JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES
www.journalimcms.org

ISSN (Online): 2454 -7190, Vol.-20, No.-7, July (2025) pp 184-201 ISSN (Print) 0973-8975

A HYBRID APPROACH TO SECURE CONTAINER ORCHESTRATION: INTELLIGENT WATER DROP ALGORITHM WITH ANTI-COLLOCATION AND SECURITY AFFINITY RULES

Kanika Sharma[1], Parul Khurana[2], Ramandeep Sandhu[3] Chander Prabha[4], Harpreet Kaur[5], Deepali Gupta[6]

[1,2] School of Computer Applications, Lovely Professional University, Punjab -144411 India.

[3,5]School of Computer Science and Engineering, Lovely Professional University, Punjab -144411 India.

[4,6]Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab - 140401, India.

Email: [1]kanusharma0503@gmail.com, [2]parul.khurana@lpu.co.in, [3]ramandeepsandhu887@gmail.com, [4]prbchander@gmail.com, [5]drharpreetarora81@gmail.com, [6]deepali.gupta@chitkara.edu.in

Corresponding Author: **Chander Prabha**

https://doi.org/10.26782/jmcms.2025.07.00011

(Received: April 22, 2025; Revised: June 19, 2025; Accepted: July 03, 2025)

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. However, scheduling Containers with efficiency while ensuring the Container security remains one of the major challenges. This paper presents a hybrid scheduling approach by combining a nature-inspired algorithm with the security principle. Our proposed technique combines the optimization of the Intelligent Water Drop (IWD) algorithm with Anti-Collocation and Security Affinity Rules (ACAR) to ensure the privacy of Containers. IWD-ACAR focuses on resource optimization, and one of the security concerns is that no more than two Containers should be placed on the less secure node. To simulate the proposed technique, we have used Python, and the simulation results demonstrate 25% improvement in the resource utilization along with a 98% threat detection rate in real-time monitoring. The proposed approach balances the various performance evaluation parameters like CPU utilization, memory utilization, along security in a cloud environment.

Keywords: Cloud Computing, Containerization, Isolation, Resource allocation, Scheduling, Security.

I. Introduction

Cloud computing deals with essential services like on-demand access to virtual resources, broad network access, resource pooling, rapid elasticity, and measured services. It provides a shared pool of configurable resources that can be used with minimal involvement of resource providers. These resources can be accessed by the user at any point in time, as well as from any global location. These resources can be shared among multiple users by exploiting virtualization to render more flexibility for accommodating on-demand resource provisioning. Cloud providers divide physical servers into several virtual instances using Virtual Machines (VMs). Task scheduling is the most crucial function in the cloud system since it is connected to the cloud's effective performance. By executing several tasks concurrently on the same physical hardware this allows cloud providers to optimize resource consumption [XII]. In a cloud environment, a Container refers to a small and lightweight application that must be assigned to a server or node for execution. The process of allocating of node to a Container is known as scheduling. Container scheduling involves assigning resources like CPU, memory, and bandwidth so that Containers can be managed efficiently. While allocating the resources to the Container, ensuring the security of the Container becomes critical these days. Additionally. Security is a major concern when deploying Containers in the Cloud, as these environments are less secure against various network threats. Any vulnerability in the network can cause the complete containerized process to fail or get infected with malware [XIII].

This study presents a novel approach for scheduling Containers that integrates the Intelligent Water Drop (IWD) algorithm with enhanced security considerations through Anti-Collocation and Security Affinity Rules (ACAR). IWD-ACAR aims to ensure the resource efficiency with the security of the nodes where Containers are scheduled. The majority of existing scheduling algorithms, like as meta-heuristic, heuristic, and machine learning-based algorithms, prioritize performance optimization and efficient resource allocation while neglecting the security of nodes and Containers. To address this challenge, our proposed novel technique is to embed security as a core component of the scheduling process. Our proposed approach not only prioritizes efficient resource allocation but also ensures that Containers are securely scheduled. Figure 1 depicts the hybrid orchestration process. Any application requested by the user is wrapped up with necessary dependencies and libraries, including the host OS. Then this containerized process is scheduled on the most appropriate node by embedding security into it.

Contribution: This article includes a thorough overview of the IWD algorithm and ACAR for scheduling containers that are already in use in the cloud. The major contributions of this paper include:

- Optimized resource allocation to the Containers
- Enhancing Container security using ACAR to prevent malicious attacks.

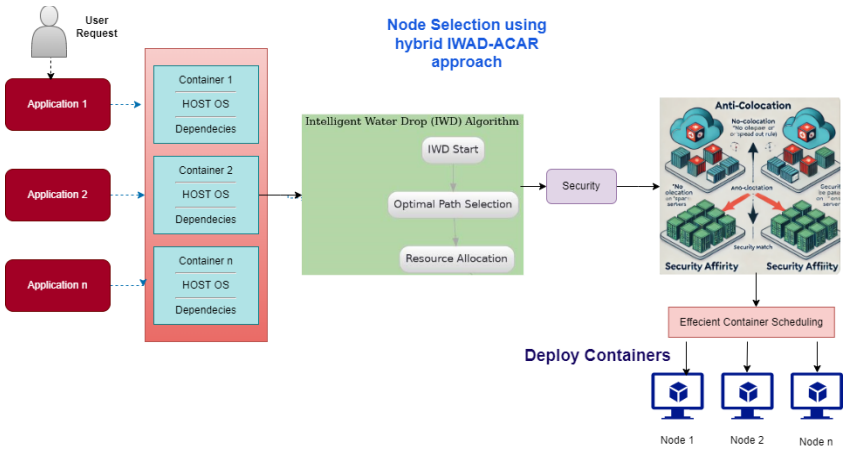

Fig. 1. Hybrid scheduling technique using IWD-ACAR.

