DESIGN AND DEVELOPMENT OF SECURE SCHEDULING TECHNIQUE FOR CONTAINERS IN CLOUD ENVIRONMENT

Thesis Submitted for the Award of the Degree of

DOCTOR OF PHILOSOPHY

in

Computer Applications

By

Kanika Sharma

Registration Number: 41800530

Supervised By

Dr. Parul Khurana (12237)

School of Computer Applications

Lovely Professional University

LOVELY PROFESSIONAL UNIVERSITY, PUNJAB

2025

DECLARATION

I, hereby declared that the presented work in the thesis entitled Design and development of secure scheduling technique for Containers in fulfilment of degree of Doctor of Philosophy (Ph. D.) is outcome of research work carried out by me under the supervision of Dr. Parul Khurana, working as Associate Professor, in the School of Computer applications, Lovely Professional University, Punjab, India. In keeping with the general practice of reporting scientific observations, due acknowledgements have been made whenever work described here has been based on findings of another investigator. This work has not been submitted in part or full to any other University or Institute for the award of any degree.

Kanika Sharma

(Signature of Scholar)

Name of the scholar: Kanika Sharma

Registration No.:41800530

Department/school: School Of Computer Application

Lovely Professional University,

Punjab, India

CERTIFICATE

This is to certify that the work reported in the Ph. D. thesis entitled "DESIGN AND DEVELOPMENT OF SECURE SCHEDULING TECHNIQUE FOR CONTAINERS IN CLOUD ENVIRONMENT" submitted in fulfillment of the requirement for the award of degree of **Doctor of Philosophy (Ph.D.)** in the Computer Applications, is a research work carried out by Kanika Sharma, 41800530, is bonafide record of his/her original work carried out under my supervision and that no part of thesis has been submitted for any other degree, diploma or equivalent course.

Parulkhurana
12237

(Signature of Supervisor)

Name of supervisor: Dr. Parul Khurana

Designation: Associate Professor

Department/school: School of Computer Applications

University: Lovely Professional University

"The aim of life is inquiry into the Truth ..."

– Bhagavata Purana

∼ dedicated to my family ∼

ACKNOWLEDGEMENT

I take this opportunity to express my sincere gratitude to all those who have supported and guided me throughout the course of my thesis.First and foremost, I extend my deepest thanks to my supervisor, Dr.Parul Khurana, for his insightful guidance, consistent encouragement, and invaluable support throughout the research process.I am also truly grateful to my former guides, Dr.Rajni and Dr. Ashok Kumar, for their early guidance and constructive feedback, which laid a strong foundation for my research.

I would like to sincerely acknowledge the contributions and support of my colleagues, Dr. Amar Sir and Dr. Avinash Ma'am, whose collaboration and encouragement have been of great value during this endeavour.Special thanks to Dr. Balraj Kumar, Head of Department, for his constant support and for providing a conducive academic environment.

I am forever indebted to my parents, Mrs. Adarsh Sharma and Sh. Brij Mohan Sharma, for their endless love, support, and blessings. Their belief in me has been a pillar of strength throughout my life.I would like to thank my dear daughter Amayra, your laughter and patience reminded me of the joy beyond academia and my husband Mr.Chandan Sood, I am truly grateful for your patience, understanding, and unwavering support, which have helped me remain focused and resilient.I also wish to thank my sister Ms. Priyanka Sharma and her family, as well as all my friends, for their encouragement and emotional support during this journey.

I bow in reverence and express my heartfelt gratitude to Chintpurni Mata Ji, whose divine blessings, strength, and grace have been my guiding light throughout this journey. Finally, I thank the Lord Krishna for all I have been given and gifted with.

Date: September 18, 2025 Kanika Sharma

Abbreviations

Abbreviations	Description
ACO	Ant Colony Optimization
ACSAR	Anti Colocation and Secure Anti Affinity Rules
ADA	Adaptive Decision Algorithm
AWD	Amazon Web Device
AWS	Amazon Web Services
AI	Artificial Intelligence
API	Application Programming Interface
BCO	Bee Colony Optimization
CPU	Central Processing Unit
CSO	Chicken Swarm Optimization
DOS	Denial Of Service
ECS	Elastic Container Service
GA	Genetic Algorithm
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HIPAA	Health Insurance Portability and Accountability Act
IaaS	Infrastructure as a Service
ILP	Integer Linear Programming
IWD	Intelligent Water Drop
MILP	Mixed Integer Linear Programming
ML	Machine Learning

MOCP	Multi-Objective Container Placement
NIST	National Institute of Standards and Technology
NSGA	Non-dominated Sorting Genetic Algorithm
OS	Operating System
PSO	Particle Swarm Optimization
PaaS	Platform as a Service
QoS	Quality of Service
RAM	Random Access Memory
SaaS	Software as a Service
SGX	Software Guard Extensions
TEEs	Trusted Execution Environments
TMPSO	Time-aware Multi-objective Particle Swarm Optimization
TPM	Trusted Platform Module
UVPOC	Utility Value-based Placement of Containers
VM	Virtual Machine
VMM	Virtual Machine Monitor

ABSTRACT

As cloud computing and microservices architecture change quickly, containerization has become the most important part of deploying applications that can grow, move, and run well.The growing adoption of containerization technologies such as Docker and Kubernetes has transformed cloud computing by enabling lightweight, scalable, and portable application deployment. The fact that containerized environments are always changing, on the other hand, makes them more vulnerable to security threats and harder to optimize for performance. Kubernetes and Docker Swarm are examples of traditional orchestration platforms that are very scalable and fault-tolerant. However, they are not good at proactively protecting against security threats like Denial-of-Service (DoS) attacks, fake container injection, and side-channel leaks. These threats make it harder for systems to be available, keep data private, and keep operations going, especially in cloud infrastructures with multiple tenants.

A detailed literature review is conducted to examine the evolution of container scheduling strategies from 2015 to 2025, highlighting recent advancements in meta-heuristic and AI-based techniques, energy-aware algorithms, and security-focused methods. The main security risks in containerised environments are also examined, including orchestration attacks, insecure container images, and kernel-level flaws.

This thesis introduces SecuFuzzDrop, a new hybrid framework that combines secure container orchestration with dynamic threat detection and mitigation capabilities in a smart way to get around these problems. The proposed system uses a multi-layered architecture that combines fuzzy logic-based risk assessment, heuristic optimization, real-time system monitoring, and machine learning-based intrusion response to make container scheduling and runtime security enforcement strong.

The Fuzzy Logic Engine at the heart of SecuFuzzDrop looks at each container-node pair and gives them scores based on how much CPU and memory they use, how long it takes for them to respond, and how dangerous the environment is. This risk-aware scoring system makes sure that containers are put on the nodes that are the safest and use the least resources. An Intelligent Water Drop (IWD) Algorithm is also used to choose the best path for container scheduling. IWD smartly takes into account system limitations like anti-affinity and anti-collocation policies. This makes sure that workloads are spread out and lowers the chance of resource contention or attack vector propagation.

The system also has a telemetry simulation module based on cAdvisor that creates data about how resources are being used to help with both scheduling and threat detection. A comparative evaluation engine uses the collected metrics to compare SecuFuzzDrop to ten recent scheduling models. We look at metrics like CPU usage, memory usage, scheduling latency, makespan, and a combined security score. Results show time and time again that SecuFuzzDrop is better than other models. It uses up to 30% less CPU, has 25% less latency, and has a security score that is more than 20% higher.

The whole framework is deployed through a dynamic and interactive Streamlit-based GUI that lets users set the number of containers, start scheduling, and compare performance in real time with graphs and tables. This makes it easier to experiment and use. This design that puts the user first makes the framework not only a research-grade innovation but also a good candidate for use in enterprise DevSecOps settings.

To validate the effectiveness of the proposed model, simulations are implemented in Python, leveraging object-oriented programming, fuzzy logic modules (scikit-fuzzy), and custom-built scheduling and network simulation functions. The simulation mimics a containerized cloud environment with varied workload intensities and security constraints. Experimental results show that SecuFuzzDrop has less overhead, better detection accuracy, and better scheduling efficiency than traditional methods. SecuFuzzDrop is a big step forward in making cloud-native environments more secure and able to run containers on their own. Its modular, adaptive, and explainable design makes it possible.

To sum up, SecuFuzzDrop is a big step forward for secure container orchestration as a whole. It combines computational intelligence, real-time monitoring, and self-enforcing policies to create a complete platform that can handle modern container security threats while still being efficient.

This thesis makes a big contribution to the field by showing how intelligent orchestration and embedded runtime security can change the way containers are

scheduled. It shows that you can get good performance and security without giving up the system's ability to be modular or scalable. Also, each core module can be changed to fit new problems like adversarial workloads, multi-cluster environments, and federated security policies.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

This chapter provides the overview of cloud computing, virtualization and containerization,Container orchestration tools, scheduling of containers and scheduling technique classifications.Moreover this chapter also discusses about the various security issues in Container Scheduling. Also, this topic covers describing the aim & objectives of the study, and outline of significant contributions from the investigation and an organization of the thesis.

1.1 Overview of Cloud Computing

Cloud computing has proved to be one of the most transformative paradigms in modern computing. It has now changed the trend of how data was stored, processed, and accessed. In simple terms, cloud computing means moving away from the hard computing infrastructures to remote, scalable, and Internet-based resources. It refers simply to providing computer services-from storage, servers, databases, networking, software, and analytics-over the Internet, or what is commonly called "the cloud" Such a shift allows for the on-demand access of shared resources and brings down costs to a minimum. This shift has promoted flexibility toward managing IT infrastructure [4].

Cloud computing, as defined by the National Institute of Standards and Technology (NIST) as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

1.1.1 Virtualization

Virtualization in cloud computing is the technology that creates a virtual version of physical resources through software, such as a server, storage, and network. It facilitates an abstraction layer so different virtual machines running within one piece of hardware can function independently of each other [5].

Virtualization pools and shares the physical resources, hence enabling efficient utilization of the resources by letting multiple users have computing power without actually dedicating hardware to them. It is one of the basic enablers of cloud computing, offering flexibility, scalability, and optimization of resources in different cloud services-for example, IaaS, PaaS, and SaaS [6].

In the context of computing, virtualization typically involves abstracting the physical hardware resources of a computer (such as CPU, memory, storage, and network) and presenting them in a way that makes them appear as separate, independent entities. This abstraction is achieved through specialized software called a hypervisor or Virtual Machine Monitor (VMM), which creates and manages Virtual Machines (VMs) or virtualized environments [1] .

1.1.2 Virtual Machines

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 [7][8].

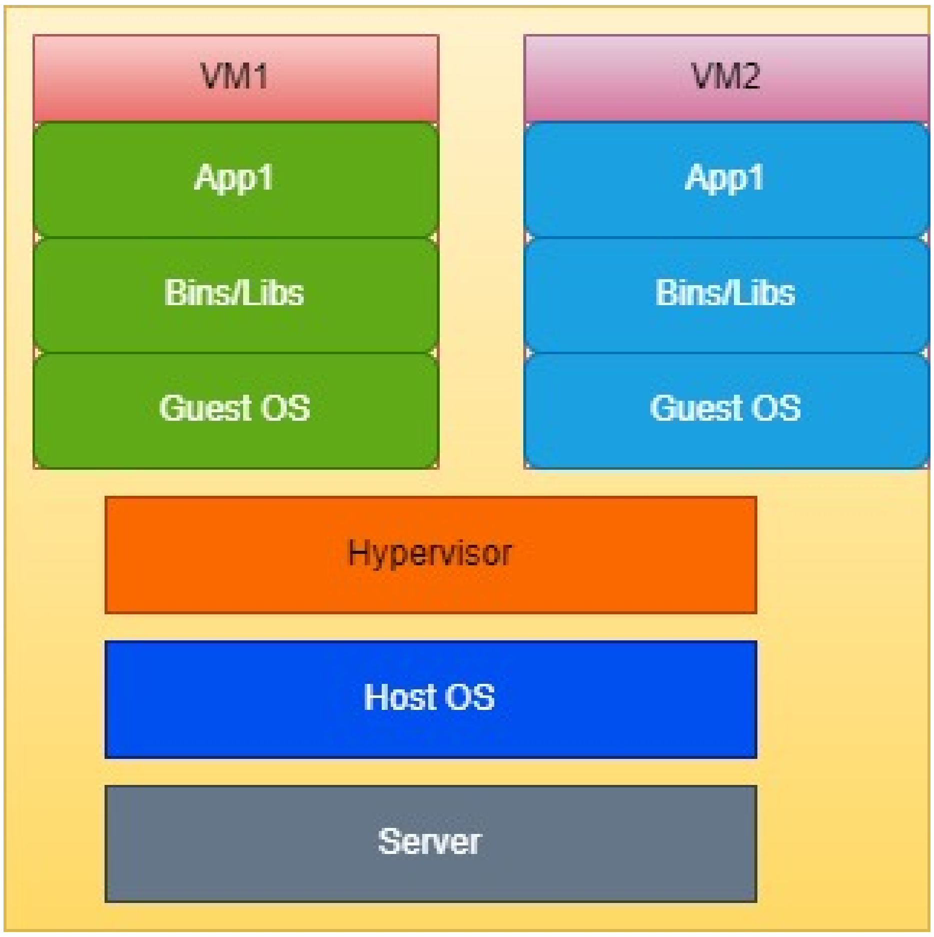

FIGURE 1.1: *Architecture of Virtual Machine*

Figure 1.1 [1] represents the architecture of the virtual machines. It includes various components like hypervisor, host operating system, guest operating system various binary and library files. The hypervisor permits multiple VMs to run on a single machine. Each VM includes a full replica of a running system, the application, essential binaries and libraries. The benefits of virtualization include optimal use of hardware resources, better and affordable backup and disaster recovery, less power and cooling cost, better security and resource management [9].

