient solutions they provide to resource allocation optimization by exploring several solutions simultaneously; this makes them very suitable for big, complex scheduling tasks. PSO has fast convergence rates, while GA is highly flexible and can escape local optima, hence assuring balanced loads. However, PSO and GA may be rather complex since their optimal performance could be reached after the tuning of several parameters. Although each of these techniques has some unique advantages, such as adaptability in CSO, quicker convergence with PSO, or the ability to avoid local optima by GA, the strength of these techniques may lie more contextually, and further, could be enhanced using hybrid approaches that may result in leveraging the strengths of multiple techniques toward better Container scheduling in cloud environments.

This section briefly discusses the ACO, BCO, CSO, PSO, and Genetic Algorithm with the help of proper pseudo code and mathematical formulation, and it also presents the shortcomings and benefits of each approach.

3.1 Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) is a nature-inspired algorithm that mimics the foraging behavior of ants to solve optimization problems. In this context, ACO is applied to optimize Container scheduling in an elastic containerized multi-cloud environment such as Docker Cloud. The ACO algorithm aims to optimize Container placement and resource allocation by guiding ants (representing Containers) towards optimal solutions, using pheromone trails to balance performance and resource availability across multiple cloud providers [41].

3.1.1 Problem Formulation

The objective of this problem is to minimize the **make-span**, which is the maximum finish time among all Container tasks.

Let us define the following:

- Set of Container tasks: $C = \{C_1, C_2, ..., C_{10}\}$
- Set of available cloud resources: $R = \{R_1, R_2, ..., R_m\}$
- Task deadline for each Container task: $D = \{D_1, D_2, ..., D_{10}\}$
- Execution time for each Container task: $E = \{E_1, E_2, ..., E_{10}\}$
- Resource requirements for each Container task: $\text{Req} = \{\text{Req}_1, \text{Req}_2, ..., \text{Req}_{10}\}$

3.1.2 Variables

- **Pheromone matrix**: Represents the pheromone levels on the edges between Container tasks and cloud resources.
- **Ant**: An agent that moves through the graph of Container tasks and cloud resources.
- **Solution**: A sequence of cloud resource assignments for each Container task.

3.1.3 Constraints

The Container scheduling problem must satisfy the following constraints:

1. Each Container task must be assigned to exactly one cloud resource:

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \in C \tag{1}$$

where x_{ij} is a binary variable that equals 1 if task C_i is assigned to resource R_j, and 0 otherwise.

2. Each cloud resource can only execute tasks within its capacity limit:

$$\sum_{i=1}^{10} \text{Req}_i \cdot x_{ij} \leq \text{Capacity}_j \quad \forall j \in R \tag{2}$$

3. The finish time of each Container task should not exceed its deadline:

$$F_i \leq D_i \quad \forall i \in C \tag{3}$$

4. The finish time F_i is determined by the start time S_i and execution time E_i:

$$S_i + E_i = F_i \quad \forall i \in C \tag{4}$$

5. $(x_{ij} \in \{0,1\}) forall (i \in C), (j \in R)$

3.1.4 Objective Function

The objective is to minimize the make-span, which is defined as the maximum finish time among all Container tasks:

$$\text{Objective} = \max(F_i) \quad \forall i \in C \tag{5}$$

Algorithm 1. Container Scheduling using Customized Ant Colony Optimization (ACO)

Input: Set of Container tasks T = {t_1, t_2, ..., t_n}, Set of cloud resources R = {r_1, r_2, ..., r_m}
Output: Optimal Container-to-resource scheduling with minimized make span
Initialize pheromone matrix $\tau_{ij} \leftarrow \tau_0$, where τ_0 is a small constant (e.g., 0.1) for each task-resource pair (i, j)
Define heuritic matrix $\eta_{ij} \leftarrow \frac{1}{\text{resource\_load}(r_j)}$ or based on estimated completion time
Set ACO parameters: α (pheromone importance), β (heuristic importance), ρ (evaporation rate), Q (pheromone deposit factor), number of ants **N**, max iterations I
best_solution ← ∅, best_makespan ← ∞
for iter = 1 to **I do**
  for k = 1 to **N do**
    Initialize solution $S_k \leftarrow \emptyset$
    Initialize resource state (available CPU, memory) for all $r_j \in R$
    for each task $t_i \in T$ **do**
      Compute selection probability for each r_j using:

$$P_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{\text{feasible}} [\tau_{il}]^{\alpha} \cdot [\eta_{il}]^{\beta}} \tag{6}$$

Select resource r_j using roulette wheel based on P_{ij} ensuring capacity constraints
Assign t_i to r_j, update resource state
Append (t_i, r_j) to S_k
    end for
Calculate the make span for S_k
Deposit pheromone locally:

$$\tau_{ij} \leftarrow \tau_{ij} + \frac{Q}{\text{makespan}_k} \tag{7}$$

for all assignments in S_k
    if makespank < best_makespan then
      best_solution ← S_k, best_makespan ← makespan_k
    end if
end for
Global pheromone evaporation:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} \quad \text{for all } (i, j) \tag{8}$$

Reinforce best solution:
$\tau_{ij} \leftarrow \tau_{ij} + \frac{Q}{\text{best\_makespan}}$ for all (t_i, r_j) ∈ best_solution
end for
return best_solution and best_makespan

The ACO algorithm constructs solutions by iteratively selecting Container tasks based on pheromone levels and heuristic information. Cloud resources are assigned to Container tasks while satisfying capacity constraints, and pheromone levels are updated based on the quality of the solutions. Over multiple iterations, the algorithm converges towards an optimal solution that minimizes the make-span while satisfying all constraints of the multi-cloud environment. Ant Colony Optimization (ACO) algorithm for Container task scheduling is proposed to efficiently schedule Container tasks over a distributed multi-cloud environment that imitates the intelligent foraging behavior of ants. The algorithm represents each artificial 'ant' as a potential solution for assigning Containerized workloads to cloud resources. Two key aspects of our model are pheromone levels, indicating the historical record of assigning tasks to certain resource(s), and heuristic information, that is, current system conditions, such as available resources, load balancing, and time. Ants initially tackle randomly the different assignments and over time gradually focus on better performance paths like lower make-span, higher resource utilization, and balanced loads, among others. Ants update the pheromone trails based on the quality of their solutions, and stronger pheromones are deposited for better solutions. Pheromone evaporation occurs over time so as not to allow premature convergence and promote exploration. This is an iterative process in which the stopping criterion is a maximum number of iterations, or convergence to a best or near-best solution. ACO is especially appropriate for these dynamic and complex scheduling problems as it reconciles exploration and exploitation, can reallocate initial resources, and considers more than one objective, such as minimizing total execution time and maximizing system throughput. Finally, the algorithm converges to a highly efficient container-to-resource assignment that meets capacity, deadline, and performance constraints of a cloud environment.