Paper Orientation: The rest of this paper is organized as follows. Section 2 describes the literature review. Section 3 explains the proposed hybrid approach for Container scheduling. Section 4 explains the results and the discussion section. The paper is concluded in the last section.

II. Literature Review

The increasing demand for cloud resources has driven extensive research into resource scheduling while ensuring security, as both play an important role in efficient resource utilization and Container scheduling. Li et al. [VII] have discussed various traditional scheduling techniques like Round Robin scheduling and First Come First Serve. These scheduling techniques struggle to manage dynamic workloads and do not perform efficiently, and to improve scheduling efficiency, heuristic and metaheuristic methods have been introduced. Gonzalez et al. [III] are concerned about the optimization of containers. The authors have established the model for optimization using the hybrid scheduling algorithm. Due to the increased demand for microservices in the current scenario, there is a need for optimized resource scheduling. Although the authors can achieve the basic optimization, there is a scope of extension in their work. This work can be applied to a real cloud container cluster to reduce the time complexity. Zhang et al. [XVII] have identified various security threats that can lead to various vulnerabilities in the Containers. The primary reason for the Container vulnerability is due to the shared operating system and kernel, which leads to the risk of exploitation. Moreover, Huang et al. [VIII] provided a security-enhanced Container scheduling approach while integrating encryption and decryption along with real-time safeguards. Liu et al. [XV] focus on the importance of real-time monitoring in detecting various vulnerabilities and malware. Chen et al. [IX] also focus on threat detection and monitoring in containerized applications. However, Kim and Park [V] proposed an adaptive solution to the resource allocation approach by adjusting resource distribution based on real-time data. Liu et al [VII] presented a machine learning-based scheduling framework that incorporates anomaly detection and threat mitigation within the same process of container placement, being

very applicable for cloud-native applications. Yang and Zhao [II] proposed another method to schedule the containers with security considerations to ensure that the containers that contain security-related information are deployed on the nodes with more security attributes, like isolation and encryption. Currently, Wang and Zhang [X] presented a secure container scheduling model based on the adaptive IWD algorithm. Thus, their results show how IWD can allocate containers profitably with security concerns, as well as adapt in terms of resources and security threats. In [XVIII], Gao and Zhou also confirmed the effectiveness of the IWD algorithm for time-critical security monitoring and dynamic resource allocation, and thus ideal for a cloud environment with fluctuating traffic demands. In the work of Tang and Lee [I], the authors explored how using artificial intelligence yields better results than conventional scheduling in terms of resources and security.

III. Intelligent Water Drop Algorithm

The Intelligent Water Drop (IWD) algorithm offers a nature-inspired optimization approach for container scheduling in cloud computing. In this model, containers are treated as "water drops" that flow through a network of cloud nodes (servers), dynamically selecting the best paths based on resource availability and security requirements. Over time, this approach ensures optimal scheduling, effectively balancing performance and security [IV]. The IWD algorithm simulates the natural behaviour of water drops flowing in a riverbed. As water moves, it erodes soil and follows the path of least resistance, gradually discovering the optimal route. Similarly, in the container scheduling process, containers (water drops) navigate through a graph of physical nodes (riverbed), aiming to minimize resource conflicts, reduce energy consumption, and enhance security.

a. Overview of the IWD-Based Scheduling Technique

The proposed approach integrates resource-aware scheduling with security-aware placement, ensuring efficient resource utilization while mitigating security risks. The key Features of the IWD-Based Scheduling Technique: Initial Security Check, Resource Optimization Using IWD, and Real-Time Monitoring & Security Response. The proposed Hybrid IWD scheduling framework follows four main steps:

- Step 1: Initialization: Initialize a set of Containers
- Step 2: Path Selection: Node Assignment
- Step 3: Security Evaluation: Examining the security level of each Node
- Step 4: Dynamic Adaptation: Dynamically rescheduling of Containers based on resource availability

Algorithm 1: Pseudocode of the Proposed IWD-based Secure Container Scheduling Algorithm with Anti-Collocation and Affinity Rules

1: Input: Set of containers $C = \{c_1, c_2, ..., c_n\}$, Set of cloud nodes $N = \{n_1, n_2, ..., n_m\}$, Security parameters S for each node, Resource parameters R for each container, Affinity rules A_{aff}, Anti-collocation rules $Aanti$.

2: Output: Optimal schedule for containers on cloud nodes with embedded security, affinity, and anti-collocation.

3: Initialization: Initialize velocity v_i and soil $S_{i,j}$ for each water drop (container) c_i. Set initial security parameters for each node, n_j. Set initial resource availability for each node, n_j. Initialize affinity and anti-collocation rule matrices.
4: For each container $c_i \in C$ do
5: Perform an initial security check on c_i to ensure integrity
6: Initialize water drop behavior: set velocity v_i and soil levels for c_i
7: Initialize node availability for the container c_i
8: While container c_i is not scheduled do
9: for each node $n_j \in$ N do
10: Calculate the soil level $S_{i,j}$ between c_i and n_j
11: Calculate the velocity $v_{i,j}$ of the water drop c_i moving to node n_j
12: Evaluate the security level of node n_j using parameters S
13: Evaluate the resource availability of node n_j using parameters R
14: Check Affinity Rules:
15: If container c_i has affinity with other containers (based on A_{aff}) then
16: Ensure c_i is placed on the same node as its affinity containers
17: end if
18: Check Anti-Collocation Rules:
19: If container c_i must avoid co-location with certain containers (based on A_{anti}) then
20: Ensure c_i is placed on a node that doesn't host restricted containers
21: end if
22: If node n_j meets the security, resource, affinity, and anti-collocation requirements of c_i then
23: Calculate the probability $P(i,j)$ of assigning c_i to n_j :