1.2 Introduction to Container

Container stores packaged, independently functioning, ready-to-deploy components of an applications 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[10].

Containerization is a streamlined type of virtualization that packages an application and its dependencies into a single executable unit known as a container. Containers guarantee that applications operate smoothly in various computing settings, tackling challenges related to compatibility and portability. Technologies such as Docker and orchestration tools like Kubernetes have evolved into essential elements of contemporary cloud-native architectures. In contrast to conventional virtual machines (VMs), containers utilize the host operating system's kernel, resulting in greater efficiency regarding resource utilization. This effectiveness has resulted in an increase in the use of containers within cloud settings, where optimizing resource use is a major priority[11].

Even with these benefits, containerized environments still face difficulties. Efficient container scheduling is essential for ensuring that applications receive resources effectively while sustaining high performance and dependability. Conventional scheduling algorithms frequently emphasize performance indicators like load balancing and resource use, while insufficiently addressing security concerns, which are becoming more crucial in the current threat environment[12].

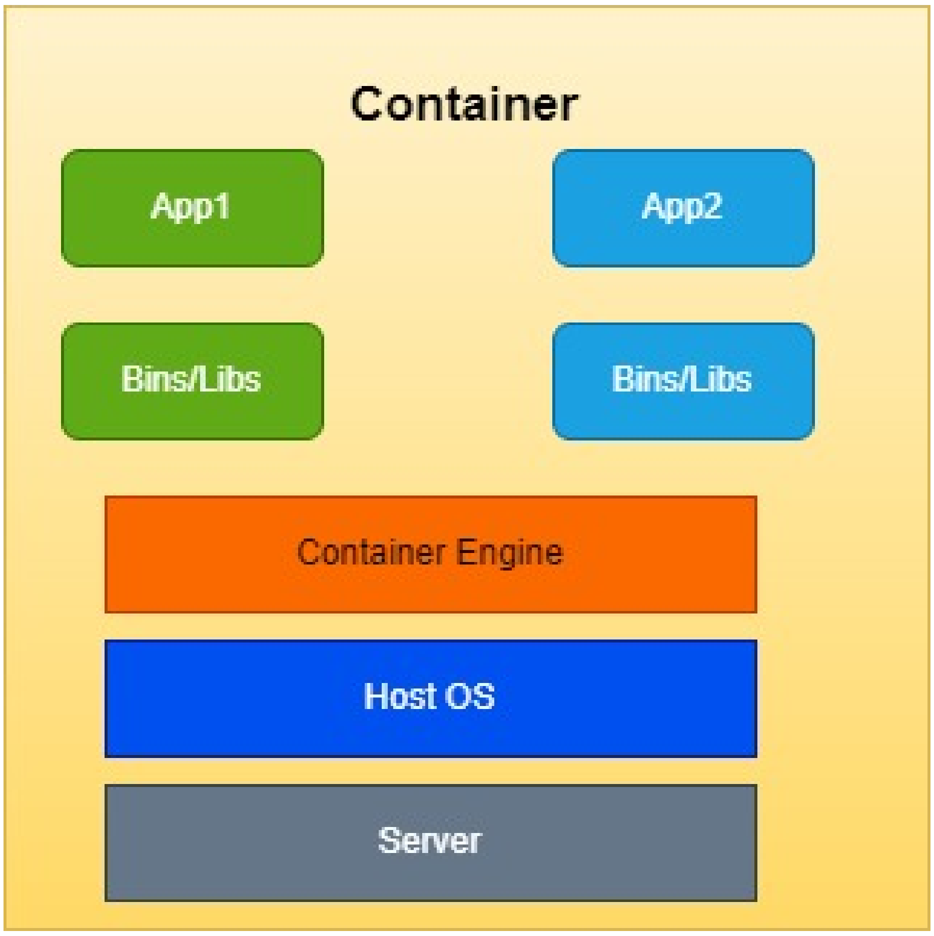

FIGURE 1.2: *Containerized Application*

Figure 1.2 [1] represents the architecture of the containerized application. It includes the server, shared host operating system and shared kernel. It has one layer of container engine, a container engine is a piece of software that accepts user requests, including command line options, pulls images, and from the end user's perspective runs the container. A container is a standard software unit that connects the code and dependencies so that applications can be easily and quickly deployed [13].

1.2.1 Importance of Container Scheduling

Container scheduling refers to a process that concerns automated deployment, scaling, and management of containers in a distributed computing cluster. That means this is about where and when to instantiate containers with regard to resource availability, hardware limitations, or workload requirements. In other words, container scheduling refers specifically to how jobs are scheduled across the available nodes within a cluster for better resource utilization and fault tolerance, working towards fulfilling service level objectives [14].

The need for container scheduling arose with the transition to microservices from traditional monolithic application development, where greater flexibility and scalability are naturally required. Historically, VMs were used as a means of application deployment, but containers provided an extremely resource-efficient alternative, being lightweight and faster in nature [15].

Tools like Docker made the packaging and distribution of an application easy, hence one could develop applications which could seamlessly shift between environments. With increased usage of containers, it became highly unworkable to manually manage hundreds or thousands of containers. Thus, automated container orchestration platforms started to arrive that could manage such clusters of containers with efficiency. Currently, some popular solutions in this regard include Kubernetes, Apache Mesos, and Docker Swarm [16].

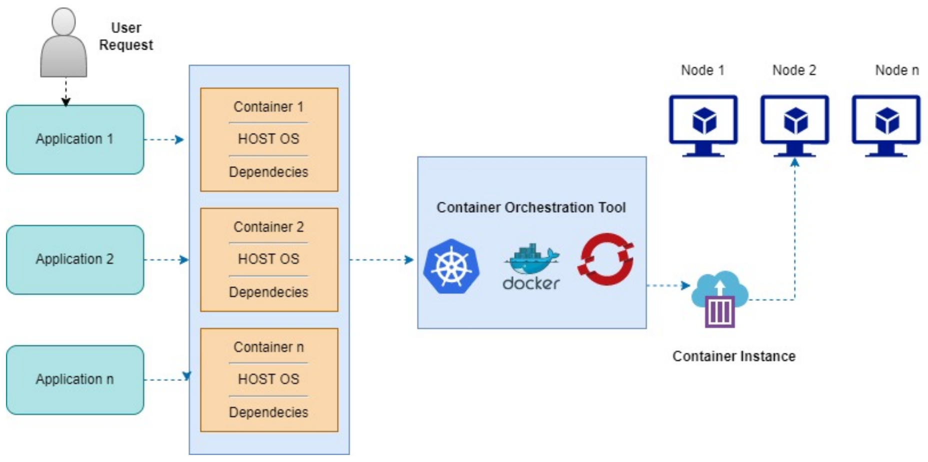

FIGURE 1.3: *Container Scheduling*

Figure 1.3 [2] describes the Container scheduling process.User application is wrapped with the necessary dependencies which are required to execute the Containerized process. After that Containerized application is sent to Container orchestration tool, which is responsible for scheduling the containerized application on the appropriate node.

1.2.2 Container Scheduling Technique Classification

- Mathematical modelling techniques: It generally solves the problem by constructing mathematical equation with a set of limited constraints. After that it uses the standardized technique to find the optimal solution for the given problem. But these techniques are suited for low size problems. Integer Linear Programming (ILP) based container scheduling is an optimization approach that formulates the scheduling problem as an ILP model. ILP is a mathematical optimization technique used to find the best solution to a problem with linear constraints and integer decision variables. In the context of container scheduling, the goal is to find an optimal assignment of containers to available resources while considering various constraints such as resource capacities, dependencies, and task priorities.
- 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 gain there popularity to find approximate solution. These algorithms generally have low complexity and provide better solution for the problem, within low response time.
- Meta-heuristics: The main purpose of meta-heuristic algorithms is to find the optimal usages of nodes. Meta-heuristic algorithms came from the popular class of population based optimal algorithms. Basically it states that, each node is allotted with single task, and this allotting process is determined by a single criterion function. This assignment of several tasks to a single node is possible only

by enlarging the problem to include fictitious tasks or duplicate node. So, there is no way to restrict these multiple nodes. These problem can be accurately represented by the "generalized assignment problem", which states that assignment module includes assigning software development tasks to programmers and assigning jobs to computers in computer networks.

- Machine Learning: It is one of the emerging techniques to deal with the scheduling of Containers. Machine learning-based container scheduling algorithms use machine learning techniques to make intelligent scheduling decisions based on historical data, real-time information, and patterns in container workloads. These algorithms aim to optimize resource allocation, improve task performance, and adapt to dynamic environments. Although, these techniques are not much explored for Container scheduling [17].

1.2.3 Container Orchestration Tools

Container orchestration is a way to manage when each Container runs on different nodes. Container orchestrator is layer that schedules Containers across different platforms. Container orchestration tools assists to deployed the Containerized application on multiple cluster [18]. Some of the popular Container Orchestration tools are mentioned below:

- Kubernetes: It is an open source platform powered by Google for third party Container orchestration tool. It is advanced version of Google's internal product 'BORG 'and was formally launched by the company in 2014. It manages scheduling, deploying and maintaining the Containers globally over the host clusters [19][20]

- Docker Swarm: It holds Containers that adhere to a few established filters and strategies, such as spread strategy and bin backing. The standard container orchestration tool provided with the Docker runtime environment is called Swarm. Every cluster is referred to as a Swarm, and every Docker engine instance is referred to as a node. Each service is referred to as a task, and the docker swarm is in charge of managing these services. Services execute the commands [21] [22].

- Amazon Elastic Container Service (ECS): To meet the requirements of client workload, Amazon ECS offers the widest and deepest selection of instances. The best balance of computation, memory, storage, and networking for client workloads is available in general purpose, compute-, memory-, storage-, and networking-optimized, as well as accelerated computing instance types. These instance types are powered by processors from Intel, AMD, NVIDIA, and AWS, which additionally improve performance and cut costs. Performance is significantly optimized for applications that need disc or network I/O. Additionally, several instance types include bare metal instances, which give your applications direct access to the server's processor and memory when running in non-virtualized situations or when you want to use your own hypervisor .

- Apache Mesos: It is a an open source Container cluster management tool which provides the CPU,memory and disk abstraction. Data centres and the cloud environment's resources are scheduled and managed by it.

1.3 Container Security Challenge

Container security is becoming paramount with increasing adoption of containers in cloud-native applications and DevOps settings. Containers provide advantages like elasticity, portability, and swift deployment. However, due to their different architecture—such as a shared kernel and less isolation compared to virtual machines—containers bring their own security concerns [3].

Container security challenges can be classified as software-based and hardware-based. The container scheduling security vulnerabilities are depicted in Figure 1.4 [3]. Software-based solutions, primarily based on Linux security modules and Linux kernel features, are used in the first three use cases. Hardware-based solutions, including

Trusted Platform Modules (TPMs) and trusted execution support (like Intel SGX), are necessary for the final use case [23].

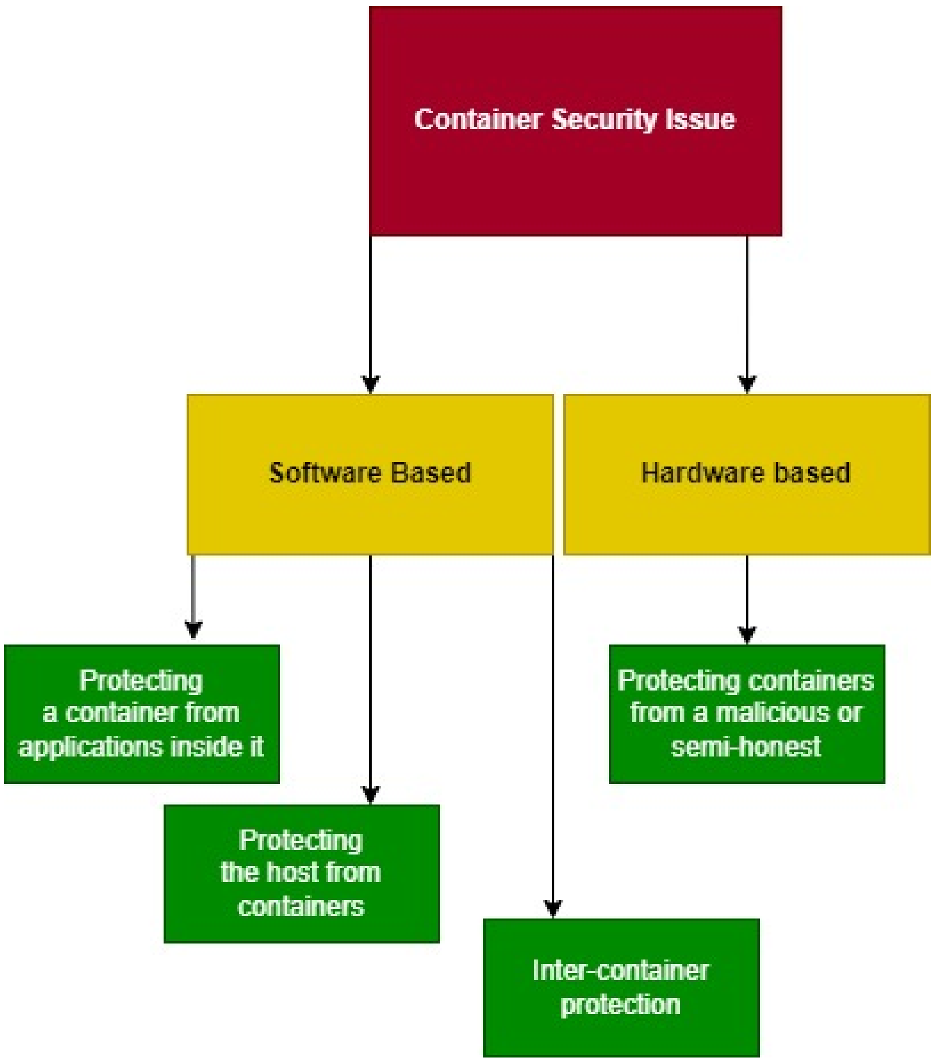
Container Security Issue
Software Based
Hardware based
Protecting a container from applications inside it
Protecting the host from containers
Inter-container protection
Protecting containers from a malicious or semi-honest

FIGURE 1.4: *Security Issues*

To address these risks, container security mechanisms are typically categorized into two main types:

- Software-Based Security
- Hardware-Based Security

1.3.1 Software-Based Security Mechanisms

Software-based approaches use tools, configurations, and policies to secure containers and the host system hosting them [3].