Table 7. Ant Colony Optimization (ACO) Customization for Container Scheduling

Element	Customization Details
Pheromone (τ_{ij})	Represents the desirability of assigning task t_i to resource r_j. Initialized with small values and updated based on solution quality. Higher pheromone values increase the probability of selection.
Heuristic Information (η_{ij})	Represents domain-specific knowledge such as the inverse of resource load, task execution time, or estimated completion time. Used to guide ants toward promising task-resource pairings.
α(Pheromone Importance)	Controls the influence of pheromone trails on task selection. Higher αemphasizes previously successful assignments. Typically set to 1.
β(Heuristic Importance)	Controls the influence of heuristic information on decision-making. A higher β value encourages ants to follow the most efficient or underutilized resources. Typically set to 2 or more.
ρ(Evaporation Rate)	Determines the rate at which pheromone information decays over time. Prevents unlimited pheromone accumulation and encourages exploration. Typical values are around 0.1.
Q(Pheromone Deposit Factor)	Controls the amount of pheromone deposited by each ant. Can be inversely proportional to the make span or other performance metrics to favor better schedules.
Probability Rule	Combining pheromone and heuristic values using a probabilistic formula to guide α β task-resource assignments:
Pheromone Update	Includes both local update (after each assignment) and global update (after all ants construct solutions). Best solutions reinforce good paths.
Capacity Constraints	Ensures that resource assignments made by ants do not exceed CPU, memory, or bandwidth capacities. Invalid paths are pruned from the probability distribution.
Termination Condition	The algorithm terminates after a maximum number of iterations or if no better solution is found over several iterations, helping balance efficiency and accuracy.

Table 7. summarizes the Ant Colony Optimization (ACO) algorithm modified and utilized for task scheduling in a cloud computing environment. Pheromone values (τ_{ij}), indicating how desirable it is to assign a task to a resource, guide each task-resource assignment and are updated upon evaluation of the solution. Heuristic information (η_{ij}) represents domain-specific knowledge, such as resource load or execution time, which informs each decision. The parameters α and β determine the relative influence of pheromone trails versus heuristic data; higher values make ants more likely to follow optimal or proven task-resource pairings. Pheromone evaporation rate (ρ) prevents saturation and promotes exploration, while the deposit factor (Q) controls how much pheromone is added based on solution quality. The probability rule combines τ_{ij} and η_{ij} to probabilistically guide ants toward the best paths. Both local and global pheromone updates reinforce effective solutions. Capacity constraints ensure solution feasibility by preventing resource overload.

3.2 Chicken Swarm Optimization (CSO) Algorithm

Chicken Swarm Optimization (CSO) is a metaheuristic approach inspired by the foraging behavior of chickens. It is applied to Container scheduling in an elastic Containerized multi-cloud environment like Docker Cloud to optimize resource allocation, load balancing, and minimize response time by efficiently distributing Containers across cloud providers based on their performance characteristics and availability. CSO leverages the collective intelligence of virtual "chickens" to find near-optimal solutions and adapt dynamically to changing conditions in the cloud environment.

3.2.1 Objective

Establishing a mathematical formulation to minimize the make-span (maximum finish time) of all Container tasks.

Input

- Set of Container tasks: $C = \{C_1, C_2, ..., C_{10}\}$
- Set of available cloud resources: $R = \{R_1, R_2, ..., R_m\}$
- Task deadline for each Container task: $D = \{D_1, D_2, ..., D_{10}\}$
- Execution time for each Container task: $E = \{E_1, E_2, ..., E_{10}\}$
- Resource requirements for each Container task: $Req = \{Req_1, Req_2, ..., Req_{10}\}$

Variables

- **Chicken**: An agent that represents a potential solution (scheduling assignment)
- **Flock**: A collection of chickens forming the population
- **Fitness**: A measure of how well a chicken (solution) performs

Constraints

- Each Container task must be assigned to exactly one cloud resource.
- Each cloud resource can execute tasks within its capacity limit.
- Deadline constraint for each Container task.

Algorithm 2. CSO Adapted for Container Scheduling

1: Set a population of chickens, where each chicken signifies a possible solution for the Container scheduling problem.
2: Estimate the fitness of each chicken by calculating the make-span objective function based on the assigned positions of Containers to cloud resources.
3: Set the best chicken and its fitness as the initial best solution.
4: **repeat**
5: **for** each chicken in the population **do**
6: Determine the neighboring chickens that the current chicken will interact with.
7: Exchange information or knowledge among the interacting chickens to modify their solutions, potentially through local search or exploration.
8: Evaluate the fitness of the modified solutions and compare them to the current best solution.
9: **if** the modified solution has a better fitness **then**
10: Update the best chicken and its fitness.
11: **end if**
12: **end for**
13: Update the population by applying individual or collective movement rules based on the exchanged information.
14: Perform additional local search or exploration to refine the solutions of the chickens.
15: Update the fitness values of the chickens in the population.
16: **until** an end state is met (e.g., a maximum number of iterations or convergence criteria)
17: Retrieve the best chicken (solution) found in the final iteration.
18: Decode the best chicken to obtain the scheduling assignments for each Container task.
19: Calculate the make-span based on the assigned positions and return it as the optimal solution.

3.3 Genetic Algorithm (GA)

The Genetic Algorithm (GA) is an evolutionary algorithm inspired by natural selection and genetics. It can be applied to optimize Container scheduling in an elastic Containerized multi-cloud environment such as Docker Cloud. The GA generates a population of potential solutions, evaluates their fitness, and iteratively evolves the population through selection, crossover, and mutation operations to find near-optimal solutions for Container scheduling.