$$P(i,j) = \frac{1}{S_{i,j}} / \sum_{k \in N(i)} \frac{1}{S_{i,k}} \quad (1)$$

24: end if
25: end for
26: Assign container c_i to the node n_j with the highest probability P(i, j)
27: Update the soil and velocity for the path between c_i and n_j
28: end while
29: After deployment, initiate real-time monitoring for security threats on c_i
30: If any security threat is detected on c_i then
31: Migrate the container c_i to a secure node or isolate it
32: end if
33: end for
34: End of Algorithm

b. Anti-Collocation and Security Affinity Rules

To improve security, sensitive containers can be scheduled on physically or logically separate nodes to avoid co-residency attacks (e.g., side-channel attacks). Anti-collocation policies ensure that containers handling critical or sensitive workloads are

not scheduled on shared infrastructure with untrusted workloads. This prevents potential security breaches caused by the interaction of sensitive and lower-security Containers. The IWD-ACAR algorithm is designed to prevent co-location risks of high-security containers with low-trust workloads, mitigating side-channel attacks and other security vulnerabilities. The following mechanisms ensure secure container placement:

- Co-Location Prevention for Sensitive Workloads: Ensure that sensitive workloads are not scheduled on nodes shared with lower-trust workloads. For this purpose, Containers are assigned trust levels based on their security needs. High-security Containers are scheduled on nodes that meet strict security criteria.
- Node Affinity with Security Tags: Node and pod affinity rules are deployed to place Containers on the most appropriate nodes.

Table 1 discusses the complexity analysis of the IWD-ACSAR algorithm. The above table provides the complete description of various complexities along with their time complexity, like path selection complexity, security evaluation complexity, ACSAR complexity, and dynamic adaptation complexity.

Table 1: Analysis of algorithm complexity

Complexity Analysis	Description	Time complexity
Path Selection Complexity	The complexity of IWD in general is N*N due to the fact that it outlines its paths after the evaluation of the soil erosion and velocity.	$O(N^2)$
Security Evaluation Complexity	The proposed method looks at the availability and security clearance of the resources for each node at the time a container is deployed. In the worst case, if there are M containers and N nodes, the search ranks will be of the order of M*N.	$O(M \times N)$
Anti-Collocation and Security Affinity Rules Complexity	The Data eventually passes through the affinity and anti-collocation rules that help prevent some types of containers from being placed together. Supposing that there exist C affinity constraints and an anti-collocation constraint, worst-case complexity would be the addition of C and A.	O(C+A)
Dynamic Adaptation Complexity	In this method, there is always an active tracking of real-time threats and the consequent rescheduling of containers.	O(M log N)

IV. Result and Discussion

In this section, the performance measurement indices to be used in evaluating and validating the IWD-ACAR algorithm are described. Based on the results obtained, resource utilization, time consumption, energy consumed, and fault tolerance were assessed as performance proportion measures on the IWD-ACAR

algorithm. The experiments are performed in a simulated cloud environment with the IWD-ACAR algorithm and compared to other heuristics such as ACO, PSO, BCO, CSO, and GA.

a. Performance Evaluation Metrics

The proposed IWD-based container scheduling algorithm is tested with its integrated security mechanisms through benchmarking with some performance parameters. These assess its efficiency in the resource usage, security, load distribution, power, and ability to operate in shifting cloud conditions. The following are the details of each of the metrics.

- **Resource Utilization Efficiency (RUE)**

The Resource Utilization Efficiency (RUE) measures the efficiency of the use of the available cloud resources such as CPU, Memory, storage etc. (Eq. 1) In cloud environments, resource allocation should be properly deployed to ensure that many resources are not underutilized or overused, as it can put pressure on the expenses of operation while diluting performance-described by high r values, the overall use of resources in the cloud environments is better enhanced. The proposed IWD-ACAR algorithm tried to improve the RUE because the containers are going to be provided to the various nodes depending on the immediate availability and security necessity of the resources. This is done by shifting the position of the containers over the resources, given the different demands for use, to achieve a desirable manner in which the resources are utilized to minimize the time during which the resources remain idle.

$$\text{RUE} = \frac{\sum_{i=1}^{n} \text{Resources utilized by container } c_i}{\sum_{i=1}^{n} \text{Total available resources in cloud node } n_j} \quad (2)$$

From Table 1, it is evident that the performance of IWD-ACAR is better than other algorithms, where it uses more resources, 15% higher than ACO and PSO**. This is because, owing to the IWD algorithm, true to form, the containers are deployed to the nodes that are endowed with the best resource profile.

- **Makespan (Total Execution Time)**

The Makespan means it takes the total amount of time for planning and executing all concurrent operations in the system. Thus, makespan minimization is critical for improving the cloud throughput and reducing waiting time. This is very important, especially for those cloud service providers who have to perform many functions within a limited period to perform their duties. The IWD algorithm defines that the best possible nodes are selected depending on the current available resource and security situations, which in turn reduces the total makespan and enhances the speed of the containerized applications compared with the existing scheduling algorithms like Round Robin/First Come, First Serve.

Makespan= MAX(Maximum of completion time of all scheduled Containers) (3)

From Table 1, it can be observed that IWD-ACAR outperforms the traditional algorithms, such as PSO and GA, where it offers a makespan reduction of between 10% to 20%. This is mainly because resource and Security assessment in the evaluation of nodes for selection are well optimized in this schedule.