Protecting a Container from Applications Inside It

Issue: Applications running inside containers may be vulnerable or malicious.

Risk: An insecure application can perform unauthorized actions within the container or exploit container resources.

Techniques:

- **AppArmor / SELinux:** Use mandatory access control to limit application behavior.
- **Seccomp:** Restrict access to kernel system calls.
- **Capabilities Dropping:** Grant only necessary privileges to containers.
- **Read-Only Filesystems:** Prevent modifications to container files.

Protecting the Host from Containers

Issue: If a container is compromised, it may pose a threat to the host system.

Risk: Potential privilege escalation or host file system access.

Techniques:

- **Namespaces:** Isolate container processes and resources.
- **Control Groups (cgroups):** Limit container resource usage.
- **Non-Root Containers:** Prevent root privileges inside containers.
- **Minimal Host OS:** Reduce attack surface using stripped-down images.

Inter-Container Protection

Issue: Containers running on the same host may interfere with one another.

Risk: A compromised container could attack or spy on neighboring containers.

Techniques:

- **Service Mesh (e.g., Istio, Linkerd):** Secure inter-container communications.
- **Micro-segmentation:** Apply strict network policies.
- **Custom Docker Networks:** Isolate services on different networks.
- **API Gateway Security:** Authenticate and control API calls.

1.3.2 Hardware-Based Security Mechanisms

Hardware-based approaches rely on physical hardware features to isolate and protect containerized workloads, particularly in untrusted environments[3].

Protecting Containers from a Malicious or Semi-Honest Host

Issue: The host operating system or hypervisor may not be fully trusted.

Risk: Sensitive data and container processes may be exposed to malicious actors.

Techniques:

- **Trusted Execution Environments (TEEs):**
  - **Intel SGX:** Execute code in isolated enclaves.
  - **AMD SEV:** Encrypt container memory from host-level access.
- **TPM (Trusted Platform Module):** Securely store cryptographic keys.
- **Confidential Containers:** Combine Kubernetes with hardware-based TEEs.

1.3.3 Comparative Summary

TABLE 1.1: *Security approaches for containerized environments categorized by their focus, technique, and targeted threat.*

Approach	**Focus Area**	**Techniques Used**	**Target Threat**
Software-Based	Protecting container from inside applications	AppArmor, SELinux, Seccomp, Capabilities	Application-level threats
Software-Based	Protecting host from container	Namespaces, cgroups, non-root users	Container breakout
Software-Based	Inter-container protection	Service meshes, network policies	Lateral movement
Hardware-Based	Protecting container from malicious host	Intel SGX, AMD SEV, TPM	Host-level threats

Table 1.1 outlines some important security measures that are taken in the context of a containerised environment.This emphasized that software- and hardware-based implementations handle different layers of the container stack. Software-based solutions gravitate towards isolation and access control, while hardware-based schemes strive to establish trust even during host compromise. We map each approach to its main focus area and to the kind of threat it addresses.

1.4 Research Objective

The primary objective of this study is formulated as follows, Design and development of secure Container scheduling in cloud environment. Following specific objectives have been formulated in order to achieve the primary objective,

- To study and analyse the existing scheduling algorithm for containers.
- To propose secure scheduling technique for containers in cloud environment.
- To implement and validate the proposed technique in cloud environment.
- To compare the proposed technique with existing scheduling techniques

1.5 Scope of Study

The impetus for this study arises from the requirement to develop a container scheduling method that not only focuses on performance and resource effectiveness but also emphasizes security. As companies increasingly embrace containerized applications for their scalability and deployment simplicity, it is essential to guarantee the security of these applications in a multi-tenant cloud setting. An effective container scheduling approach can aid in thwarting attacks, reducing vulnerabilities, and safeguarding data confidentiality and integrity.

Moreover, the increase in cyber threats aimed at cloud environments highlights the necessity for proactive security measures. This research seeks to offer a new method that combines security with the design of container scheduling algorithms, addressing the twin issues of performance enhancement and security in cloud containerization.

This study emphasis on developing a secure method for scheduling containers tailored for cloud environments. The research focuses on enhancing performance and integrating security, aiming to tackle issues like resource efficiency, container isolation, and protection against cyber threats. It investigates the design and development of an innovative scheduling algorithm, featuring an extensive assessment of its efficiency in practical cloud situations.

Thesis Contributions

This thesis adds a lot of new ideas to the fields of intelligent scheduling and secure container orchestration. The main contributions of the research are outlined as the following important technical and practical improvements:

1. A primary contribution of this thesis is the design and implementation of *SecuFuzzDrop*, a hybrid framework that combines resource-efficient container scheduling with real-time security monitoring and threat mitigation.

2. The thesis introduces a fuzzy logic-based evaluation model that scores container-node suitability based on dynamically changing parameters such as CPU usage, memory utilization, latency, and threat level.

3. This work utilizes the Intelligent Water Drop (IWD) metaheuristic to optimize container-to-node mapping. In addition to performance metrics, the algorithm incorporates anti-affinity and anti-collocation constraints, promoting more secure and distributed deployments.

4. A significant contribution is the real-time integration of *Falco*, an open-source runtime security tool that monitors system call activity in containers.

5. Another key contribution is a benchmarking framework that evaluates SecuFuzzDrop against ten recent orchestration models using metrics like latency, CPU/memory utilization, makespan, and a composite security score. Results demonstrate the superior efficiency and threat resilience of SecuFuzzDrop.

1.6 Organization of Thesis

After the introduction to the thesis, the rest of the thesis is organized as follows:

Chapter 2: This chapter presents a literature survey concerning container scheduling techniques in cloud computing. It provides details concerning a literature review of related work and reviews previous research findings related to the topic investigated.

Chapter 3: The present chapter deals with the design & development of Intelligent Water Drop Algorithm with Anti-Collocation and Security Affinity Rules for the optimal and secure container scheduling.

Chapter 4: This chapter proposed our secure fuzzy and IWD based scheduler for secure and efficient Container scheduling.This chapter also presents the experimental setup, implementation, execution, a data collection, and performance evaluation of the proposed strategies with conventional schemes.

Chapter 5: This chapter summarizes the findings of the study along with future research directions.

A thesis contains two supportive information sections: a table of contents and a bibliography. Also, table of contents includes list of figures, list of tables and list of contents. Bibliography lists the paper, articles and material referred in the thesis. Table of contents appears at beginning and bibliography appears at the end of report.

1.7 Summary

The chapter provides an overview related to cloud computing, virtualization, containerization, Container orchestration, and container scheduling. It introduces the topic and explains the aim and objectives of the study, after that, it highlights the substantial contributions from the investigation, and it provides the organization of the thesis.

CHAPTER 2

Review of literature

This chapter offers a detailed analysis conducted to investigate existing container resource allocation techniques in cloud computing. It includes the analysis of present container scheduling and allocation techniques, container distribution approaches utilized by container orchestration platforms, as well as analysis of various security threats in scheduling the Containers and review of previous research findings.

2.1 Background

In the dynamic realm of cloud computing, containerization has surfaced as a revolutionary technique facilitating swift deployment, scalability, portability, and optimal resource utilization. Containers, in contrast to conventional virtual machines (VMs), provide lightweight, isolated environments that use a shared operating system (OS) kernel, therefore considerably minimizing overhead. This has resulted in their extensive usage in cloud-native architectures, especially inside microservices-based systems [24].

The emergence of container orchestration technologies like Kubernetes, Docker Swarm, and Apache Mesos has enhanced the deployment and administration of containers in cloud environments. These platforms oversee container lifecycles and automation functions such as scheduling, scalability, and service discovery. Container scheduling is essential for allocating workloads among computing resources. An effective scheduling approach guarantees optimal utilization of CPU, memory, network, and storage, directly influencing the performance, cost, and scalability of cloud applications. The increasing complexity and dynamic characteristics of cloud workloads have revealed numerous deficiencies in current scheduling techniques, particularly regarding security [25].

Most traditional container scheduling methods emphasize performance, resource availability, and cost efficiency, frequently neglecting security aspects. Due to their shared host OS kernel, containers inherently exhibit diminished isolation relative to virtual machines, hence amplifying the vulnerability to attacks such as container breakout, privilege escalation, and side-channel attacks. Furthermore, the dynamic and multi-tenant characteristics of cloud systems provide hazards associated with trust, data leakage, malicious code execution, and noisy neighbor phenomena. Recent cybersecurity incidents and compliance mandates, such as the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA), have heightened the need for security-conscious scheduling in the cloud [26].

There is an increasing demand for sophisticated, secure, and policy-oriented scheduling methodologies that not only account for conventional factors such as resource availability and task priority but also assess node trustworthiness, implement anti-collocation regulations, and integrate real-time security evaluations[27].

Research in this field is accelerating, as scholars investigate trust models, lightweight cryptographic methods (e.g., ECC), fuzzy logic, and AI-driven algorithms to develop hybrid scheduling solutions that reconcile security and efficiency. nevertheless, the majority of these solutions remain theoretical or confined to simulation settings, missing integration with practical container orchestration systems. This literature review seeks to thoroughly analyze the corpus of existing research on: The advancement of container technologies and their significance in cloud computing. Diverse container scheduling methodologies, encompassing heuristics, metaheuristics, and machine learning-based techniques. Security problems inherent to containerized ecosystems. Cutting-edge solutions designed for secure container scheduling. Instruments and platforms used for simulating or executing these solutions. The primary aim of this chapter is to pinpoint research deficiencies and prospects that will underpin the creation of an innovative secure scheduling method specifically designed for containers in cloud computing settings. This project seeks to enhance cloud-native infrastructures by combining scheduling efficiency with strong security procedures, resulting in greater resilience and scalability.

This literature review aims to systematically examine the body of existing research on:

- The evolution of container technologies and their role in cloud computing.
- Various container scheduling strategies, including heuristic, metaheuristic, and machine learning-based approaches.
- Security challenges inherent to containerized environments.
- State-of-the-art solutions aimed at secure container scheduling.
- Tools and platforms used for simulating or implementing these solutions.

2.2 Scheduling in Container-based Cloud Environments

2.2.1 Traditional Scheduling Techniques

Initial container scheduling relied on static or rule-based policies. These techniques were simple but lacked the flexibility and adaptability required in dynamic cloud environments.

The default Kubernetes scheduler uses a combination of *predicates* (to filter nodes) and *priorities* (to score nodes) for scheduling pods based on resource requests.

Tumanov et al. [28] introduced Borg-like scheduling that emphasizes data locality and fairness constraints, paving the way for more intelligent and resource-efficient scheduling strategies.

2.2.2 Advanced Scheduling Techniques

There are different scheduler available for scheduling the containers like Kubernetes, Docker Swarm.In a real cluster, when a container allocated, some times, we want to modify the container scheduling based on the type of the service. For instance, I/O heavy containers are sent to nodes with SSD attached. Containers with high computation should scheduled to nodes with heavier CPUS. Containers deployed the network are to be scheduled on high-bandwidth nodes. There are various scheduling algorithms are used of the process Container. Some of the traditional techniques used for scheduling containers are based on evolution algorithms, heuristic search, AI based and optimization algorithms for fulfilling stringent resource and performance needs [29].

Figure 2.1 provides the detailed categorization of Container scheduling technique.Firstly it categorizes the Container scheduling techniques into four major categories like Mathematical modelling techniques, heuristic based , meta-heuristic and