3.3.1 Problem Formulation

The goal is to minimize the **make-span**, which is the maximum finish time of all Container tasks. The following sets and parameters define the problem:

- Set of Container tasks: $C = \{C_1, C_2, ..., C_{10}\}$
- Set of available cloud resources: $R = \{R_1, R_2, ..., R_m\}$
- Task deadlines for each Container task: $D = \{D_1, D_2, ..., D_{10}\}$
- Execution time for each Container task: $E = \{E_1, E_2, ..., E_{10}\}$
- Resource requirements for each Container task: $Req = \{Req_1, Req_2, ..., Req_{10}\}$

3.3.2 Variables

- **Chromosome**: A representation of a potential solution (scheduling assignment).
- **Gene**: An element in the chromosome representing the assignment of a Container task to a cloud resource.
- **Population**: A set of chromosomes representing potential solutions.
- **Fitness**: A measure of how well a chromosome (solution) performs based on the objective function.

3.3.3 Constraints

The Container scheduling problem is subject to the following constraints:

1. Each Container task must be assigned to exactly one cloud resource:

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \in C \tag{9}$$

where x_{ij} is a binary variable that equals 1 if task C_i is assigned to resource R_j, and 0 otherwise.

2. Each cloud resource can only execute tasks within its capacity:

$$\sum_{i=1}^{10} \text{Req}_i \cdot x_{ij} \leq \text{Capacity}_j \quad \forall j \in R \tag{10}$$

3. The finish time of each Container task must not exceed its deadline:

$$F_i \leq D_i \quad \forall i \in C \tag{11}$$

4. The finish time F_i is determined by the start time S_i and execution time E_i:

$$S_i + E_i = F_i \quad \forall i \in C \tag{12}$$

5. $(x_{ij} \in \{0,1\}) forall (i \in C), (j \in R)$

3.3.4 Objective Function

The objective is to minimize the make-span, which is the maximum finish time among all Container tasks:

$$\text{Objective} = \max(F_i) \qquad \forall i \in C \tag{13}$$

3.3.5 Genetic Algorithm for Container Scheduling

Algorithm 3. Container Scheduling using Customized Genetic Algorithm (GA)

1: **Input:** Set of Container tasks $T = \{t_1, t_2, ..., t_n\}$, Set of resources $R = \{r_1, r_2, ..., r_m\}$
2: **Output:** Optimal task-to-resource assignment with minimized makespan
3: Set GA parameters: population size P, crossover rate C_r, mutation rate M_r, number of generations G 4: Initialize population with Pchromosomes (each is a vector of length n, mapping tasks to resource IDs) 5: Evaluate fitness (e.g., inverse of makespan) for each chromosome
6: **for** g= 1 to G**do**
7: Select parents using tournament or roulette-wheel selection
8: Apply crossover with probability C_r to generate offspring:
Use 1-point, 2-point, or uniform crossover to exchange parts of the chromosome
9: Apply mutation with probability M_r to randomly reassign a task to a different resource (if feasible)
10: Evaluate fitness of new population
11: Apply elitism: retain top-performing chromosomes from current generation
12: Replace worst individuals to maintain population size P
13: **end for**
14: Select and return the chromosome with the highest fitness as the best solution

The GA algorithm evolves a population of solutions through selection, crossover, and mutation. Fitness evaluation is based on the make-span, and the algorithm aims to converge towards an optimal solution that minimizes the make-span while satisfying the constraints of the Containerized multi-cloud environment.

Table 8 contains the main components of the Genetic Algorithm (GA) for the container task scheduling in the cloud environments. A vector that represents each solution or chromosome, in which each index represents a container task and the value indicates the cumulative value of the assigned cloud resources. The fitness function generally evaluates solutions by making span (totally based on the inverse), and optionally other metrics like energy efficiency, resource utilization, and SLA compliance. To explore new solutions, crossover operations like one-point, two-point, or uniform crossover combine the features of two parent chromosomes to produce combinations of features.

Table 8. Genetic Algorithm (GA) Customization for Container Scheduling

Element	Customization Details
Chromosome Representation	Each chromosome is a vector where the index represents the Container task ID and the value represents the assigned cloud resource ID.
Fitness Function	Typically calculated as the inverse of makespan; can be enhanced with weighted metrics like resource utilization, energy consumption, and SLA satisfaction.
Crossover	Promotes exploration of new solutions by exchanging segments between two parent chromosomes; common methods include 1-point, 2-point, or uniform crossover.
Mutation	Introduces diversity by randomly changing the assignment of a task to another fea- sible resource; helps avoid local optima.
Selection	Selects parent chromosomes based on fitness using strategies like tournament selec- tion or roulette-wheel selection.
Elitism	Ensures the best-performing chromosomes are carried over to the next generation to preserve optimal partial solutions.
Constraints	Ensures that each chromosome (schedule) satisfies resource constraints such as CPU, memory, and bandwidth limitations.

3.4 Bee Colony Optimization (BCO)

Bee Colony Optimization (BCO) is a bio-inspired optimization algorithm based on the foraging behavior of honeybee colonies. In the context of Container scheduling in an elastic Containerized multi-cloud environment such as Docker Cloud, BCO optimizes Container allocation and resource management by dynamically adjusting based on resource availability, performance, and load balancing [34]. The BCO algorithm seeks to minimize the make-span, which is the maximum finish time of all Container tasks.

3.4.1 Problem Formulation

The objective is to minimize the **make-span**, which is the maximum finish time of all Container tasks. The following sets and parameters define the problem:

- **Set of Container tasks**: $C = \{C_1, C_2, \ldots, C_{10}\}$
- **Set of available cloud resources**: $R = \{R_1, R_2, \ldots, R_m\}$
- **Task deadlines for each Container task**: $D = \{D_1, D_2, \ldots, D_{10}\}$
- **Execution time for each Container task**: $E = \{E_1, E_2, \ldots, E_{10}\}$
- **Resource requirements for each Container task**: $\text{Req} = \{\text{Req}_1, \text{Req}_2, \ldots, \text{Req}_{10}\}$

3.4.2 Variables

- **Food source**: A potential solution (scheduling assignment).
- **Scout bee**: A bee that searches for new food sources (new solutions).
- **Employed bee**: A bee that explores and exploits existing food sources.
- **Onlooker bee**: A bee that selects food sources based on their quality and evaluates them.