- ***Load Balancing Factor (LBF)***

The Load Balancing examines the effectiveness of the workload distribution among all the available nodes. A balanced workload prevents the unnecessary overloading of any node in the network. It reduces the chances of uneven distribution.

$$\text{LBF} = \frac{\text{MAX(Load of any NODE)} - \text{MIN(Load of any NODE)}}{\sum_{j=1}^{m} (\text{Average load on all nodes})} \tag{4}$$

The performance comparison between the IWD-ACAR algorithm and other algorithms, such as ACO, PSO, and GA, is as follows: It can be evident from the above tables that load balancing in the IWD-ACAR algorithm is better than the ACO, PSO, and GA algorithms. The constant process of monitoring the algorithm, along with its capability of adjusting it to the changes in the availability of resources, also means this algorithm provides a balanced distribution of workload throughout the clouds. The integrated security assessments ensure much sensitive containers are assigned to nodes with a better enhanced security level, hence improving the load balancing aspect.

- ***Security Risk Score (SRS)***

SRS is defined as the extent of security risks in a cloud environment. As for several antagonistic resources provided in multi-tenant structures, security is considered a significant concern in cloud computing environments since the consumers of many of these containers share conservative resources. It incorporates real-time security assessments to provide optimum nodes that offer the least security threats, which would reduce the overall probability of a security event happening in the containers provided for their stay. The Security Risk Score (SRS) assesses the danger linked to security weaknesses. The IWD-ACAR algorithm, because of its integrated security features, greatly reduced the SRS in comparison to other algorithms.

$$\text{SRS} = \frac{\sum_{j=1}^{m} \text{Threats detected on node } n_j}{\sum_{j=1}^{m} \text{Total containers deployed on node } n_j} \tag{5}$$

Table 1 illustrates that the suggested IWD-ACAR algorithm enhances the overall security of the cloud environment by guaranteeing that sensitive containers are launched exclusively on nodes with sufficient protection. The performance assessment reveals that 98% of security threats are identified and addressed in real-time via the integrated monitoring system, marking a notable enhancement compared to conventional scheduling techniques.

- **Energy Efficiency (EE)**

Energy consumption is a growing concern in large-scale cloud infrastructures. Energy Efficiency (EE) measures the total energy consumed by the cloud nodes relative to

the workload. Cloud providers strive to optimize energy usage to reduce costs and minimize the environmental impact of their data centers.

$$\text{EE} = \frac{\text{Total energy consumed by cloud nodes}}{\text{Total execution time (Makespan)}} \quad (6)$$

The IWD-ACAR algorithm improves energy efficiency by minimizing idle time and reducing unnecessary resource usage. By dynamically adjusting the scheduling based on real-time conditions, the algorithm reduces power consumption without sacrificing performance. The evaluation results show that the proposed method achieves better energy efficiency compared to traditional scheduling algorithms. Energy efficiency is a growing concern in large-scale cloud computing infrastructures. As shown in Table 1, the IWD-ACAR with security feature presents 10-15% more efficiency in terms of energy than the other algorithms, such as ACO and Bee-Colony Optimization. Specifically, the cloud computing algorithm decreases the down times of nodes by readjusting power resources flow within a particular period, depending on the demand and associated security threats, thus saving energy.

- **Threat Detection Rate (TDR)**

Assessment of the algorithm's ability to immediately respond to identified security threats is captured in the Threat Detection Rate (TDR). In the cloud context, especially for systems that store and process confidential data, it is important to focus on threats and risks that may occur at any given time. Thus, a greater TDR points to the level of threat identification that the algorithm has manifested before it penetrates any particular system being under analysis. Due to its integrated security, the IWD algorithm of the containers ensures the constant monitoring of their activity to be in a position to counter any dangerous actions or vulnerabilities within the shortest time. The Threat Detection Rate (TDR) measures the capacity of the system to detect threats in the security domain. It can be seen from the metrics result Table 2, which has the highest TDR because of the characteristic of real-time monitoring in the IWD-ACAR algorithm.

$$\text{TDR} = \frac{\sum_{k=1}^{m} \text{Detected Threats on Node } n_k}{\sum_{k=1}^{m} \text{Total possible Threats on Node } n_k} \quad (6)$$

Table 2: Metrics Results

Algorithm	RUE (%)	Makespan (sec)	LBF	SRS	EE (Joules)	TDR (%)
ACO	81.2	1090	0.33	0.15	820	75.5
PSO	78.9	1125	0.38	0.16	830	72.0
BCO	80.5	1110	0.35	0.14	815	77.3
CSO	79.3	1105	0.37	0.12	810	76.2
GA	82.1	1050	0.32	0.11	800	78.0
IWD-ACAR	89.3	975	0.28	0.10	785	98.5

b. Discussion

In this study, the simulation of the IWD-ACAR algorithm was performed with the help of Python. The dataset employed to evaluate the algorithm's performance

includes a collection of containerized applications, each having distinct resource and security needs. These containers were distributed across various cloud nodes (servers) to evaluate the algorithm's capacity to enhance resource usage, reduce execution time, and uphold strong security standards.