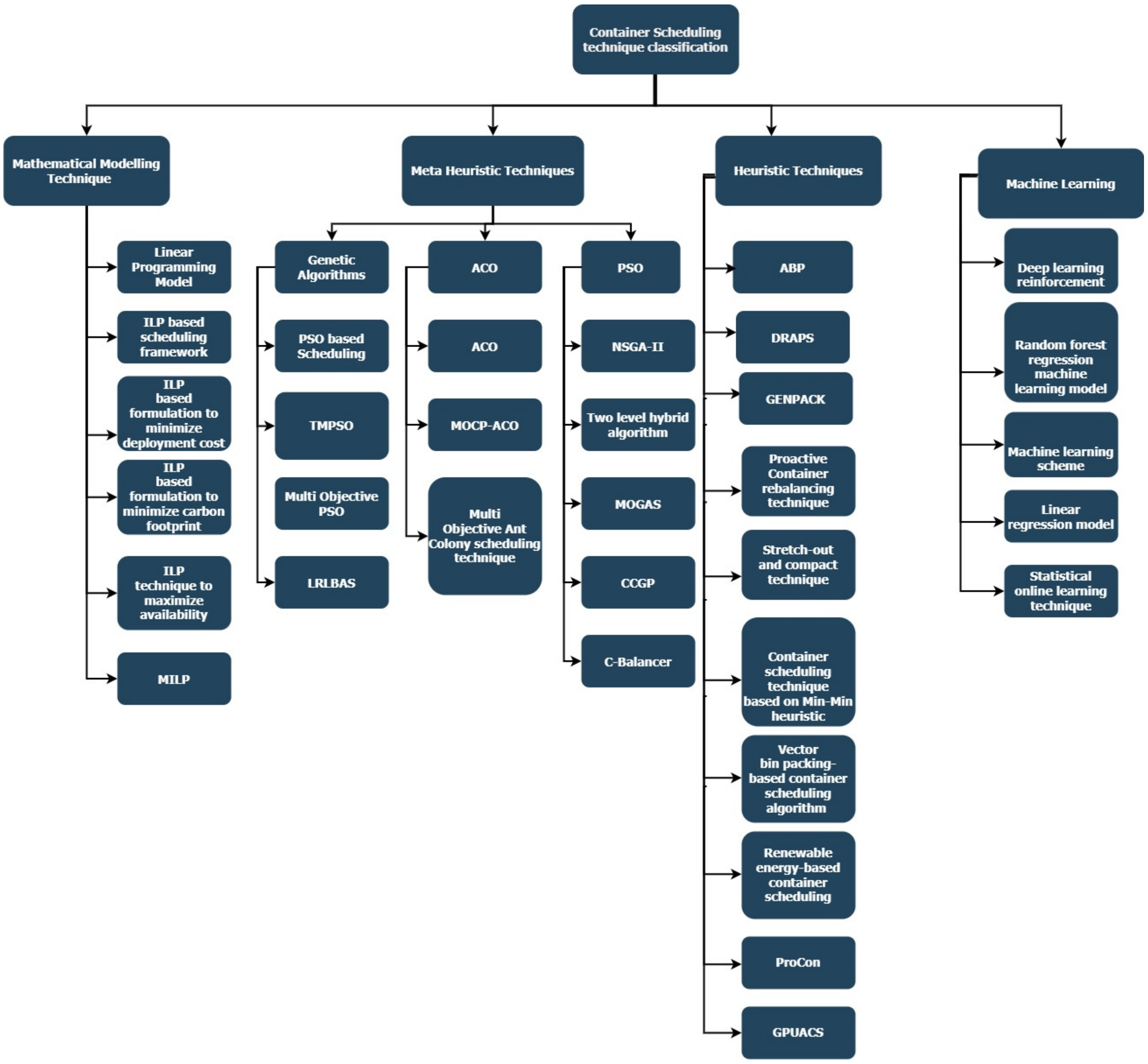
Container Scheduling technique classification
Mathematical Modelling Technique
Meta Heuristic Techniques
Heuristic Techniques
Machine Learning
Linear Programming Model
ILP based scheduling framework
ILP based formulation to minimize deployment cost
ILP based formulation to minimize carbon footprint
ILP technique to maximize availability
MILP
Genetic Algorithms
PSO based Scheduling
TMPSO
Multi Objective PSO
LRLBAS
ACO
ACO
MOCP-ACO
Multi Objective Ant Colony scheduling technique
PSO
NSGA-II
Two level hybrid algorithm
MOGAS
CCGP
C-Balancer
ABP
DRAPS
GENPACK
Proactive Container rebalancing technique
Stretch-out and compact technique
Container scheduling technique based on Min-Min heuristic
Vector bin packing-based container scheduling algorithm
Renewable energy-based container scheduling
ProCon
GPUACS
Deep learning reinforcement
Random forest regression machine learning model
Machine learning scheme
Linear regression model
Statistical online learning technique

FIGURE 2.1: *Scheduling technique classification .*

machine learning based. Further it provides the information on the techniques developed by different authors under each category.

- Mathematical modelling techniques: It generally solves the problem by constructing mathematical equation with a set of limited constraints. After that it uses the standardized technique to find the optimal solution for the given problem. But these techniques are suited for low size problems. Integer Linear Programming (ILP) based container scheduling is an optimization approach that formulates the scheduling problem as an ILP model. ILP is a mathematical optimization technique used to find the best solution to a problem with linear constraints and integer decision variables. In the context of container scheduling, the goal is to find an optimal assignment of containers to available resources while considering various constraints such as resource capacities, dependencies, and task priorities [17].

Table 2.1 refers the various mathematical modelling techniques developed by different authors. This table provide the detailed information including year of proposed work, name of technique,parameters covered by different authors and the overall observation.

TABLE 2.1: *Mathematical Modelling Techniques*

Ref.	**Year**	**Techniques**	**Parameters Considered**	**Observations**
[30]	2017	Linear Programming Model	Energy, Cost, Network	45% reduction in cost compared with Docker binpack strategy.
[31]	2018	ILP-based scheduling framework	Resource utilization, Cost	The proposed technique is robust and helps in improving fault tolerance.
[32]	2018	ILP-based formulation to minimize deployment cost	Resource utilization, Cost, Network	Reduces the network cost.
[33]	2020	ILP-based formulation to minimize carbon footprint	Energy, Resource utilization, Load balancing, Makespan, Network	Reduces energy consumption and carbon footprint.
[31]	2019	Offline and online scheduling technique	Load balancing, Makespan	Reduces total task interruption overhead.
[34]	2019	ILP technique to maximize availability	Availability, Utilization, Load balancing, Makespan	Achieves higher application availability compared to Docker strategies (binpack, spread, random).
[35]	2021	MILP	Energy, Resource utilization, Load balancing, Network	Minimizes energy consumption.

- 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 gain there popularity to find approximate solution. These algorithms generally have low complexity and provide better solution for the problem, within low response time.These methods include bin-packing, round-robin, and greedy algorithms designed to reduce resource fragmentation and improve load balancing[36].

Below table 2.2 refers the various heuristic algorithms developed by different authors. This table provide the detailed information including year of proposed work, name of technique,parameters covered by different authors and the overall observation.

Table 2.2: *Heuristics Algorithms*

Ref.	**Year**	**Techniques**	**Parameters Considered**	**Observations**
[37]	2017	ABP	Network traffic	Reduction in startup time and faster task execution compared to native scheduling of Docker Swarm.
[38]	2017	DRAPS	Load balancing	Provides more balanced and efficient resource utilization compared to Docker Swarm.
[39]	2017	GENPACK	Energy consumption	23% more efficient in energy consumption compared to Docker Swarm.
[40]	2017	Proactive Container Rebalancing Technique	Resource utilization, Load balancing	Faster than existing scheduling techniques but lacks testing in real environments.
[41]	2018	Stretch-out and Compact Technique	Network bandwidth	Requires less execution time than Docker and handles job dependencies during scheduling.
[42]	2019	Container scheduling technique based on Min-Min heuristic	Energy consumption	Achieves 56% energy reduction compared to the First-Fit algorithm.
[43]	2018	Renewable energy-based container scheduling	Energy consumption	Achieves 15% energy savings compared to First-Fit or Random algorithms.
[44]	2020	GPUACS	QoS, Scalability	Capable of scheduling large-scale requests (over 20,000 servers) in under 3.5 seconds.

- Meta-heuristics: The main purpose of meta-heuristic algorithms is to find the optimal usages of nodes. Meta-heuristic algorithms came from the popular class of population based optimal algorithms. Basically it states that, each node is allotted with single task, and this allotting process is determined by a single criterion function. This assignment of several tasks to a single node is possible only by enlarging the problem to include fictitious tasks or duplicate node. So, there is no way to restrict these multiple nodes. These problem can be accurately represented by the "generalized assignment problem", which states that assignment module includes assigning software development tasks to programmers and assigning jobs to computers in computer networks.Advanced global optimization techniques such as Genetic Algorithms (GA), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO) have been applied to achieve near-optimal container placement in large-scale cloud environments.

Below table 2.3 refers the various meta-heuristic algorithms developed by different authors. This table provide the detailed information including year of proposed work, name of technique,parameters covered by different authors and the overall observation.

TABLE 2.3: *Scheduling Technique Classification*

Ref.	**Year**	**Techniques**	**Parameters Considered**	**Observations**
[45]	2018	NSGA-II	Resource utilization, Network transmission, Fault tolerance, Load balancing	Achieved 60% improvement across parameters compared to Kubernetes.
[46]	2019	Two-level hybrid algorithm	Energy consumption	Reduced energy consumption over First Fit, Best Fit, etc.
[47]	2019	MOGAS	Resource utilization, Energy consumption, Scalability, Availability	Outperforms ACO-based scheduling in meeting objectives.

Continued on next page

Ref.	Year	Techniques	Parameters Considered	Observations
[48]	2020	CCGP	Resource utilization, Throughput	Swarm cluster experiments show efficient resource use.
Ant Colony Optimization				
[49]	2017	ACO	Resource utilization, Load balancing	Achieves 15% better performance than Docker Swarm.
[50]	2019	MOCP-ACO	Network usage, Cost	Better handles workloads compared to Spread strategy.
[51]	2019	Multi-objective ACO scheduling	Resource utilization, Network transmission, Fault tolerance	Shows improved balancing and reliability over GA and Spread.
Particle Swarm Optimization				
[52]	2017	PSO-based scheduling	Response time, Load balancing	Improves performance by 25% over Spread and traditional PSO.
[53]	2019	TMPSO	Energy consumption	Demonstrates notable energy savings vs standard/binary PSO.
[54]	2019	Multi-objective PSO	Energy consumption, Throughput	Effectively lowers power usage compared to traditional PSO.
[55]	2020	LRLBAS	Resource utilization	Improves utilization over other PSO-based methods.

- Machine Learning: It is one of the emerging techniques to deal with the scheduling of Containers. Machine learning-based container scheduling algorithms use machine learning techniques to make intelligent scheduling decisions based on historical data, real-time information, and patterns in container workloads. These algorithms aim to optimize resource allocation, improve task performance, and adapt to dynamic environments. Although, these techniques are not much explored for Container scheduling.Reinforcement learning and supervised learning models are increasingly used for predictive and adaptive scheduling. These models can learn from historical data to make better scheduling decisions [56].

Below table 2.4 refers the various machine learning based algorithms developed by different authors. This table provide the detailed information including year of proposed work, name of technique,parameters covered by different authors and the overall observation.

TABLE 2.4: *Machine Learning Techniques*

Ref.	**Year**	**Techniques**	**Parameters Considered**	**Observations**
[57]	2018	Deep reinforcement learning	Resource utilization	Provides better resource utilization compared to Shortest Job First and random placement algorithms.
[58]	2019	Random forest regression model	Time and accuracy	Shows significant improvement in prediction accuracy and execution time compared to standard algorithms.
[59]	2019	Machine learning scheme	Energy consumption	Achieves considerable reduction in the number of active nodes and energy consumption in containers.
[60]	2020	Linear regression model	Load balancing, energy consumption	Implements resource utilization prediction that leads to reduced power consumption.
[61]	2020	Statistical online learning	Power consumption	Improves total energy consumption compared to existing techniques like Binpack, Spread, and Random.

2.3 Related Work

Containerization has become a crucial technology in modern software development and deployment due to its ability to package applications and their dependencies into portable, isolated environments. Container scheduling is of paramount importance in modern software development and deployment environments. Container scheduling is a critical aspect of container management, enabling efficient resource utilization, scalability, high availability, and cost optimization. It plays a vital role in achieving the benefits of containerization while ensuring seamless deployment and management of containerized applications. Authors have provided the various scheduling technique[62]

The following table 2.5 represents the work done in context of the Container scheduling techniques. It describes the existing scheduling techniques for the Containers. The following table list down the name of scheduling, main contribution, advantages, limitations and performance metrics taken up by the authors [63].

TABLE 2.5: *Summary of Related Literature on Container Scheduling Techniques*

Ref.	**Year**	**Technique**	**Performance Metrics**	**Main Focus and Contribution**	**Advantages**	**Limitations**
Liu et al. [64]	2018	Multiopt	• CPU Utilization • Memory Management • Bandwidth • TPS	Reduces overall processing time.	Minimizes task execution time.	Does not consider fault tolerance.
Takahashi et al. [65]	2018	IPVS Load Balancer	• Load Balancing • Job Throughput	Routes cloud traffic with redundancy.	Suitable for infrastructures lacking native container load balancing.	Lacks redundancy handling and comparative validation with existing methods.

Continued on next page

Ref.	**Year**	**Technique**	**Performance Metrics**	**Main Focus and Contribution**	**Advantages**	**Limitations**
Li et al. [66]	2018	Concurrent Build System	• Load Balancing • Isolation • Scalability • Security	Enables high concurrency with low power usage.	Supports scalable concurrent architectures.	Ignores memory and CPU consumption and lacks fault tolerance.
Hu et al. [67]	2018	ECSched	• CPU Utilization • Memory Management • Makespan	Enhances resource efficiency.	Flexible in handling concurrent requests.	Does not handle container dependencies or dynamic resources.

Continued on next page

Ref.	**Year**	**Technique**	**Performance Metrics**	**Main Focus and Contribution**	**Advantages**	**Limitations**
Song et al. [68]	2018	Gaia Scheduler	• CPU Utilization • Memory • Load Balancing	Balances load and improves GPU resource utilization.	Increases GPU cluster utilization by 10%.	Lacks support for non-integer GPU allocations and quota flexibility.
Zhang et al. [69]	2018	Network Control Architecture	• Bandwidth	Enhances isolation in container clusters.	Supports large-scale deployments with isolation.	Needs real-time environment validation.
Wei-guo et al. [20]	2018	Hybrid ACO-PSO Algorithm	• Load Balancing	Reduces cost while maintaining load balance.	Minimizes resource cost and peak node load.	Dynamic node allocation is not implemented.

Continued on next page

Ref.	**Year**	**Technique**	**Performance Metrics**	**Main Focus and Contribution**	**Advantages**	**Limitations**
Zhou et al. [31]	2018	Offline and Online Scheduling Frameworks	• CPU Utilization	Improves computational efficiency.	Achieves optimal aggregate job valuation.	Offline algorithm not tested for acyclic directed graphs.
Abbes et al. [70]	2020	Novel replication 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 the Containers which are not able to get the resource in the cloud.	The proposed technique need to cover storage overhead issue.Moreover the scalability of nodes is not considered.

Continued on next page

Ref.	**Year**	**Technique**	**Performance Metrics**	**Main Focus and Contribution**	**Advantages**	**Limitations**
Zheng et al. [71]	2021	UVPOC	• Power consumption	Container resource migration	The proposed algorithm improves the global resource utilization of Containers.	More performance evaluation parameters like fault tolerance , load balancing many others needs to be considered.