3.4.3 Constraints

The Container scheduling problem is subject to the following constraints:

1. Each Container task must be assigned to exactly one cloud resource:

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i \in C \tag{14}$$

where $x_{ij} = 1$ if Container task C_i is assigned to resource R_j, and $x_{ij} = 0$ otherwise.

2. Each cloud resource must execute tasks within its capacity limit:

$$\sum_{i=1}^{10} \text{Req}_i \cdot x_{ij} \leq \text{Capacity}_j \quad \forall j \in R \tag{15}$$

3. The finish time of each Container task must not exceed its deadline:

$$F_i \leq D_i \qquad \forall i \in C \tag{16}$$

4. The finish time F_i is calculated as the sum of the start time S_i and the execution time E_i:

$$S_i + E_i = F_i \qquad \forall i \in C \tag{17}$$

5. $(x_{ij} \in \{0,1\}) forall (i \in C, j \in R)$

3.4.4 Objective Function

The objective is to minimize the make-span, which is the maximum finish time among all Container tasks:

$$\text{Objective} = \min(\max(F_i)) \qquad \forall i \in C \tag{18}$$

Algorithm 4. Container Scheduling using Bee Colony Optimization (BCO)

1: Initialize the food sources (potential solutions) randomly or through a heuristic method.
2: Evaluate the fitness of each food source by calculating the make-span based on the assigned positions.
3: Set the best food source and its fitness as the initial best solution.
4: **repeat**
5: **Employed bees phase:**
6: **for** each employed bee **do**
7: Select a neighboring food source to explore based on a defined strategy (e.g., roulette wheel selection).
8: Apply local search or neighborhood exploration to the selected food source by making small modifications.
9: Evaluate the fitness of the modified food source and compare it to the current best food source.
10: Update the best food source if the modified food source has a better fitness.
11: **end for**
12: **Onlooker bees phase:**
13: **for** each onlooker bee **do**
14: Select a food source to evaluate based on a defined strategy (e.g., roulette wheel selection weighted by fitness).
15: Apply local search or neighborhood exploration to the selected food source by making small modifications.
16: Evaluate the fitness of the modified food source and compare it to the current best food source.
17: Update the best food source if the modified food source has a better fitness.
18: **end for**
19: **Scout bees phase:**
20: **for** each scout bee **do**
21: Randomly generate a new food source and evaluate its fitness.
22: Compare the fitness of the new food source to the current best food source.
23: Update the best food source if the new food source has a better fitness.
24: **end for**
25: Update the employed bees and onlooker bees based on the fitness values of their food sources. 26: **until** A termination condition is met (e.g., a maximum number of iterations or convergence criteria). 27: Retrieve the best food source (solution) found in the final iteration.
28: Decode the best food source to obtain the scheduling assignments for each Container task.
29: Calculate the make-span based on the assigned positions and return it as the optimal solution.

The BCO algorithm uses a colony of artificial bees to search for and optimize Container scheduling solutions. The employed bees explore existing food sources (potential solutions), onlooker bees evaluate and exploit the best solutions, and scout bees search for new food sources. Through iterations of exploration and exploitation, the algorithm converges towards an optimal solution that minimizes the make-span while satisfying the Container scheduling constraints.

3.5 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a population-based optimization technique inspired by the social behavior of birds and fish. In the context of Container scheduling in an elastic Containerized multi-cloud environment such as Docker Cloud, PSO can optimize Container placement and resource allocation by iteratively updating the positions and velocities of particles (representing Container scheduling solutions) based on personal and global best solutions.

3.5.1 Problem Formulation

The objective is to minimize the **make-span**, which is the maximum finish time among all Container tasks. The following sets and parameters define the problem:

- Set of Container tasks: $C = \{C_1, C_2, ..., C_{10}\}$
- Set of available cloud resources: $R = \{R_1, R_2, ..., R_m\}$
- Task deadlines for each Container task: $D = \{D_1, D_2, ..., D_{10}\}$
- Execution time for each Container task: $E = \{E_1, E_2, ..., E_{10}\}$
- Resource requirements for each Container task: $\text{Req} = \{\text{Req}_1, \text{Req}_2, ..., \text{Req}_{10}\}$

3.5.2 Variables

- x_{ij}: Binary variable, $x_{ij} = 1$ if Container task C_i is assigned to cloud resource R_j, otherwise $x_{ij} = 0$.
- S_i: Start time of Container task C_i.
- F_i: Finish time of Container task C_i.

3.5.3 Constraints

The Container scheduling problem is subject to the following constraints:

1. Each Container task must be assigned to exactly one cloud resource:

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i \in C \tag{19}$$

2. Each cloud resource can execute tasks within its capacity limit:

$$\sum_{i=1}^{10} \text{Req}_i \cdot x_{ij} \leq \text{Capacity}_j \qquad \forall j \in R \tag{20}$$

3. The finish time of each Container task must not exceed its deadline:

$$F_i \leq D_i \qquad \forall i \in C \tag{21}$$

4. The finish time F_i is calculated as the sum of the start time S_i and the execution time E_i:

$$S_i + E_i = F_i \qquad \forall i \in C \tag{22}$$

5. $(x_{ij} \in \{0,1\})\text{forall}(i \in C, j \in R)$

3.5.4 Particle Swarm Optimization Algorithm

Algorithm 5. Container Scheduling using Particle Swarm Optimization (PSO)

1. Initialize the particle swarm population with random positions (Container scheduling solutions) and velocities for each particle.
2. Evaluate the fitness of each particle by calculating the make span based on the current positions.
3. Set the personal best position and fitness of each particle as the initial positions and fitness values.
4. Set the global best position and fitness as the position and fitness of the particle with the best make-span.
5. **repeat**
6. Update the velocity and position of each particle based on the PSO equations:

$$v_i^{(t+1)} = w v_i^{(t)} + c_1 r_1 \left(p_i^{(t)} - x_i^{(t)}\right) + c_2 r_2 \left(g^{(t)} - x_i^{(t)}\right) \tag{23}$$
$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}$$

where:
v(t) is the velocity of particle i at iteration t,
x(t) is the position of particle i at iteration t,
p(t) is the personal best position of particle i,
g(t) is the global best position of the swarm,
ω is the inertia weight,
c1, c2 are acceleration coefficients, and
r1, r2 are random values between 0 and 1.