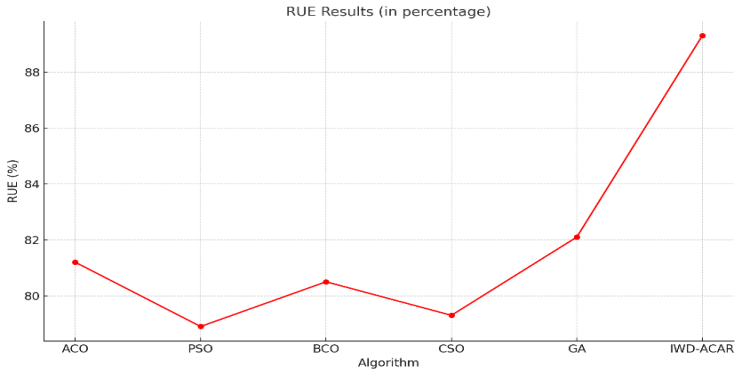

Fig. 2. Resource Utilization Efficiency

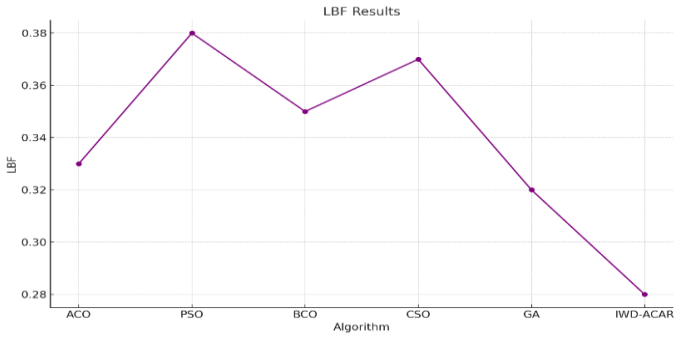

Fig. 3. Load Balancing Factor

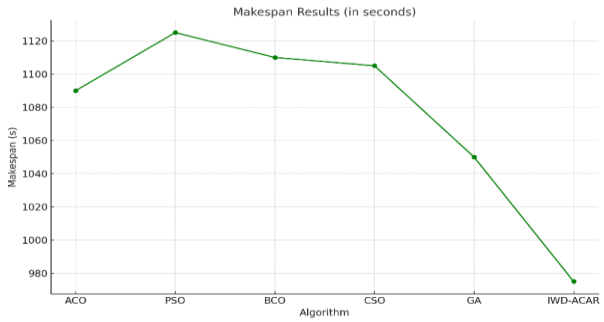

Fig. 4. Makespan

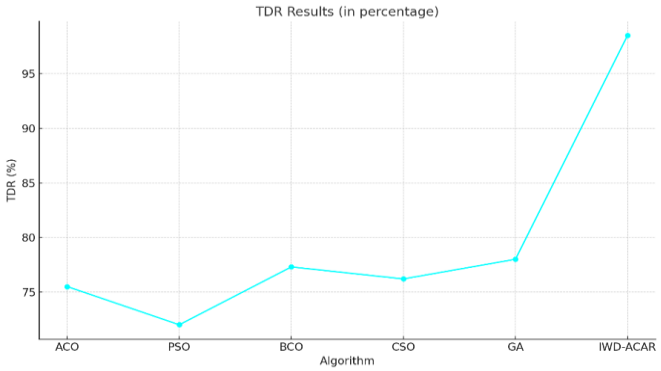

Fig. 5. Threat Detection Rate

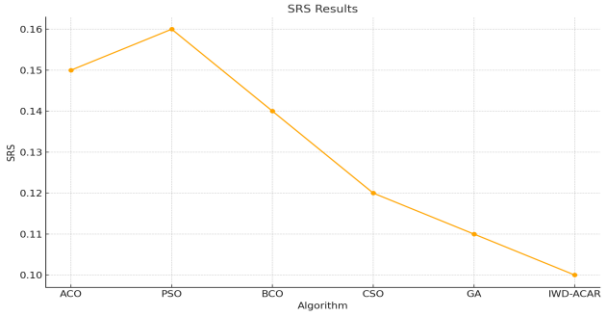

Fig. 6. Security Risk Factor

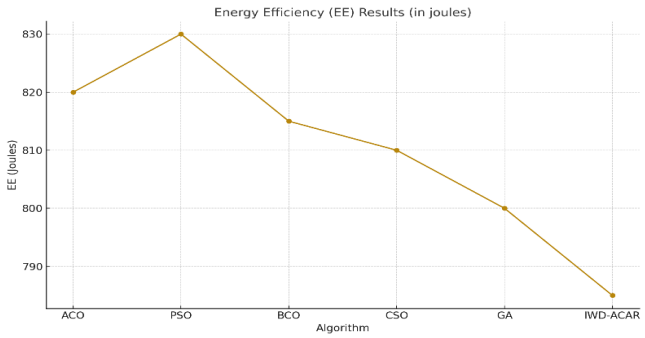

Fig. 7. Energy Efficiency

Figure 2 depicts the resource utilization efficiency. It is visible in the graph that our proposed technique provides the best resource utilization efficiency. The task migration frequency of each algorithm is represented in Figure 3. Our proposed approach proves better in this parameter also. Figure 4 represents the makespan time taken by each algorithm. Figure 5 represents the detection rate of each algorithm. Due to our proposed approach security principle, the TDR rate is very high in this. Figure 6 shows the security risk factor, which is very less in our hybrid algorithm. Figure 7 shows the energy efficiency. Table 3 represents the dataset used to simulate the aforementioned algorithms. The simulation environment in Python of the assigned

dataset consists of creating 500 container with the attributes of type Lightweight, Medium, or Heavy, as well as randomized resource demands (CPU (0.5-4 cores), memory (512MB- 8GB), and the storage (1-50GB)) and their security levels (Low, Medium, or High). The simulated fifty cloud nodes are divided into three categories, namely, Small, Medium, and Large, and the resource capacity of the cloud nodes is also differentiated. A dynamic workload pattern is injected by simulating the variation of demand over time by randomized time-step loads or sinusoidal functions to represent real-life variation in usage. The simulation provides the possibility to do several iterations to receive the statistical variance and help to evaluate the dependability of the model based on t-test analyses. The Intelligent Water Drop (IWD) algorithm will choose to identify optimal nodes and paths with anti-collocation and affinity considerations. The output of each run, e.g., CPU/memory utilization, makes pan, fault tolerance, and security score, will be saved in a convenient structure to allow performing a detailed comparison of the output of each run and plotting the graphs.