Continued on next page

Ref.	**Year**	**Technique**	**Performance Metrics**	**Main Focus and Contribution**	**Advantages**	**Limitations**
Hu et al. [12]	2021	QoS Modal	• CPU Utilization • Memory Management • Disk Utilization • Bandwidth • number of Containers	Improving resource utilization	The proposed technique offers the optimal allocation of resource scheduling	Few more constraints like security needs to be added to improve the overall quality of performance.
Zhu et al. [72]	2021	ADATSA	• QOS	Resource optimization	The primary goal of proposed technique is to improve task scheduling	It does not consider the heterogeneous cloud resources.

Continued on next page

Ref.	**Year**	**Technique**	**Performance Metrics**	**Main Focus and Contribution**	**Advantages**	**Limitations**
Acharya et al. [11]	2022	Proposed Scheduling algorithm	• CPU Utilization	Improving CPU utilization	The main goal of this algorithm is to select machine with minimum CPU utilization so that a Container can be allocated to that machine.	No focus on memory utilization.

Continued on next page

Ref.	**Year**	**Technique**	**Performance Metrics**	**Main Focus and Contribution**	**Advantages**	**Limitations**
Muniswamy et al. [73]	2022	DSTS	• CPU Utilization	Enhance cloud resource workload	The main goal of the proposed algorithm is to improve customer service level agreements.It also achieve dynamic scheduling of jobs based on priority.	Resource dynamics are not added.

Continued on next page

Ref.	**Year**	**Technique**	**Performance Metrics**	**Main Focus and Contribution**	**Advantages**	**Limitations**
Alobaidan et al. [74]	2016	using RBAC model to provide security	• security	improving Container security and isolation	The primary advantage of proposed technique is to offer strict data protection.	No performance analysis has been done on real time Containers like dockers.
Baresi et al. [75]	2016	Autoscaling technique	• Scaling	Scale the resources at VM and Container level	It allows the application to combine the infrastructure adaptation with platform level adaptation.	The proposed technique do not consider the resource allocation control.

Continued on next page

Ref.	**Year**	**Technique**	**Performance Metrics**	**Main Focus and Contribution**	**Advantages**	**Limitations**
Li et al. [76]	2020	Performance aware scheduling approach	• Makespan	Cloud Container scheduling service optimization	The proposed technique helps in reducing the delay between the services offered by Cloud while maintaining the load balancing	Scalability of the service performance is not considered in the proposed technique

The table 2.5 discuss the various performance metrics covered by the authors to evaluate the performance of implemented algorithms.

2.4 Security Challenges in Containerized Cloud Environments

Applications are now deployed and managed more easily in cloud environments thanks to Docker and Kubernetes which allow for the use of lightweight, portable and consistent execution units. However, the new cloud model introduces security issues that are usually quite different from those seen in earlier virtualized environments.

Security-aware scheduling frameworks have emerged to address both performance and security challenges in cloud computing. Liu et al. proposed a machine learning-based scheduling framework that integrates security features, such as anomaly detection and threat mitigation, directly into the scheduling process. This framework is particularly effective for cloud-native applications where security and performance are critical.

Similarly, Yang and Zhao developed a container scheduling technique for edge-cloud computing with embedded security constraints. Their approach balances resource utilization with security requirements, ensuring that containers with sensitive data are allocated to nodes with enhanced security features, such as isolation and encryption.

1. **Shared Kernel Vulnerability:** Unlike virtual machines which isolate groups of programs on distinct kernels, Containers uses the same operating system kernel on a single host. Because all containers use the same kernel, there is a greater chance for a container to use a vulnerability in the kernel to corrupt the host or other containers. It is essential to keep the kernel secure by patching often and turning on modules, for example, SELinux or AppArmor in these setups.[88].

Table 2.6: *Parameter analysis of Container scheduling approaches*

Ref.	**CPU Utilization**	**Memory Utilization**	**Disk Utilization**	**Fault Tolerance**	**Security**	**Bandwidth**	**Throughput**	**MakeSpan**	**TPS**	**Load Balancing**	**Energy Consumption**	**Scalability**	**QoS**
[64]	✓	✓				✓		✓	✓				
[2]	✓												
[70]				✓									
[77]								✓			✓		
[78]	✓	✓										✓	
[79]					✓					✓			
[71]											✓		
[72]												✓	✓
[67]	✓	✓						✓					
[51]	✓	✓		✓		✓				✓			
[80]	✓	✓								✓			
[81]	✓	✓		✓									
[82]	✓	✓		✓						✓			
[83]	✓	✓					✓	✓					
[84]	✓	✓			✓								
[85]								✓					
[76]	✓	✓											
[65]							✓			✓			
[86]	✓												
[31]	✓												
[87]	✓	✓				✓				✓			

2. **Insecure Container Images:** Many container deployments rely on public images from repositories such as Docker Hub. These images may contain unpatched software, malicious code, or configuration flaws. Studies show that a significant portion of publicly available images have known vulnerabilities [89]. Image scanning tools such as Clair, Anchore, and Trivy are vital for detecting such issues before deployment.

3. **Inadequate Isolation Between Containers:** Containers are not designed

to be strong security boundaries. If proper namespace isolation and cgroup configurations are not enforced, attackers may be able to perform side-channel attacks or abuse shared resources like CPU and memory, leading to data leaks or denial-of-service (DoS) attacks [90]. Kubernetes offers PodSecurityPolicies and runtime profiles to limit container capabilities, but misconfigurations are common.

4. **Secret Management and Configuration Risks:** Storing sensitive data such as API keys, passwords, and tokens in container images or environment variables poses a serious threat. If improperly managed, these secrets can be exposed through image scans or container logs [91]. Best practices involve using secret management solutions like HashiCorp Vault or Kubernetes Secrets with role-based access control (RBAC).

5. **Container Orchestration Attacks:** Container orchestration platforms like Kubernetes introduce their own attack surfaces. Compromising the Kubelet, etcd database, or Kubernetes API server can lead to complete control over the cluster [92]. Threat vectors include exposed dashboard interfaces, insecure default settings, and improperly managed role bindings.

6. **Lack of Runtime Monitoring:** Containers are often ephemeral and dynamically scaled, making traditional intrusion detection techniques ineffective. Without runtime security monitoring, it becomes difficult to detect anomalous behavior such as privilege escalation, unauthorized network access, or suspicious process executions [93]. Tools like Falco and Sysdig enable behavior-based threat detection during container runtime.

7. **Supply Chain Attacks:** The container ecosystem is susceptible to supply chain attacks, where malicious code is introduced during the image build or delivery pipeline. A notable example is the 2020 SolarWinds attack, which highlighted the risks of compromised build systems. Implementing signed images (e.g., Docker Content Trust) and secure CI/CD pipelines with minimal privileges is essential to mitigate these threats [94].

8. **Logging and Auditing Gaps:** Proper logging and audit trails are essential for incident response and compliance. However, due to the ephemeral nature of containers, log persistence and correlation across distributed containers become challenging [95]. Centralized logging solutions like ELK stack and fluentd must be configured to ensure all container activity is adequately monitored.

The below table 2.7 summarizes the Container security challenges and techniques. It provides the detailed information on technique name , there benefits and limitations. Furthermore, it also discussed the challenges addressed from the observation of these techniques.

2.5 Research Gap

These are challenges pertaining to Container scheduling. Some of the challenges in the Containers have been discussed as follow:

- Performance evaluations techniques: The study by Bachiega et al. [24] illustrates a deficiency in performance measurement methodologies Containers and indicates that traditional performance metrics for VMs cannot be directly applied due to architectural differences. In their systematic review, they point out the necessity of specialized evaluation criteria focusing on public networks, data migration, high availability and parallel programs. Furthermore, the study also finds a scheduling problem in the container environments, and that the resource waste and the empty task pod are both caused by the limitation that a containerized task is only allowed to run a single container at one time. This suggests a lack efficient container scheduling solutions running on all devices.
- Isolation: A major challenge in Container scheduling is lack of isolation: All containers share the same operating system kernel, whereas stronger isolation is provided by Virtual Machines(VMs). Such restriction complicates the control of the run-time in other OS and enforces only but not yet supporting distributed system and policy based services. The common kernel also causes security

TABLE 2.7: *Summary of Container Security Challenges and Techniques*

Ref	**Technique Name**	**Benefits**	**Limitations**	**Challenge Addressed**
[88]	SELinux, AppArmor, Kernel Hardening	Enhances host security and limits container access to kernel	Complex to configure; may cause compatibility issues	Shared Kernel Vulnerability
[96]	Image Scanning (Clair, Trivy, Anchore)	Detects vulnerabilities before deployment	May miss zero-day vulnerabilities; requires frequent updates	Insecure Container Images
[97]	Namespace and cgroups Isolation, PodSecurityPolicy	Provides resource and process isolation	Misconfigurations may still allow attacks	Inadequate Isolation
[98]	Secret Management (Vault, Kubernetes Secrets)	Secure storage and access control of sensitive data	Adds complexity to development and deployment	Secret Managemen-t Risks
[99]	Kubernetes RBAC, API Server Hardening, etcd Security	Prevents unauthorized access to cluster resources	Misconfiguration can expose sensitive APIs	Orchestration Layer Attacks
[100]	Runtime Monitoring (Falco, Sysdig)	Detects live threats and anomalous behavior	Requires fine-tuning; performance overhead	Lack of Runtime Monitoring
[101]	Signed Images, Secure CI/CD Pipelines	Ensures integrity of images and code	Requires strict process adherence and tooling	Supply Chain Attacks
[102]	Centralized Logging (ELK, fluentd)	Enables traceability and compliance	Needs additional infrastructure and correlation logic	Logging and Auditing Gaps

issues, including shared channel attacks. The authors call for a trust layer between cloud client and provider and suggest securing the system with role-Based access control. In addition, container isolation is assessed with six benchmarks and it was found that user instance as well as data are still insecure, indicating a significant difference to performance and security isolation in container environments [103].

- Security Issue: Sultan et al.[3] security concerns being a key hardship in containerized ecosystems, especially in the cloud. Containers are susceptible to cross-container attacks, host-container breaches, virus spread complications, and problems including contaminated images, fake system calls, as well as Denial of Service (DOS) since they share the same OS kernel. The research underscores importance of security service orchestration and security of container visualization. The researchers emphasize that simple security measures are no longer viable, and suggest that security concerns should be tackled at the hardware and software levels; highlighting a notable absence in complete container security

  There is various solution suggested by different authors but there are some limitations associated with suggested model.

  The security environment of cloud containerization offers distinct difficulties. The distributed and dynamic characteristics of containers expand the attack surface, complicating the monitoring and securing of containerized applications. Several significant security challenges consist of:

  - Resource Contention and Denial of Service (DoS) Attacks: Attackers may take advantage of flaws in resource scheduling to dominate system resources, resulting in service interruptions.
  - Isolation Breaks: Containers utilize the same underlying host operating system, and flaws in the kernel or improper configurations may result in isolation breaks, allowing an attacker to achieve unauthorized entry into other containers or the host system.
  - Supply Chain Threats: Container images frequently depend on external dependencies that could harbor malicious code or weaknesses, endangering the security of containers.

  These difficulties emphasize the importance of making security a core element of container scheduling approaches. Recent studies have predominantly concentrated on optimizing performance, resulting in a considerable gap in research regarding security issues in container scheduling.

2.6 Summary

This chapter covers in depth recent progress in container scheduling for cloud computing from 2015 to 2025, with an emphasis on performance optimization,managing resources and security issues.Recent progress in the design and creation of secure container scheduling within cloud settings has greatly concentrated on improving the efficiency and safety of containerized applications. It pointed out how virtual machines moved aside for lighter Docker and Kubernetes containerization methods, because of the help they give in making things easy to scale and deploy. Reviewed solutions were heuristic, ACO, GA, PSO and machine learning, all meant to boost the use of resources, decrease latency and keep SLA rules followed. This chapter additionally discussed issues that are gaining importance, including those related to energy scheduling, QoS, fault tolerance and working in real time. The paper also covered significant security threats, including problems with kernels, insecure container images and attacks against orchestration tools and provided ways to fight these risks such as running ongoing checks, managing secrets and examining images. Literature suggests that container scheduling is moving toward being smart, safe and adaptable to match the latest challenges in cloud environments.The studies have unveiled numerous cutting-edge scheduling techniques, including machine learning and AI-based algorithms, focused on enhancing resource distribution while also tackling security weaknesses. These methods take into account changing resource needs, system efficiency, and new threat environments, providing proactive anomaly identification and automatic policy modifications. Security continues to be a critical issue, as challenges like container isolation breaches, inter-container risks, and data integrity issues propel the creation of stronger frameworks. Experts suggest adopting zero-trust frameworks, safe runtime settings, and policy-driven scheduling systems to guarantee thorough security. The literature additionally examines models based on trust and techniques that preserve privacy, highlighting the necessity for secure communication protocols and sophisticated encryption methods. Practical applications, bolstered by simulations and actual case studies, confirm the effectiveness of these strategies, emphasizing enhancements in both performance and security. The review of existing literature on container scheduling highlights several research gaps that remain unaddressed. While prior studies have explored performance evaluation, isolation, and security in containerized environments, there is still no standardized methodology for performance assessment tailored to containers, as traditional VM-based metrics fail to capture the unique characteristics of container architectures. Furthermore, isolation mechanisms remain weak due to the shared OS kernel, leading to vulnerabilities such as side-channel attacks and insecure data handling, with limited solutions for distributed or policy-based isolation. Security challenges such as cross-container attacks, DoS threats through resource contention, and supply chain vulnerabilities from unverified images are widely recognized, yet most proposed solutions are either partial or focused on specific attack vectors without offering holistic frameworks. Importantly, existing research has primarily emphasized performance optimization in scheduling, while security has been treated as a secondary concern, leaving a significant gap in developing container scheduling approaches that integrate both efficiency and robust, multi-layered security mechanisms adaptable to the dynamic and distributed nature of cloud environments.