7. Apply position update limits to ensure feasibility of solutions (e.g., ensure that the Container task assignment respects capacity and resource constraints).
8. Evaluate the fitness of each particle at the updated positions and update personal best and global best positions if necessary.
9. **until** A termination condition is met (e.g., maximum number of iterations or convergence criteria)
10. Retrieve the best position found by the swarm in the final iteration.
11. Decode the best position to obtain the scheduling assignments for each Container task.
12. Calculate the make-span based on the assigned positions and return it as the optimal solution.

The PSO algorithm models each Container scheduling solution as a particle in the swarm. Each particle's position represents a possible solution, and its velocity determines how it moves through the solution space. The algorithm iteratively updates the particle positions and velocities based on personal best and global best solutions, searching for the optimal Container scheduling solution that minimizes the make-span.

3.6 Discussion

The Container scheduling algorithms were implemented in Python because it provides a clean, simple, extensive library support and the ability to rapidly prototype the optimization algorithm. Custom Python scripts implementing the said metaheuristic algorithms (ACO, PSO, BCO, CSO, and GA) were developed with the help of libraries like NumPy

and Matplotlib for numerical and plotting regarding derivatives over a NumPy array. Simulations were run on a computer running Windows 10 and had an Intel Core i7 processor, 16 GB RAM, and Python version 3.10. The code was structured in a way such that the parameter tuning and repeated experiments were easy, and any evaluation was consistent. This environment proved very useful for the implementation, testing, and comparison of the scheduling algorithms flexibly and efficiently.

Table 10 provides a comparative analysis of five popular metaheuristic algorithms, namely Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Bee Colony Optimization (BCO), Chicken Swarm Optimization (CSO) and Genetic Algorithm (GA), applied for Container scheduling in a cloud environment. Critical performance metrics like CPU utilization, makespan, load balancing efficiency, convergence speed, scalability and implementation complexity are compared. We present the results which indicate that PSO and GA have good performance in terms of CPU utilization and in scalability, whereas CSO suffers relatively from most of the metrics. Under certain circumstances, ACO has balanced performance with slower convergence than BCO.

Fig. 4 depicts the comparative performance of five metaheuristic algorithms of ACO, PSO, BCO, CSO, and GA concerning two parameters of CPU utilization as well as make span. CPU utilization of PSO turns out to be the highest while the make span is the lowest, indicating an efficient utilization of resources and shorter time for completing the tasks. CSO (along with GA) also shows high performance on both metrics, while CPU utilization and makespan of CSO is the least. The graph offers a graphical summary so that the proper scheduling algorithms for cloud-based Container environments can be decided on a decision basis.

Table 10. Comparative Analysis of Metaheuristic Algorithms for Container Scheduling

Algorithm	CPU Utilization (%)	Makespan (sec)	Load Balancing Efficiency	Convergence Speed	Scalability	Code Complexity
Ant Colony Optimization (ACO)	85	120	Medium	Slow	High	Medium
Particle Swarm Optimization (PSO)	88	105	High	Fast	High	Low
Bee Colony Optimization (BCO)	82	115	Medium	Medium	Medium	Medium
Chicken Swarm Optimization (CSO)	80	125	Low	Medium	Medium	High
Genetic Algorithm (GA)	87	110	High	Medium	High	Medium

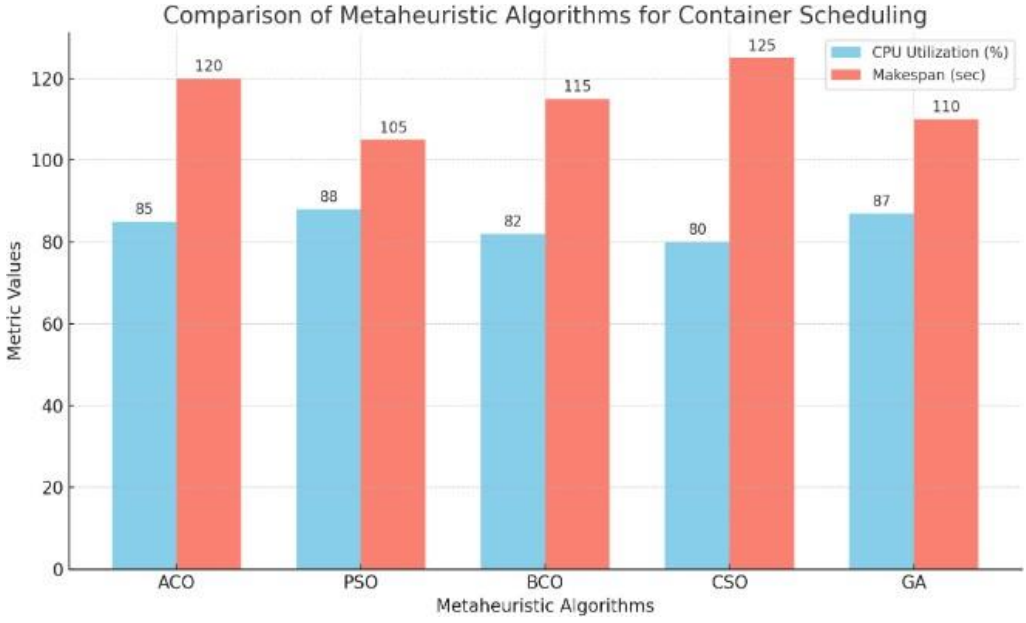

Fig. 4. Comparison of Metaheuristic Algorithms for Container Scheduling

4. Proposed Hybrid Approach for Efficient And Secure Container Scheduling

Implementing a secure Container scheduling algorithm is very crucial as it helps in enhancing the security of Containers in cloud computing by providing security practices such as regular security updates, access control, network segmentation, secure coding practices, encryption, vulnerability scanning, and strong authentication. Generally, Container scheduling algorithms are categorized into four types, namely, meta-heuristic, heuristic, machine learning, and mathematical modelling techniques. Container scheduling is an NP-hard problem [18].