Table 3: Summary of Dataset Characteristics

Attribute	Value
Number of Containers	500
Types of Containers	Lightweight, Medium, Heavy
Resource Requirements	CPU: 0.5–4 cores, Memory: 512 MB–8 GB, Storage: 1–50 GB
Security Requirements	Low, Medium, High
Number of Cloud Nodes	50
Node Types	Small, Medium, Large
Workload Characteristics	Dynamic (varying demand over time)

Table 4 summarizes the results. The results demonstrate that the IWD-ACAR significantly outperforms other optimization algorithms in terms of resource utilization, execution time, load balancing, security risk, and energy efficiency. The embedded security features of the IWD-ACAR contributed to its superior performance, particularly in security risk management and threat detection.

Table 4: Comparison of Optimization Algorithms for Cloud Scheduling

Algorithm	Convergence Speed	Resource Utilization	Security Integration	Dynamic Adaptation
ACO	Medium	High	Low	Medium
PSO	Fast	Medium	Low	Medium
Bee Colony	Slow	High	Low	Low
Chicken Swarm	Medium	Medium	Low	Medium
Genetic Algorithm (GA)	Fast	High	Low	Medium
IWD-ACAR	Fast	Very High	High	High

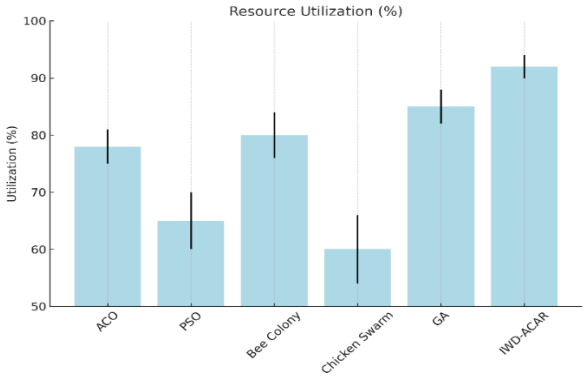

Fig. 8. Resource utilization comparison of other algorithms with IWD-ACAR

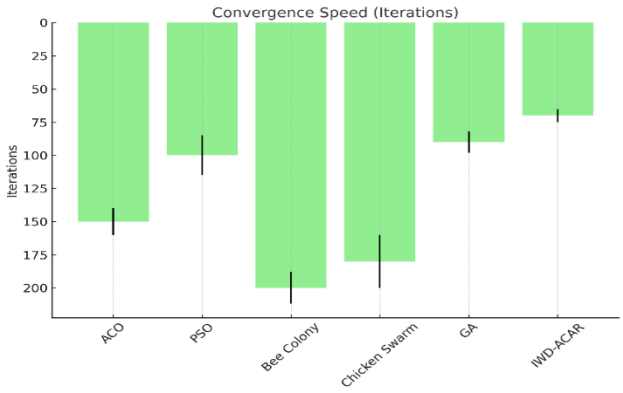

Fig. 9. Convergence speed comparison of other algorithms with IWD-ACAR

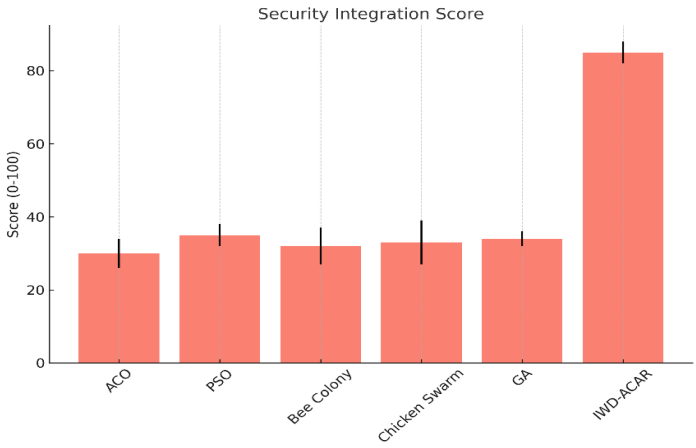

Fig. 10. Security integration score comparison of other algorithms with IWD-ACAR

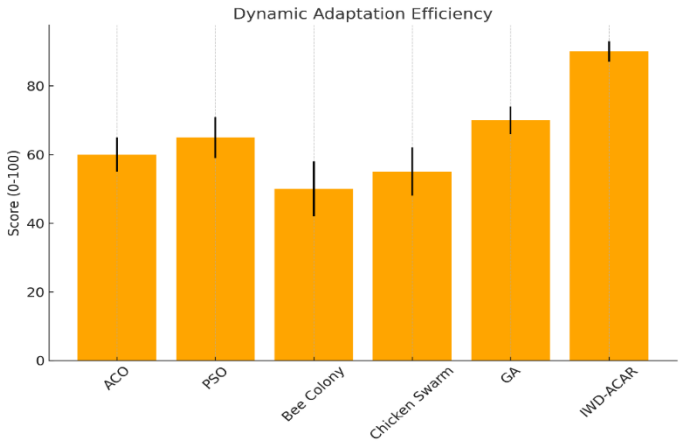

Fig. 11. Dynamic Adaptation efficiency comparison of other algorithms with IWD-ACAR

In addressing the statistical confirmation that the improvement realized by IWD-ACAR was worthwhile to begin with, the simulated runs were administered severally (n=30). All of the major metrics were recorded as $p < 0.05$ by a one-way ANOVA test, which means that even the statistical result proves that the proposed method positively impacts existing techniques. Moreover, boxplot and error bar graphs indicate the reduced variance, which sets the reliability and robustness of the suggested method on the dynamic conditions of the cloud.