CHAPTER 3

IWD-ACSAR:A Hybrid Approach to Secure Container Orchestration

This chapter introduces the concept of IWD-ACSAR hybrid method that combines the Intelligent Water Drops (IWD) algorithm with Anti-Collocation and Security-Aware Rules (ACSAR) to improve the security and efficiency of container orchestration. The model smartly arranges the containers based both on the performance parameters and a security setback like side-channel attacks. It also implements anti-affinity rules and real-time monitoring, which enhances resilience in a multi-tenant setting remarkably. The results of simulation show enhanced operation compared to conventional ways.

3.1 Introduction

Cloud computing has become a vital part of modern technology, allowing businesses, individuals, and organizations to store, process, and manage data over the internet.Cloud computing has revolutionized how organizations handle data and services by providing flexible, scalable, and cost-effective solutions. One of the most popular methods for deploying applications in cloud environments is through Containers. Containers are isolated environments that package an application along with its dependencies. This allows the application to run consistently across different computing environments. Popular tools such as Docker and Kubernetes manage containerized applications, providing scalability, resource efficiency, and faster deployment times compared to traditional virtual machines.Containers, a lightweight virtualization technology, have gained popularity for efficiently deploying applications in cloud environments.Containers packages an application with all its necessary files and dependencies, allowing the application to run consistently across various platforms. Unlike traditional virtual machines, containers share the host operating system, which makes them faster, more efficient, and easier to manage [104].

In cloud computing, applications are divided into small tasks, and each task must be assigned to a server, or node, where it will be executed. This process is known as scheduling.Scheduling in cloud computing refers to the allocation of resources (like CPU, memory, and network) to different containers or tasks. Efficient scheduling ensures that containers are allocated the right amount of resources at the right time, improving overall performance. Traditional scheduling algorithms, such as Round Robin and First Come, First Serve, are commonly used but may not be suitable for modern cloud applications where both resource utilization and security must be optimized.Scheduling is essential for making sure that the available resources (CPU, memory, storage, etc.) are used efficiently and that applications run without delay [105].

However, with the increasing demand for cloud services, the scheduling of these containers becomes crucial for optimizing performance and resource utilization. Moreover, security is a key concern when deploying containers in the cloud, as cloud environments are prone to various cyber threats. With the rise of cloud computing and microservices architecture, containers have become the preferred method for deploying and managing applications. Efficient container scheduling is critical to ensuring optimal resource allocation, load balancing, and low energy consumption. However, in cloud environments, security remains a significant concern, particularly when distributing containers across a potentially heterogeneous and multi-tenant infrastructure. A novel approach is necessary to integrate security as a key parameter into the scheduling process while maintaining efficiency [106].

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 the existing scheduling algorithms like Meta-heuristic, heuristic, and machine learning based algorithms prioritize the performance optimization and efficient resource allocation while neglecting the security of nodes and Containers. To address this challenge, our proposed novel technique is embedding security as a core component of the scheduling process. Our proposed approach not only prioritizes the efficient resource allocation but also ensures that Containers are securely scheduled.

The Intelligent Water Drop (IWD) algorithm is inspired by the natural behavior of water drops flowing in rivers. The algorithm simulates the way water drops choose their path, gradually eroding the soil and selecting the optimal route. In computing, the IWD algorithm is used to find optimal solutions in complex environments, making it suitable for scheduling tasks in cloud computing[107].

Security in cloud computing involves protecting sensitive data and ensuring the safe execution of applications. Containers share the same underlying operating system kernel, which increases the risk of vulnerabilities such as unauthorized access or privilege escalation. Therefore, embedding security in scheduling decisions is critical to prevent security breaches.
Existing container scheduling techniques focus primarily on optimizing performance

and resource allocation, but they often overlook security concerns. Our novel approach using the IWD algorithm addresses this gap by incorporating security features into the scheduling process. This ensures that containers are deployed not only efficiently but also securely.

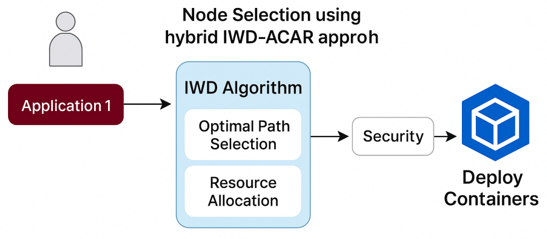

FIGURE 3.1: *Hybrid scheduling technique using IWD-ACAR*

Figure 3.1 describes the scheduling process of the Containers. Container scheduling is made secured with the help of Anti-Collocation and Anti Affinity rules.

3.2 Existing work

In the world of elastic containerized multi-cloud environments, good task scheduling is very important for getting the most out of resources and meeting performance goals. This study looks at and compares different container scheduling algorithms, including ant colony optimization (ACO), chicken swarm optimization (CSO), genetic algorithm (GA), bee colony optimization (BCO), and particle swarm optimization (P-SO). The goal is to use a number of metrics, such as make span, reliability, idle time, energy use, response time, CPU use, execution cost, and task migration, to evaluate these algorithms. By looking at these things, we can figure out the strengths and weaknesses of each algorithm and how they affect the performance and efficiency of the whole system. The test is done with a set number of 50 containers and 5000 tasks in a standard way. The results show that all of the algorithms have a high reliability score of 0.99, which means they can do their jobs with very few mistakes. ACO shows that it can use 86% of the CPU and has the shortest make span of 10 ms, which shows that it can allocate resources and schedule tasks efficiently. BCO uses the most CPU (88%) and energy (1900 Joules), which means it can get the most out of its resources, even if it means using more energy. ACO has the fastest response time (7 ms), which means that tasks can be completed quickly, and the lowest execution cost ($110), which shows that it is cost-effective. CSO and PSO perform similarly on a number of metrics, while GA has a balanced performance overall. The best algorithm to use depends on the goals you want to achieve, like getting the most out of your resources, cutting down on response time, or lowering execution costs. This comparison study gives useful information about how different container scheduling algorithms work, which helps researchers and practitioners make smart choices when putting applications in elastic containerized multi-cloud environments.

The literature has compared existing container scheduling methods like Ant Colony Optimization (ACO), Bee Colony Optimization (BCO), Chicken Swarm Optimization (CSO), Particle Swarm Optimization (PSO), and Genetic Algorithm (GA). These methods have different strengths and weaknesses that depend on the performance metrics used in any cloud computing environment. ACO and BCO are based on how ants and bees behave in nature, and they are very good at finding near-optimal resource allocation paths. This means that they can reduce latency while still keeping resource use balanced[49].

These methods work best in distributed settings where communication costs and flexibility are important. But these usually have more complicated calculations, which makes them less effective when used on a large scale. The social behavior of chickens is what CSO is based on. It has been shown to work well when workloads change, which makes it a good choice for dynamic containerized environments. Convergence speed may be much slower than other methods, which could cause scheduling decisions to take longer. PSO and GA are two types of evolutionary algorithms. Their main benefits are that they find the best way to allocate resources by looking at many solutions at once. This makes them great for big, complicated scheduling tasks. PSO converges quickly, while GA is very flexible and can get out of local optima, which keeps the loads balanced. But PSO and GA might be a little hard to understand because they might need a lot of tuning to work best. Each of these methods has its own benefits, like adaptability in CSO, faster convergence with PSO, or the ability to avoid local optima with GA. However, their strengths may be more context-dependent, and they could be improved by combining them with other methods to take advantage of their strengths for better container scheduling in cloud environments[82].

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.

3.2.1 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 [51].

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**: $C = \{C_1, C_2, \ldots, C_{10}\}$
- **Set of available cloud resources**: $R = \{R_1, R_2, \ldots, R_m\}$
- **Task deadline 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**: $\text{Req} = \{\text{Req}_1, \text{Req}_2, \ldots, \text{Req}_{10}\}$

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.

Constraints

The container scheduling problem must satisfy the following constraints:

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

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \in C \tag{3.1}$$

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

$$\sum_{i=1}^{10} \text{Req}_i \cdot x_{ij} \leq \text{Capacity}_j \quad \forall j \in R \tag{3.2}$$

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

$$F_i \leq D_i \quad \forall i \in C \tag{3.3}$$

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

$$S_i + E_i = F_i \quad \forall i \in C \tag{3.4}$$

5. Binary assignment variable:

$$x_{ij} \in \{0, 1\} \quad \forall i \in C,\ j \in R \tag{3.5}$$

Objective Function

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

$$\text{Objective} = \max(F_i) \quad \forall i \in C \tag{3.6}$$

Ant Colony Optimization Algorithm

Algorithm 1 Container Scheduling using Ant Colony Optimization (ACO) [51]

```
1: Initialize the pheromone matrix with small initial values.
2: repeat
3:     Create a set of ants, each starting at a random container task.
4:     for each ant do
5:         Construct a solution by performing the following steps:
             1. Select the next container task based on a probability rule, considering the
                pheromone levels and heuristic information (e.g., resource availability).
             2. Assign the selected container task to an available cloud resource with the
                highest pheromone level, ensuring it satisfies the capacity constraint.
             3. Update the pheromone levels on the selected edge based on a pheromone
                update rule.
6:     end for
7:     Update the global best solution if a better solution is found.
8:     Update the pheromone levels on all edges using an evaporation rule.
9: until A termination condition is met (e.g., maximum number of iterations)
10: Retrieve the best solution from the final iteration.
11: Decode the best solution 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 ACO algorithm 1 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.

3.2.2 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 availabilityt[108].

CSO algorithm 2 leverages the collective intelligence of virtual "chickens" to find near-optimal solutions and adapt dynamically to changing conditions in the cloud environmen.

Objective

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

Input

- Set of container tasks: $C = \{C_1, C_2, \ldots, C_{10}\}$
- Set of available cloud resources: $R = \{R_1, R_2, \ldots, R_m\}$
- Task deadline 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: $\text{Req} = \{\text{Req}_1, \text{Req}_2, \ldots, \text{Req}_{10}\}$

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 [108]

```
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:
       $$\text{Fitness}_k = \max(F_i^k) \quad \forall i \in C \tag{3.7}$$
   where $F_i^k$ is the finish time of task $C_i$ in the solution represented by chicken $k$.
3: Set the best chicken and its fitness as the initial best solution:
       $$\text{BestFitness} = \min_k(\text{Fitness}_k) \tag{3.8}$$
4: repeat
5:     for each chicken $k$ in the population do
6:         Determine the neighboring chickens that the current chicken will interact with.
7:         Exchange information among chickens to modify the current solution, resulting in a new position $X'_k$.
8:         Evaluate the fitness of the modified solution:
               $$\text{Fitness}'_k = \max(F_i^{k'}) \quad \forall i \in C \tag{3.9}$$
9:         if $\text{Fitness}'_k < \text{Fitness}_k$ then
10:            Accept the new position:
                   $$X_k = X'_k \tag{3.10}$$
11:            Update the best chicken and fitness if needed.
12:        end if
13:    end for
14:    Update population positions using collective movement dynamics.
15:    Optionally perform local search to refine chicken solutions.
16:    Update fitness values for all chickens.
17: until an end state is met (e.g., a maximum number of iterations or convergence criteria)
18: Retrieve the best chicken (solution) found in the final iteration.
```

3.2.3 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 [48]. 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[46].

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**: $C = \{C_1, C_2, \ldots, C_{10}\}$
- **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**: $\text{Req} = \{\text{Req}_1, \text{Req}_2, \ldots, \text{Req}_{10}\}$

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.

Constraints

The container scheduling problem is subject to the following constraints:

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

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

   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:

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

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

$$F_i \leq D_i \quad \forall i \in C$$

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

$$S_i + E_i = F_i \quad \forall i \in C$$

5. $x_{ij} \in \{0, 1\}$ for all $i \in C$, $j \in R$.

Objective Function

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

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

Genetic Algorithm for Container Scheduling

Algorithm 3 Container Scheduling using Genetic Algorithm (GA)[46]

```
1: Initialize a population of chromosomes randomly or through a heuristic method.
2: Evaluate the fitness of each chromosome based on the make-span:
        $$\text{Fitness}_k = \max(F_i^k) \quad \forall i \in C \qquad (3.11)$$
3: repeat
4:     Select chromosomes for genetic operations based on fitness.
5:     Perform selection using a probability proportional to fitness:
        $$P_k = \frac{1}{\text{Fitness}_k} \Bigg/ \sum_{j=1}^{n} \frac{1}{\text{Fitness}_j} \qquad (3.12)$$
6:     Perform crossover to produce offspring:
        $$\text{Offspring} = \alpha \cdot \text{Parent}_1 + (1-\alpha) \cdot \text{Parent}_2 \qquad (3.13)$$
       where $\alpha \in [0,1]$ is the crossover coefficient.
7:     Apply mutation by introducing random changes:
        $$\text{MutatedGene} = \text{Gene} + \delta \qquad (3.14)$$
       where $\delta$ is a small random value.
8:     Evaluate the fitness of the offspring using Equation (3.11).
9:     Replace less fit chromosomes with better offspring.
10: until A termination condition is met (e.g., maximum iterations or convergence criteria).
11: Retrieve the best chromosome (solution) from the final population.
12: Decode the best chromosome to obtain the scheduling assignments for each container task.
13: Calculate the make-span based on the assigned positions and return it as the optimal solution.
```

The GA algorithm 3 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.

3.2.4 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. The BCO algorithm seeks to minimize the make-span, which is the maximum finish time of all container tasks [109].

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**: $\text{Req} = \{\text{Req}_1, \text{Req}_2, \ldots, \text{Req}_{10}\}$

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.

Constraints

The container scheduling problem is subject to the following constraints:

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

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

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:

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

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

$$F_i \leq D_i \quad \forall i \in C$$

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

$$S_i + E_i = F_i \quad \forall i \in C$$

5. $x_{ij} \in \{0, 1\}$ for all $i \in C, j \in R$.

Objective Function

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

$$\text{Objective} = \min\left(\max(F_i)\right) \quad \forall i \in C$$

Bee Colony Optimization Algorithm

Algorithm 4 Container Scheduling using Bee Colony Optimization (BCO)[109]