It can be effectively solved with an algorithm inspired by nature. A lot of nature-inspired algorithms are proposed to improve Container scheduling performance, such as Ant colony optimization, Particle swarm optimization, Bee colony optimization, Chicken swarm optimization, and many others. This study proposes another nature-inspired algorithm, the Intelligent Water Drop Algorithm (IWD) with Anti-collocation and Anti-Affinity rule. This approach utilizes anti-collocation to avoid putting similar Containers on the same host, avoiding the risk of resource contention and further ensuring better load balancing. Moreover, anti-affinity rules would not allow those Containers to be placed near each other that would affect the performance of each other, thereby further optimizing system stability along with efficiency. The hybrid approach significantly amplifies the Container scheduling by improving the resource allocation and workload balance while minimizing performance bottlenecks in a cloud computing environment.

Intelligent Water Drops (IWDs) is an optimization algorithm based on the behavior of real water droplets. Inside Container scheduling, the IWD method attempts to place Containers over a number of hosts in a cloud environment such that Containers consume resources, balance load, and reduce execution time. Replicating the cooperative behaviour of water droplets and oper- ating on the concepts of swarm intelligence [10], it decides the best routes for node selection.

Each water drop is a separate placement of Container on the available resources. Properties it can take into include Container placement selections. The performance measures associated with the system are resource utilization and latency and security configurations such as isolation techniques, access control policies, energy consumption estimates, fault tolerance strategies, and load balancing policies. After defining the parameters there exists a need of quality evaluation which may include factors like resource utilization, security, QOS and energy efficiency [11].

Fig. 5. shows the Container orchestration workflow that can determine the optimal scheduling decisions within a cloud environment. This begins with a user issuing an application request, which, along with its dependencies, and the host operating system is packaged into a Container. IWD algorithm runs through a hybrid orchestration pipeline, identifying the most suitable node for a deployment.

The scheduling phase accounts for anti-collocation and anti-affinity rules to achieve higher fault tolerance, performance isolation as well as distributing resources in the cloud infrastructure. These rules ensure that no two Containers are on the same node or even in the same group of nodes. The intelligent, secure and policy-driven Container scheduling cycle then completes by executing the application in the selected node.

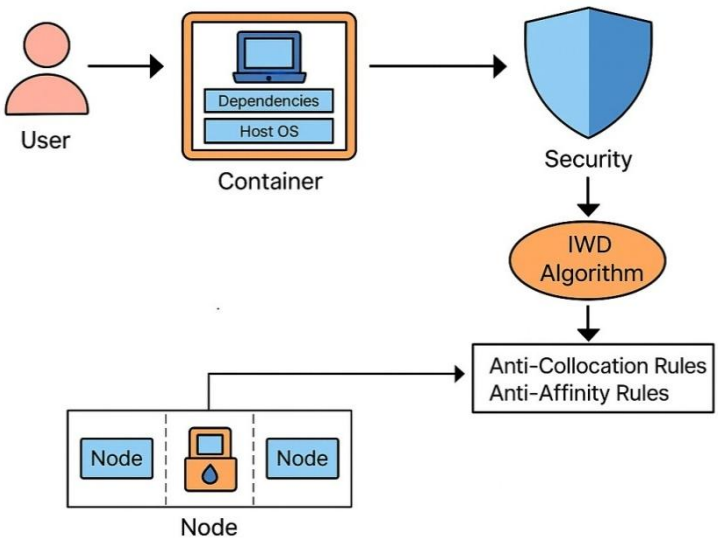

Fig. 5. IWD embedded with security

4.1 Novelty of this study

This paper presents a novel comparative analysis of existing Container scheduling techniques, specifically focusing on Ant Colony Optimization (ACO), Bee Colony Optimization (BCO), Chicken Swarm Optimization (CSO), Particle Swarm Optimiza- tion (PSO), and Genetic Algorithm (GA). The novelty of this paper is demonstrated through the following unique contributions:

- **Comprehensive Evaluation Metrics:** Unlike previous comparative studies, this paper provides an in-depth evaluation of these algorithms across multiple performance metrics specific to cloud environments, including computational com- plexity, adaptability, load balancing efficiency, latency reduction, and convergence speed. This holistic approach helps to understand the trade-offs of each technique under various workload conditions.
- **Context-Specific Analysis:** A key novelty lies in the context-specific analysis where each scheduling technique is eval- uated based on its performance in different cloud deployment scenarios—ranging from static environments to highly dynamic, Containerized multi-cloud setups. This detailed context-specific evaluation is critical for practitioners who must choose the best scheduling technique based on specific cloud configurations and workload dynamics.
- **Identification of Strengths and Weaknesses in Real-World Scenarios:** The paper identifies specific strengths and weak- nesses of each scheduling approach when applied to realistic cloud environments. For example, the analysis highlights the scalability challenges of ACO and BCO in large-scale environments and the slow convergence of CSO in dynamic contexts. By presenting these findings, the paper offers practical insights into the applicability of each algorithm.
- **Hybrid Approach Recommendation:** Another significant contribution is the recommendation of potential hybrid ap- proaches that could leverage the strengths of multiple techniques. The paper identifies combinations of existing meth- ods—such as merging the adaptability of Intelligent Water Drop algorithm with Anti-Collocation and Security Affinity Rules so that it could provide superior performance in Container scheduling with security. This suggestion for hybrid solutions opens avenues for future research and development.
- **Adaptability and Computational Complexity Trade-off Analysis:** This paper uniquely contributes to the understanding of the trade-off between adaptability and computational complexity in scheduling algorithms. While traditional analyses often focus solely on optimizing one performance metric, this study elucidates the inherent trade-offs, providing readers with a clear understanding of how adaptability impacts computational overhead.
- **New Benchmarks for Cloud Scheduling Performance:** The paper introduces new benchmark scenarios for cloud con- tainer scheduling, including fluctuating workloads, high communication overhead, and scalability requirements. By bench- marking the performance of ACO, BCO, CSO, PSO, and GA across these scenarios, this paper sets a foundation for future comparative studies in cloud scheduling research.

The contributions highlighted in this section set this work apart from other comparative studies by providing a detailed, context- sensitive and practical evaluation of Container scheduling techniques. The insights gained from this comprehensive analysis can guide both researchers and cloud practitioners in selecting the most suitable scheduling algorithm or designing novel hybrid approaches for improved performance in various cloud computing scenarios.