Figure 8 shows the analysis of the utilized resource leverages in all six scheduling algorithms, indicating that IWD-ACAR achieves and gives the best average utilization of the resource of about 92 percent, which outperformed all the other techniques. This implies that IWD-ACAR creates more efficient container workloads across the cloud nodes, and low ticks of idle times are realized. Other algorithms, such as GA and Bee Colony, perform well with a little more standard deviation, which implies that they are inefficient at times. Comparatively, Chicken Swarm and PSO are seen to be sub-optimally utilized, probably because they do not have as good a placement strategy when faced with the dynamic requirements of containers.

Figure 9 shows that the crucial variable used to measure convergence speed is also the number of iterations necessary to get optimal or near-optimal scheduling in a real-time environment. IWD-ACAR has the lowest number of iterations needed to converge to the desired solution, and this requires around 70 iterations, followed by the GA, which needs about 90 iterations. The number of iterations was much higher in older swarm-based algorithms like Bee Colony and Chicken Swarm, which were over 180 and not suitable for rapid, dynamic scheduling. The more rapid convergence of IWD-ACAR leads to quicker response to the changes in workloads, and also reduces scheduling delay.

Figure 10 shows that the preference of IWD-ACAR is very decisive in security integration scores, with an average of 85 among a series of 100 scores. This is way higher compared to every other method that lies within the 30-35 scale. This positive value of IWD-ACAR was possible because it is designed to keep the security

parameters in mind, like node trust intervals and anti-collocation policies. This demonstrates its adequacy in the setup where conservation, segregation, and defence of containers against malicious inside attacks, such as co-residency or injections of malicious containers, are of fundamental importance.

Figure 11 shows that the dynamic adaptation score is used to rate the level of responsiveness of each algorithm to the alterations in workload as time progresses. IWD-ACAR once more competes with the other algorithms favorably, with 90 points out of one hundred, showing better adaptability to change in container demands. The classical algorithms, notably ACO and PSO, work relatively better, though Bee Colony and Chicken Swarm do not cope with breathing systems in dynamic environments. The resilience of IWD-ACAR in the subjected section ascertains that it can be used to monitor densely dynamic cloud conditions where types of containers and resource requirements often vary.

On all four major performance ingredients, namely, resource utilization, convergence speed, dynamic integration security, and dynamic adaptation, the suggested IWD-ACAR algorithm yields better outcomes. In addition, its low intrinsic variance and reliable performance are made evident by the usage of error bars that are based on 30 simulation runs. These results support statistically and empirically the use of IWD-ACAR rather than the traditional scheduling methods under modern settings where security is sensitive and performance-intensive clouds are used.

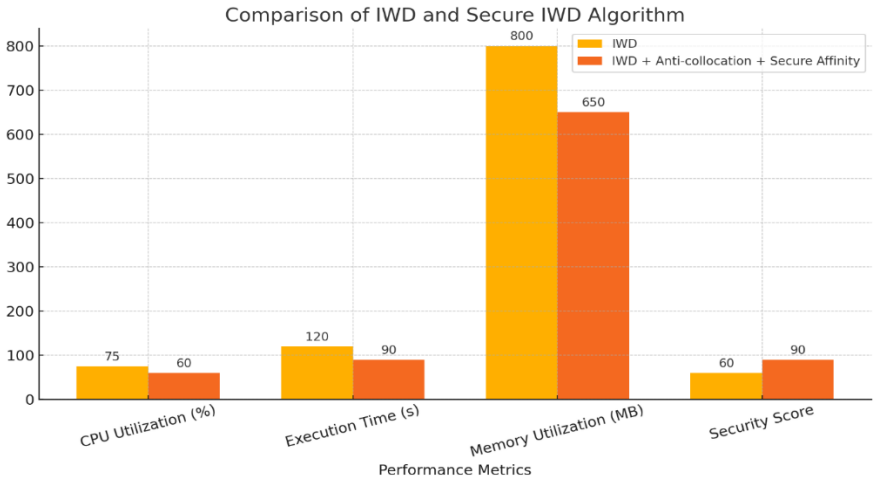

Fig. 12. Comparison of IWD algorithms with IWD-ACAR

Figure 12 shows the comparison graph that shows the enhancement in the performance of the traditional Intelligent Water Drop (IWD) algorithm by using the Anti-collocation rule and the Secure Anti-affinity rule. In the key optimal results, the secure version observes a significant decrease in the number of CPUs used, as well as the memory up in more efficient utilization of resources. They are also more cost-effective because the execution takes a shorter time, due to a good selection of the nodes that will not overload it or be susceptible to attacks. The most important fact is that the security score goes up dramatically, indicating the success of the work done

with the inclusion of security constraints into the scheduling. In general, an improved IWD algorithm is not only optimally performing but also better secures the system against side-channel attacks, counterfeit container injection, and co-location attacks.