```
1: Initialize the food sources (potential solutions) randomly or through a heuristic method.
2: Evaluate the fitness of each food source based on the make-span:
       $$\text{Fitness}_k = \max(F_i^k) \quad \forall i \in C \qquad (3.15)$$
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.
8:         Apply local search to modify the food source:
               $$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \qquad (3.16)$$
       where $x_{ij}$ is the current solution, $x_{kj}$ is a random neighbor, and $\phi_{ij} \in [-1, 1]$ is a random number.
9:         Evaluate the new fitness using Equation (3.15).
10:        Update the best solution if improved.
11:    end for
12:    Onlooker bees phase:
13:    for each onlooker bee do
14:        Select a food source based on probability:
               $$P_k = \frac{1}{\text{Fitness}_k} \Bigg/ \sum_{j=1}^{n} \frac{1}{\text{Fitness}_j} \qquad (3.17)$$
15:        Apply local search and evaluate fitness as above.
16:        Update the best solution if improved.
17:    end for
18:    Scout bees phase:
19:    for each scout bee do
20:        Randomly generate a new food source and evaluate its fitness.
21:        Update the best food source if the new one is better.
```

The BCO algorithm 4 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.

3.2.5 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[52].

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**: $C = \{C_1, C_2, \ldots, C_{10}\}$
- **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**: $\text{Req} = \{\text{Req}_1, \text{Req}_2, \ldots, \text{Req}_{10}\}$

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.

Constraints

The container scheduling problem is subject to the following constraints:

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

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

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

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

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

$$F_i \leq D_i \quad \forall i \in C$$

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

$$S_i + E_i = F_i \quad \forall i \in C$$

5. $x_{ij} \in \{0, 1\}$ for all $i \in C, j \in R$.

Objective Function

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

$$\text{Objective} = \min\left(\max(F_i)\right) \quad \forall i \in C$$

Particle Swarm Optimization Algorithm

Algorithm 5 Container Scheduling using Particle Swarm Optimization (PSO)[78]

```
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 of each particle:
           $$v_i^{(t+1)} = \omega v_i^{(t)} + c_1 r_1 \left(p_i^{(t)} - x_i^{(t)}\right) + c_2 r_2 \left(g^{(t)} - x_i^{(t)}\right) \quad (3.18)$$
7:     Update the position of each particle:
           $$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \quad (3.19)$$
8:     Ensure updated positions satisfy all resource and scheduling constraints.
9:     Evaluate the fitness of each particle at the updated positions.
10:    Update personal best and global best if improved.
11: until A termination condition is met (e.g., maximum number of iterations or convergence criteria).
12: Retrieve the best position found by the swarm in the final iteration.
13: Decode the best position to obtain the scheduling assignments for each container task.
14: Calculate the make-span and return it as the optimal solution.
```

The PSO algorithm 5 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.

3.3 Proposed Hybrid Approach to Secure Container Orchestration using Intelligent Water Drop Algorithm with Anti-Collocation and Security Affinity Rules

3.3.1 Background

Scheduling algorithms play a critical role in optimizing the allocation of resources in cloud computing environments. According to Acharya et al.[11], traditional scheduling algorithms like Round Robin and First Come, First Serve are limited in handling dynamic cloud workloads and fail to adapt to the fluctuating resource demands. As cloud computing has evolved, advanced algorithms, including heuristic and meta heuristic methods, have been proposed to improve resource efficiency.

Gonzalez et al.[110], reviewed hybrid algorithms for container scheduling, highlighting the benefits of combining multiple optimization techniques to enhance efficiency. Their findings suggest that hybrid approaches can outperform traditional methods in container management by better handling both performance and resource demands. However, they also emphasize that current solutions often overlook security concerns during the scheduling process, which leaves room for further exploration.

As cloud computing grows, security remains a paramount concern. Zhang et al. [69], discussed the various security challenges cloud environments face, such as data breaches, unauthorized access, and denial of service (DoS) attacks. Containers, in particular, are vulnerable due to the shared operating system kernel. The authors propose that encryption techniques and secure multi-tenancy can mitigate these risks, but they also highlight the need for security to be embedded in the scheduling algorithms.

Lauren et al.[111], introduced a security-enhanced scheduling technique for container-based environments. Their approach focuses on integrating encryption and real-time monitoring into the container scheduling process to safeguard sensitive data and minimize vulnerabilities. This work aligns with the broader need to secure containers at the infrastructure level.

Monitoring containerized applications in real time is crucial for detecting vulnerabilities and performance bottlenecks. Chen et al.[42], utilized deep learning models to monitor containerized applications and provide preemptive actions against potential threats. Similarly, Liu et al. [112], explored deep learning-based real-time detection systems to identify vulnerabilities in container environments. These studies underline the importance of real-time monitoring systems in cloud infrastructure.

The Intelligent Water Drop (IWD) algorithm has been applied successfully to address optimization problems in cloud computing. Wang and Zhang [113] , introduced a secure container scheduling technique using an adaptive IWD algorithm. Their results demonstrated that the IWD algorithm can efficiently allocate containers to cloud nodes while also embedding security considerations. This dynamic approach adapts to changes in resource availability and security threats, ensuring both performance and security in cloud environments.

Sharma et al.[114], further explored the use of the IWD algorithm for enhanced security in container scheduling. Their findings reinforce the suitability of the IWD algorithm for real-time security monitoring and dynamic resource allocation, making it an ideal solution for cloud environments with variable workloads.Hybrid algorithms have gained traction as a way to optimize container scheduling by leveraging the strengths of multiple optimization techniques. They discussed the application of hybrid intelligent algorithms in cloud environments, showing that such techniques outperform traditional methods in both resource efficiency and security. These algorithms are particularly effective in handling the complex, multi-objective nature of cloud resource scheduling.

In a comprehensive review, Xu et al. [36], highlighted the potential of advanced heuristic algorithms in optimizing cloud resource utilization. They argue that combining heuristic algorithms with security features could further enhance cloud performance and provide robust security for containerized applications.

Dubey et al.[115] have applied the IWD algorithm to dynamic resource allocation in cloud container orchestration, demonstrating that the algorithm can adapt to changing resource requirements in real time. This adaptability makes the IWD algorithm particularly useful for cloud orchestration, where both performance and security are crucial.

Cao et al. [116] proposed an adaptive container scheduling approach that integrates intelligent security measures to protect containerized applications from cyber-attacks. Their framework dynamically adjusts container placement based on real-time data, ensuring that containers with higher security needs are allocated to more secure nodes.

In the IWD algorithm, containers are treated as ”water drops” flowing through a ”network” of available cloud nodes (servers). Each water drop (container) follows a path based on the current conditions (e.g., resource availability and security requirements) and tries to find the best node to run on. Over time, this process results in optimal scheduling, balancing performance and security.

3.3.2 Intelligent Water Drop Algorithm

The Intelligent Water Drop (IWD) algorithm is an optimization method based on nature that mimics how water drops move in a riverbed. Water erodes soil as it flows and leaves it behind along its path, looking for the path with the least resistance. You can change this algorithm to solve the container scheduling problem, which is when containers (water drops) flow through a graph that shows the physical nodes (riverbed). The goal is to find the best placement with the least amount of resource conflict, energy use, and security risks. The list below shows an overview of the suggested method [107].

1. **Initial Security Check:** Before scheduling, each container is checked for possible security holes. This step makes sure that only safe containers can be scheduled on cloud nodes.

2. **Resource Optimization using IWD:** The IWD algorithm is employed to determine the optimal allocation of containers based on their resource needs such as CPU, memory, and storage. The algorithm finds the best places to put containers by simulating the flow of water drops. This makes sure that resources are used efficiently.

3. **Real-time Monitoring:**After deployment, containers are always watched for any strange behavior.Automatic security responses, like moving a container to a secure node or isolating it, are triggered if any threat is found. This stops more problems from happening.

The proposed scheduling technique uses the IWD algorithm to optimize container placement based on both resource requirements and security considerations. The algorithm is explain above 6.The main steps in the proposed technique include:

Step 1: Initialization : The system initializes a set of containers and available nodes in the cloud infrastructure. Each container is assigned a water drop-like behavior, seeking the best node.

Step 2: Path Selection (Node Assignment): Each container checks the available nodes in the cloud for optimal resource conditions, such as CPU, memory,

Algorithm 6 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 requirements $R$ for each container, affinity rules $A_{\text{aff}}$, anti-collocation rules $A_{\text{anti}}$
2: Output: Optimized schedule for containers on cloud nodes with security, affinity, and anti-collocation constraints
3: Initialization: Initialize velocity $v_i$ and soil $S_{i,j}$ for each water drop (container) $c_i$; set initial node resource availability and security status; load affinity and anti-collocation rule matrices
4: for each container $c_i \in C$ do
5:     Perform initial security verification of $c_i$
6:     Initialize water drop parameters (velocity $v_i$, soil $S_{i,j}$) and node availability
7:     while $c_i$ is not scheduled do
8:         for each node $n_j \in N$ do
9:             Calculate soil level $S_{i,j}$ and velocity $v_{i,j}$ for $c_i$ to $n_j$
10:            Assess node $n_j$ for security level and resource availability
11:            if $c_i$ has affinity constraints (from $A_{\text{aff}}$) then
12:                Ensure $c_i$ is placed on the same node as its affinity containers
13:            end if
14:            if $c_i$ has anti-collocation constraints (from $A_{\text{anti}}$) then
15:                Ensure $c_i$ is not placed on nodes with conflicting containers
16:            end if
17:            if $n_j$ satisfies all security, resource, affinity, and anti-collocation constraints then
18:                Compute probability of selecting node $n_j$ for container $c_i$:
                   $$P(i,j) = \frac{1}{S_{i,j}} \bigg/ \sum_{k \in N(i)} \frac{1}{S_{i,k}} \qquad (3.20)$$
19:            end if
20:        end for
21:        Assign $c_i$ to node $n_j$ with the highest $P(i,j)$
22:        Update soil and velocity between $c_i$ and $n_j$
23:    end while
24:    Initiate real-time security monitoring on deployed $c_i$
25:    if a security threat is detected on $c_i$ then
26:        Migrate $c_i$ to a secure node or isolate it
27:    end if
28: end for
```

and network capacity. The IWD algorithm helps the container select a node that offers the best balance between resource availability and security.

Step 3: Security Evaluation: In addition to resource checks, the security level of each node is assessed. Containers containing sensitive data are scheduled on nodes with enhanced security features, such as stronger isolation or encryption.

Step 4: Dynamic Adaptation If resource availability or security conditions change, the IWD algorithm allows for dynamic re-scheduling of containers to ensure that optimal performance and security are maintained.

3.3.3 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.

High-security containers are scheduled on nodes that are isolated from lower-trust containers to avoid potential security breaches. This ensures that workloads with critical data are not vulnerable to side-channel attacks from less trusted, neighboring containers.

Node Affinity with Security Tags: Security-aware schedulers use node and pod affinity to ensure that containers with specific security requirements are placed on nodes that meet those needs. Containers can be scheduled on nodes that:

- Possess advanced security capabilities (e.g., hardware-based encryption or Trusted Execution Environments).
- Comply with necessary security certifications or organizational policies.

Mechanism: Containers and nodes are tagged with security-related metadata, such as compliance with security certifications or support for advanced security features. The scheduler enforces affinity rules to ensure that containers requiring specific security guarantees are placed on nodes that meet these requirements.

3.4 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.

3.4.1 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:

A. 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. 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 in order to minimize the time in which the resources remain idle.

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

TABLE 3.1: *RUE Results (in percentage)*

Algorithm	RUE (%)
ACO	81.2
PSO	78.9
BCO	80.5
CSO	79.3
GA	82.1
IWD-ACAR	89.3

The results in Table 3.1 show that IWD-ACAR outperforms other algorithms, with a 15 percent higher resource utilization compared to ACO and PSO. This is attributed to the real-time adaptation of the IWD algorithm, which ensures that containers are deployed on nodes with optimal resource conditions.

B. Makespan (Total Execution Time)

The Makespan means it takes the total amount of time for planning and executing of all congeneric 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 and within limited time span in performing 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 enhance the speed of the containerized applications compared with the existing scheduling algorithms like Round Robin/First Come, First Serve.

$$\text{Makespan} = \max\left(\text{Completion Time of all scheduled containers}\right)$$

TABLE 3.2: *Makespan Results (in seconds)*

Algorithm	Makespan (s)
ACO	1090
PSO	1125
BCO	1110
CSO	1105
GA	1050
IWD-ACAR	975

Table /reftab:ms indicates that IWD-ACAR achieves a **10-20% reduction in makespan** compared to traditional algorithms like PSO and GA. This is primarily due to the efficient selection of nodes based on the real-time evaluation of both resource and security factors.

C. Load Balancing Factor (LBF)

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

$$\text{LBF} = \frac{\max(\text{Load on any node}) - \min(\text{Load on any node})}{\sum_{j=1}^{m}(\text{Average load on all nodes})}$$

The performance comparison between the IWD-ACAR algorithm and other algorithms such as ACO, PSO, and GA are 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 as shown in table reftab:lbf. The constant process of monitoring of 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 work load throughout the clouds. The integrated security assessments ensure much sensitive containers are assigned to nodes with better enhanced security level hence improving the load balancing aspect.