5. Conclusion

In this study, we review the different Container scheduling solutions and their performance in cloud environments through a comprehensive systematic literature review. The scheduling techniques are analyzed through four major approaches: machine learning based, metaheuristics and heuristics, mathematical modeling, and hybrid approaches. We explored key scheduling pa- rameters through the detailed analysis of these strategies, with focus on performance indicators such as CPU utilization, load balancing, and makespan, and corresponding tradeoffs between these metrics. The pros and cons of each technique were care- fully assessed to give a clear picture of their efficacy under specific conditions.

One important contribution of this paper is to identify the weaknesses and strengths of different scheduling methods and how these schedulers may be appropriate or not well suited to different cloud environments. Finally, a review of the schedulability factors including adaptability, scalability and computational complexity is presented which highlights the necessity of taking into account these factors when choosing an appropriate scheduling algorithm. More importantly, it highlights the promise of hybrid approaches, which combine machine learning methods with traditional optimization ones in order to improve both the performance and flexibility of Container scheduling.

There is no single algorithm which works well in all scenarios, however, a knowledge of the contextual performance and in- herent tradeoffs of each technique helps practitioners to make better decisions. Future research should identify ways to combine the adaptability of techniques such as Intelligent Water Drop (IWD) with the robustness of security and fault tolerance mecha- nisms. Moreover, exploring real time scheduling in dynamic and heterogeneous environments and integrating edge computing can create new paths to increase cloud resource utilization. Future research to achieve these goals can significantly enhance the efficiency, scalability, and sustainability of Containerized applications in cloud systems.

References

[1] Karthik Kambatla, Vamsee Yarlagadda, Íñigo Goiri, and Ananth Grama. Optimistic scheduling with service guarantees. Journal of Parallel and Distributed Computing, 135:246–258, 2020.

[2] Zhiming Shen, Zhen Sun, Gur-Eyal Sela, Eugene Bagdasaryan, Christina Delimitrou, Robbert Van Renesse, and Hakim Weatherspoon. X-Containers: Breaking down barriers to improve performance and isolation of cloud-native Containers. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 121–135, 2019.

[3] Imtiaz Ahmad, Mohammad Gh AlFailakawi, Asayel AlMutawa, and Latifa Alsalman. Container scheduling techniques: A survey and assessment. Journal of King Saud University-Computer and Information Sciences, 2021.

[4] Emiliano Casalicchio and Stefano Iannucci. The state-of-the-art in Container technologies: Application, orchestration and security. Concurrency and Computation: Practice and Experience, 32(17):e5668, 2020.

[5] Titus Team at Netflix. Scaling Container deployments with titus: Netflix's Container management platform. https://netflixtechblog.com/scaling-Container-deployments-with-titus-1f7b8f9f902b, May 2020. Accessed: 2025-04-21.

[6] Abdul Saboor, Mohd Fadzil Hassan, Rehan Akbar, Syed Nasir Mehmood Shah, Farrukh Hassan, Saeed Ahmed Magsi, and Muhammad Aadil Siddiqui. Containerized microservices orchestration and provisioning in cloud computing: A conceptual framework and future perspectives. Applied Sciences, 12(12):5793, 2022.

[7] Cai Zhiyong and Xie Xiaolan. An improved Container cloud resource scheduling strategy. In Proceedings of the 2019 4th International Conference on Intelligent Information Processing, pages 383–387, 2019.

[8] Mainak Adhikari and Satish Narayana Srirama. Multi-objective accelerated particle swarm optimization with a Container-based scheduling for internet-of-things in cloud environment. Journal of Network and Computer Applications, 137:35–61, 2019.

[9] Bo Liu, Pengfei Li, Weiwei Lin, Na Shu, Yin Li, and Victor Chang. A new Container scheduling algorithm based on multi-objective optimization. Soft Computing, 22(23):7741–7752, 2018.

[10] Yang Hu, Huan Zhou, Cees de Laat, and Zhiming Zhao. Ecsched: Efficient Container scheduling on heterogeneous clusters. In European Conference on Parallel Processing, pages 365–377. Springer, 2018.

[11] Karthik Kambatla, Vamsee Yarlagadda, Íñigo Goiri, and Ananth Grama. Ubis: Utilization-aware cluster scheduling. In 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 358–367. IEEE, 2018.

[12] Miao Lin, Jianqing Xi, Weihua Bai, and Jiayin Wu. Ant colony algorithm for multi-objective optimization of Container- based microservice scheduling in cloud. IEEE access, 7:83088–83100, 2019.

[13] Oana-Mihaela Ungureanu, Ca˘lin Vla˘deanu, and Robert Kooij. Kubernetes cluster optimization using hybrid shared-state scheduling framework. In Proceedings of the 3rd International Conference on Future Networks and Distributed Systems, pages 1–12, 2019.

[14] Kapil N Vhatkar and Girish P Bhole. Optimal Container resource allocation in cloud architecture: A new hybrid model. Journal of King Saud University-Computer and Information Sciences, 2019.

[15] Mohamed K Hussein, Mohamed H Mousa, and Mohamed A Alqarni. A placement architecture for a Container as a service (caas) in a cloud environment. Journal of Cloud Computing, 8(1):1–15, 2019.

[16] Han Li, Xinhao Wang, Sikun Gao, and Ning Tong. A service performance aware scheduling approach in Containerized cloud. In 2020 IEEE 3rd International Conference on Computer and Communication Engineering Technology (CCET), pages 194–198. IEEE, 2020.

[17] Heithem Abbes, Thouraya Louati, and Christophe Cérin. Dynamic replication factor model for linux Containers-based cloud systems. The Journal of Supercomputing, 76(9):7219–7241, 2020.

[18] Bo Liu, Jiawei Li, Weiwei Lin, Weihua Bai, Pengfei Li, and Qian Gao. K-pso: An improved pso-based Container scheduling algorithm for big data applications. International Journal of Network Management, 31(2):e2092, 2021.

[19] Tarek Menouer. Kcss: Kubernetes Container scheduling strategy. The Journal of Supercomputing, 77(5):4267–4293, 2021.

[20] Siyuan Zheng, Fenfen Huang, Chen Li, and Haobin Wang. A cloud resource prediction and migration method for Container scheduling. In 2021 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS), pages 76–80. IEEE, 2021.