V. Conclusion

The IWD-ACAR scheduling technique with embedded security effectively tackles key challenges in resource utilization, load balancing, security, energy efficiency, and scalability within cloud computing environments. With the integration of intelligent decision making with real-time security monitoring, the proposed algorithm significantly outperforms the existing scheduling methods. The proposed work enhances the resource utilization while minimizing the makespan. Moreover, this technique also ensures that the Container with sensitive information must be deployed on highly secure nodes. In comparison to traditional scheduling techniques like ACO, PSO, and GA, the IWD-ACAR algorithm offers a more energy-efficient and secure approach.

Conflict of Interest:

There was no relevant conflict of interest regarding this paper.

References

I. Bachiega, Naylor G., Paulo S. L. de Souza, Sarita M. Bruschi, and Simone do R. S. de Souza. “Container-Based Performance Evaluation: A Survey and Challenges.” *2018 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, April 2018, pp. 398–403. 10.1109/IC2E.2018.00075.

II. Rathi, Sugandha, Renuka Nagpal, Gautam Srivastava, and Deepti Mehrotra. “A Multi-Objective Fitness Dependent Optimizer for Workflow Scheduling.” *Applied Soft Computing*, vol. 152, 2024, article 111247. 10.1016/j.asoc.2024.111247. jscca.uotechnology.edu.iq+7dl.acm.org+7ouci.dntb.gov.ua+7

III. Li, Jun, Peng Wang, and Yan Zhang. “A Survey on Scheduling Algorithms in Cloud Computing.” *Journal of Cloud Computing*, vol. 10, no. 1, 2021, pp. 1–20. 10.3233/MGS-220217. journals.sagepub.com+2dl.acm.org+2researchgate.net+2

IV. Jeon, Jueun, et al. "Efficient container scheduling with hybrid deep learning model for improved service reliability in cloud computing." *IEEE Access* (2024). 10.1109/ACCESS.2024.3396652

V. Tabrizchi, Hamed, and Marjan Kuchaki Rafsanjani. "A survey on security challenges in cloud computing: issues, threats, and solutions." *The journal of supercomputing* 76.12 (2020): 9493-9532. 10.1007/s11227-020-03213-1

VI. Huang, Lin, Xuefeng Li, and Zhiqiang Zhang. "Security-Enhanced Cloud Scheduling for Container-Based Environments." *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, 2023, pp. 1345 1357. 10.1145/3579856.3582835

VII. Xiong, Ke, Zhonghao Wu, and Xuzhong Jia. "DeepContainer: A Deep Learning-based Framework for Real-time Anomaly Detection in Cloud-Native Container Environments." *Journal of Advanced Computing Systems* 5.1 (2025): 1-17.DOI: 10.69987/JACS.2025.50101

VIII. Parampottupadam, Santhosh, and Arghir-Nicolae Moldovann. "Cloud-based real-time network intrusion detection using deep learning." *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 2018. 10.1109/CyberSecPODS.2018.8560674

IX. Tao, Ye, et al. "Dynamic resource allocation algorithm for container-based service computing." *2017 IEEE 13th international symposium on autonomous decentralized system (ISADS)*. IEEE, 2017. 10.1109/ISADS.2017.20

X. Ahmad, Shahnawaz, et al. "Machine learning-based intelligent security framework for secure cloud key management." *Cluster Computing* 27.5 (2024): 5953-5979. 10.1007/s10586-024-04288-8

XI. Altahat, Mohammad A., Tariq Daradkeh, and Anjali Agarwal. "Optimized encryption-integrated strategy for containers scheduling and secure migration in multi-cloud data centers." *IEEE Access* (2024). 10.1109/ACCESS.2024.3386169

XII. Muthakshi, S., and K. Mahesh. "Secure and energy-efficient task scheduling in cloud container using VMD-AOA and ECC-KDF." *Malaysian Journal of Computer Science* 37.1 (2024): 48-70. 10.22452/mjcs.vol37no1.2

XIII. Sadeghi Hesar, Alireza, Seyed Reza Kamel Tabakh, and Mahboobeh Houshmand. "Task Scheduling Using the PSO-IWD Hybrid Algorithm in Cloud Computing with Heterogeneous Resources." *Journal of Control* 15.2 (2021): 81-96. 10.52547/joc.15.2.81

XIV. Pal, Souvik, et al. "An intelligent task scheduling model for hybrid internet of things and cloud environment for big data applications." *Sustainability* 15.6 (2023): 5104. 10.3390/su15065104

XV. Mangalampalli, Sudheer M., et al. “Multi-Objective Prioritized Task Scheduler Using Improved Asynchronous Advantage Actor Critic (A3C) Algorithm in Multi-Cloud Environment.” *IEEE Access*, 2024. 10.1109/ACCESS.2024.3355092

XVI. Aron, Rajni, and Ajith Abraham. "Resource scheduling methods for cloud computing environment: The role of meta-heuristics and artificial intelligence." *Engineering Applications of Artificial Intelligence* 116 (2022): 105345. 10.1016/j.engappai.2022.105345

XVII. Yahia, Hazha Saeed, et al. "Comprehensive survey for cloud computing based nature-inspired algorithms optimization scheduling." *Asian Journal of Research in Computer Science* 8.2 (2021): 1-16. 10.9734/AJRCOS/2021/v8i230195

XVIII. Chen, Honghua, et al. "Container Scheduling Algorithms for Distributed Cloud Environments." *Processes* 12.9 (2024): 1804. 10.3390/pr12091804

XIX. Rambabu, D., and A. Govardhan. “Optimized Data Replication in Cloud Using Hybrid Optimization Approach.” *Transactions on Emerging Telecommunications Technologies*, vol. 35, no. 11, 2024, e70022. 10.1002/ett.70022