[21] Yiwen Hu and Yuangang Lei. A Container cloud scheduling strategy based on qos. In The 2nd International Conference on Computing and Data Science, pages 1–5, 2021.

[22] Jigna Acharya and Anil C Suthar. Container scheduling algorithm in docker based cloud. Webology (ISSN: 1735-188X), 19(2), 2022.

[23] Sharma, Kanika, and Parul Khurana. "Performance Evaluation of the Nature-Inspired Algorithms for Container Scheduling for Elastic Containerized Multi-Cloud." 2024 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS). IEEE, 2024.

[24] SJingze Lv, Mingchang Wei, and Yang Yu. A Container scheduling strategy based on machine learning in microservice architecture. In 2019 IEEE International Conference on Services Computing (SCC), pages 65–71. IEEE, 2019.

[25] Tarek Menouer and Patrice Darmon. Containers scheduling consolidation approach for cloud computing. In Pervasive Systems, Algorithms and Networks: 16th International Symposium, I-SPAN 2019, Naples, Italy, September 16-20, 2019, Proceedings 16, pages 178–192. Springer, 2019.

[26] Xusheng Zhang, Ziyu Shen, Bin Xia, Zheng Liu, and Yun Li. Estimating power consumption of Containers and virtual machines in data centers. In 2020 IEEE International Conference on Cluster Computing (CLUSTER), pages 288–293. IEEE, 2020.

[27] Shubha Brata Nath, Sourav Kanti Addya, Sandip Chakraborty, and Soumya K Ghosh. Green Containerized service consolidation in cloud. In ICC 2020-2020 IEEE International Conference on Communications (ICC), pages 1–6. IEEE, 2020.

[28] Naylor G Bachiega, Paulo SL Souza, Sarita M Bruschi, and Simone Do RS De Souza. Container-based performance evaluation: a survey and challenges. In 2018 IEEE International Conference on Cloud Engineering (IC2E), pages 398–403. IEEE, 2018.

[29] Aravanan Muniswamy and Radhakrishnan Vignesh. Dsts: A hybrid optimal and deep learning for dynamic scalable task scheduling on Container cloud environment. Journal of Cloud Computing, 11(1):1–19, 2022.
[30] Ruiting Zhou, Zongpeng Li, and Chuan Wu. Scheduling frameworks for cloud Container services. IEEE/acm transactions on networking, 26(1):436–450, 2018.
[31] Kuljeet Kaur, Sahil Garg, Georges Kaddoum, and Song Guo. Esp-vdce: Energy, sla, and price-driven virtual data center embedding. In ICC 2020-2020 IEEE International Conference on Communications (ICC), pages 1–7. IEEE, 2020.
[32] Yanal Alahmad, Tariq Daradkeh, and Anjali Agarwal. Optimized availability-aware component scheduler for applications in Container-based cloud. In 2019 Sixth International Conference on Software Defined Systems (SDS), pages 194–199. IEEE, 2019.
[33] Bruno de Athayde Prata, Carlos Diego Rodrigues, and Jose Manuel Framinan. Customer order scheduling problem to minimize makespan with sequence-dependent setup times. Computers & Industrial Engineering, 151:106962, 2021.
[34] Feifei Chen, Xiaofeng Zhou, and Chao Shi. The Container scheduling method based on the min-min in edge computing. In Proceedings of the 4th International Conference on Big Data and Computing, pages 83–90, 2019.
[35] Ashok Kumar, Rajesh Kumar, and Anju Sharma. Energy aware resource allocation for clouds using two level ant colony optimization. Computing & Informatics, 37(1), 2018.
[36] Leonardo R Rodrigues, Marcelo Pasin, Omir C Alves, Charles C Miers, Mauricio A Pillon, Pascal Felber, and Guilherme P Koslovski. Network-aware Container scheduling in multi-tenant data center. In 2019 IEEE Global Communications Conference (GLOBECOM), pages 1–6. IEEE, 2019.
[37] Boxiong Tan, Hui Ma, and Yi Mei. A hybrid genetic programming hyper-heuristic approach for online two-level resource allocation in Container-based clouds. In 2019 IEEE Congress on Evolutionary Computation (CEC), pages 2681–2688. IEEE, 2019.
[38] Mahmoud Imdoukh, Imtiaz Ahmad, and Mohammad Alfailakawi. Optimizing scheduling decisions of Container manage- ment tool using many-objective genetic algorithm. Concurrency and Computation: Practice and Experience, 32(5):e5536, 2020.
[39] Boxiong Tan, Hui Ma, Yi Mei, and Mengjie Zhang. A cooperative coevolution genetic programming hyper-heuristics approach for on-line resource allocation in Container-based clouds. IEEE Transactions on Cloud Computing, 10(3):1500– 1514, 2020.
[40] Benjamin Burvall. Improvement of Container placement using multi-objective ant colony optimization, 2019.
[41] Tao Shi, Hui Ma, and Gang Chen. Energy-aware Container consolidation based on pso in cloud data centers. In 2018 IEEE Congress on Evolutionary Computation (CEC), pages 1–8. IEEE, 2018.
[42] Guisheng Fan, Liang Chen, Huiqun Yu, and Wei Qi. Multi-objective optimization of Container-based microservice scheduling in edge computing. Computer Science and Information Systems, 18(1):23–42, 2021.

Authors' Profiles

Kanika Sharma is working as an assistant professor in the department of programming techniques at the School of Computer Applications at the Lovely Professional University in Phagwara, Punjab, India. She is pursuing her Ph.D. in Computer Applications. Her research areas include cloud computing, network security, and gamification.

Parul Khurana is working as an associate professor in the department of programming techniques at the School of Computer Applications at the Lovely Professional University in Phagwara, Punjab, India. He received his Ph.D. degree from the Lovely Professional University, Phagwara, Punjab, India, in 2023. His research areas include data science, gamification, cloud computing, bibliometrics, and citation analysis.

How to cite this paper: Kanika Sharma, Parul Khurana, "Exploring and Implementing Container Scheduling Methods: A Comparative Review and Practical Approach", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.18, No.1, pp. 105-125, 2026. DOI:10.5815/ijieeb.2026.01.07