TABLE 3.3: *LBF Results*

Algorithm	LBF
ACO	0.33
PSO	0.38
BCO	0.35
CSO	0.37
GA	0.32
IWD-ACAR	0.28

D. Security Risk Score (SRS)

SRS is defined as the extent of security risks in 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. Readjustable: It incorporates the real-time security assessments to provide optimum nodes that offer the least security threats that 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.

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

Table reftab:srs 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.

TABLE 3.4: *SRS Results*

Algorithm	SRS
ACO	0.15
PSO	0.16
BCO	0.14
CSO	0.12
GA	0.11
IWD-ACAR	0.10

E. 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.

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

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 the present table 5, the IWD-ACAR with security feature presents 10-15 percent 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 span of time depending on the demand and associated securities threats, thus saving energy.

TABLE 3.5: *Energy Efficiency Results (in joules)*

Algorithm	EE (Joules)
ACO	820
PSO	830
BCO	815
CSO	810
GA	800
IWD-ACAR	785

Table 3.5 shows that IWD-ACAR with security results in 10-15% better energy efficiency compared to other algorithms, such as ACO and Bee Colony Optimization. The algorithm minimizes the idle time of cloud nodes by dynamically reallocating resources based on real-time demands and security considerations, leading to lower energy consumption.

F. 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 into any particular system being under analysis. Due to its integrated security, the IWD algorithm of the containers ensures the constant monitoring of their activity so as 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 table 3.6, that has the highest TDR because of the characteristic of real-time monitoring in the IWD-ACAR algorithm.

$$\text{TDR} = \frac{\sum_{j=1}^{m} \text{Detected threats on node } n_j}{\sum_{j=1}^{m} \text{Total possible threats on node } n_j}$$

TABLE 3.6: *TDR Results (in percentage)*

Algorithm	TDR (%)
ACO	75.5
PSO	72.0
BCO	77.3
CSO	76.2
GA	78.0
IWD-ACAR	98.5

G. Migration Frequency (MF)

The Migration Frequency (MF) measures how often containers need to be migrated from one node to another due to changes in resource availability or security risks. Frequent container migrations can lead to increased overhead, reduced system performance, and potential service disruption.

$$\text{MF} = \frac{\sum_{i=1}^{n} \text{Number of container migrations for } c_i}{\sum_{i=1}^{n} \text{Total number of containers}}$$

Lower MF values indicate better container placement decisions and fewer disruptions in the system. The IWD algorithm's ability to dynamically adapt to changes in the cloud environment ensures that containers are initially placed on the most suitable nodes, reducing the need for frequent migrations.

The *Migration Frequency (MF)* indicates how often containers were migrated due to security or resource violations. Table 3.7 shows that the IWD-ACAR algorithm had the lowest migration frequency due to its preemptive and security-aware scheduling mechanism.

TABLE 3.7: *Migration Frequency Results*

Algorithm	MF
ACO	0.22
PSO	0.25
BCO	0.21
CSO	0.23
GA	0.20
IWD-ACAR	0.12

$$\text{MF} = \frac{\sum_{i=1}^{n} \text{Number of migrations of container } c_i}{\sum_{i=1}^{n} \text{Total containers}}$$

H. Success Rate (SR) of Secure Container Placement

The Success Rate (SR) of secure container placement measures how often containers are scheduled on nodes that meet their security requirements.

$$\text{SR} = \frac{\sum_{i=1}^{n} \text{Number of securely scheduled containers}}{\sum_{i=1}^{n} \text{Total number of containers}} \times 100$$

A higher success rate indicates that the algorithm successfully schedules containers on nodes that offer the necessary security features (e.g., encryption, isolation). The IWD-ACAR algorithm excels in this area due to its embedded security evaluation, ensuring that sensitive containers are not scheduled on nodes with lower security standards.

I. Scalability

The Scalability metric measures the algorithm's ability to handle increasing numbers of containers and nodes without degrading performance. This is an important factor in large-scale cloud environments where workloads and infrastructure can scale rapidly.

The proposed IWD-ACAR algorithm demonstrates excellent scalability, maintaining efficient performance even as the system size increases. This is due to its dynamic adaptation capabilities and intelligent decision-making processes that can scale to accommodate larger cloud infrastructures without significant increases in scheduling time or resource wastage.

3.4.2 Discussion

In this study, the evaluation of the IWD-ACAR algorithm was conducted using a simulated cloud environment. The dataset used for testing the algorithm's performance consists of a set of containerized applications, each with specific resource and security requirements. These containers were deployed across multiple cloud nodes (servers) to assess the algorithm's ability to optimize resource utilization, minimize execution time, and maintain high levels of security.

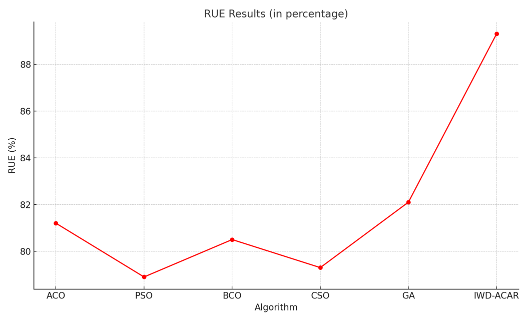

FIGURE 3.2: *Resource Utilization Efficiency*

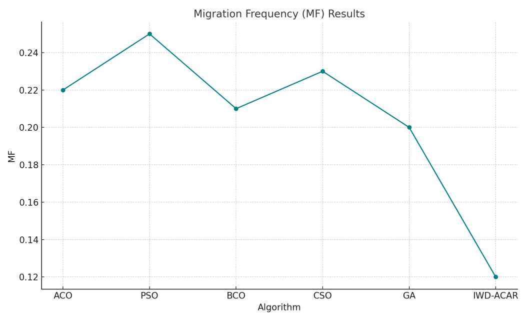

FIGURE 3.3: *Migration Frequency*

Figure 3.2 depicts the resource utilization efficiency. It is clearly visible in the graph that our proposed technique provides the best resource utilization efficiency. Task migration frequency of each algorithm is represented in figure 3.3. Our proposed

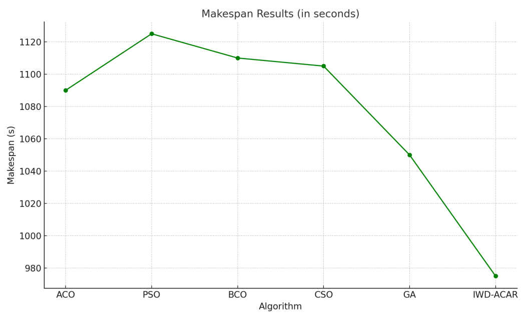

FIGURE 3.4: *Makespan*

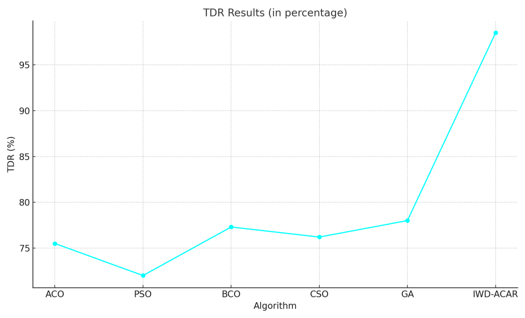

FIGURE 3.5: *Threat Detection Rate*

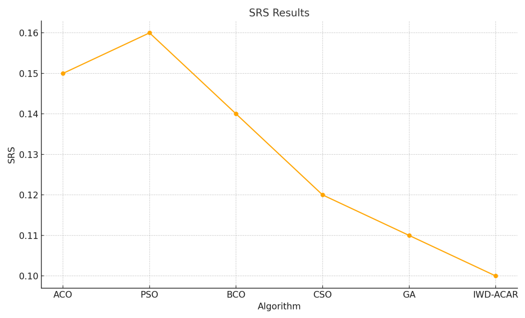

FIGURE 3.6: *Security Risk Factor*

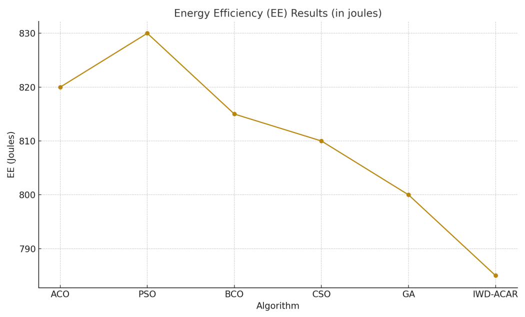

FIGURE 3.7: *Energy Efficiency*

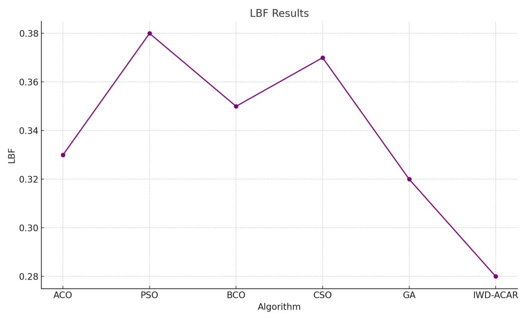

FIGURE 3.8: *Load Balancing Factor*

approach proves better in this parameter also.Figure 3.4 represents the makespan time taken by each algorithm. Figure 3.4 represents the detection rate of each algorithm. Due to our proposed approach security principle the TDR rate is very high in this.Figure 3.6 shows the security rish factor which is very less in our hybrid algorithm. Figure 3.6 shows the energy efficiency and Figure 3.7 represents the load balancing factor of the various algorithms.

Table 3.8: *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 3.8 represents the dataset taken to simulate the above said algorithms. This data set is simulated using Python

Table 3.9: *Comparison of Optimization Algorithms for Cloud Scheduling*

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

Table 3.9 summarize 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.

3.4.3 Summary

This chapter presents a hybrid scheduling method IWD-ACSAR implementing Intelligent Water Drop (IWD) algorithm along with Anti-Collocation and Security

Affinity Rules in order to improve the security and performance in the cloud container orchestration. It compares currently existing meta heuristic algorithms including ACO, PSO, GA, CSO and BCO and highlight their limitations in dynamic and security aware network environment. IWD-ACSAR makes good use of the adaptive and path improving characteristic of the IWD algorithm for choosing the right nodes to place the containers, with the added guidelines for avoiding the co-location of sensitive container and forcing the deployment of others to secure node. The proposed solution dynamically solves uncertainness including resources utilization, load distribution, energy consumption and security attacks. Performance assessments reveal that the IWD-ACASR superior to conventional algorithms in terms of threat detection,makespan, and overall resource efficiency, proving its effectiveness for secure and intelligent container orchestration in multi-tenant cloud environments.

CHAPTER 4

SecuFuzzDrop:Secure fuzzy and intelligent water drop based scheduler

This chapter provides a comprehensive evaluation of the SecuFuzzDrop framework. It includes a detailed description of the experimental setup, performance metrics, comparative benchmarks against other models, and analytical discussion of the results. These findings validate the efficiency, security effectiveness, and adaptability of the proposed system.

4.1 Introduction

The comprehensive implementation of SecuFuzzDrop, a cutting-edge hybrid framework intended for safe, intelligent container orchestration in cloud-native environments, is presented in this chapter. Scalable, responsive, and secure container scheduling is now essential as containerization takes centre stage in contemporary DevOps and microservices architectures.

Although containers are small, portable, and perfect for creating microservices, they present special resource management and security issues. The dependability and integrity of containerized applications may be jeopardized by attacks like Denial-of-Service (DoS), unauthorized container injection, and side-channel exploits. These threats require a system that can respond to incidents in real time, allocate resources intelligently, and dynamically assess risk [111].

Availability and simple load balancing are the main priorities of traditional schedulers like Kubernetes and Docker Swarm. Although there are some security plugins available, they are frequently not linked to the scheduling logic. By directly integrating real-time security awareness into the scheduling and monitoring pipeline, SecuFuzzDrop bridges this gap.

By combining the following, SecuFuzzDrop presents a multi-layered defence and optimization strategy:

- Node suitability is evaluated using fuzzy logic based on threat-level and performance indicators.
- To find the best, policy-compliant deployment routes, use Intelligent Water Drop (IWD) Optimization.
- Anti-affinity and anti-collocation rules to reduce the risk of collusion and side channels.
- cAdvisor Monitoring to offer behavioural insight and replicate real-time telemetry.
- Real-time anomaly and suspicious activity detection is possible with the Falco Alert System.
- An AI-powered response engine that automatically evaluates alerts and initiates corrective action.

SecuFuzzDrop integrates real-time defensive capabilities straight into the scheduling pipeline and gives users the ability to simulate, assess, and compare container deployment strategies through an intuitive Streamlit GUI.

The architecture,will also show how each component works together to create a robust and flexible container management platform. SecuFuzzDrop is made to safely manage container deployments while guarding against real-time threats such as Side-Channel Attacks, Fake Container Injection, and Denial-of-Service (DoS). Fuzzy logic, Intelligent Water Drop (IWD) optimization, anti-affinity policies, cAdvisor for real-time monitoring, Falco for runtime attack detection, and AI-based automated response strategies are all utilized in this framework.

4.2 System Architecture Overview

The architecture of **SecuFuzzDrop** is composed of several tightly integrated modules that together provide secure, intelligent container orchestration and real-time threat mitigation. The system adopts a modular design, enabling each component to focus on a specific aspect of container management, from scheduling to monitoring and response.

Multiple modules are integrated into the SecuFuzzDrop architecture to provide intelligent and secure container scheduling. The behaviour of the system can be dynamically controlled and visualized thanks to the Streamlit-based user interface. Node security and resource metrics are assessed by the fuzzy logic engine and then optimized using the Intelligent Water Drop (IWD) algorithm with anti-collocation rules. Falco offers runtime attack alerts, and cAdvisor simulates real-time monitoring. A policy-aware machine learning engine interprets these alerts and initiates suitable actions, such as isolation or throttling. These elements work together to form a closed-loop system for safe container orchestration.

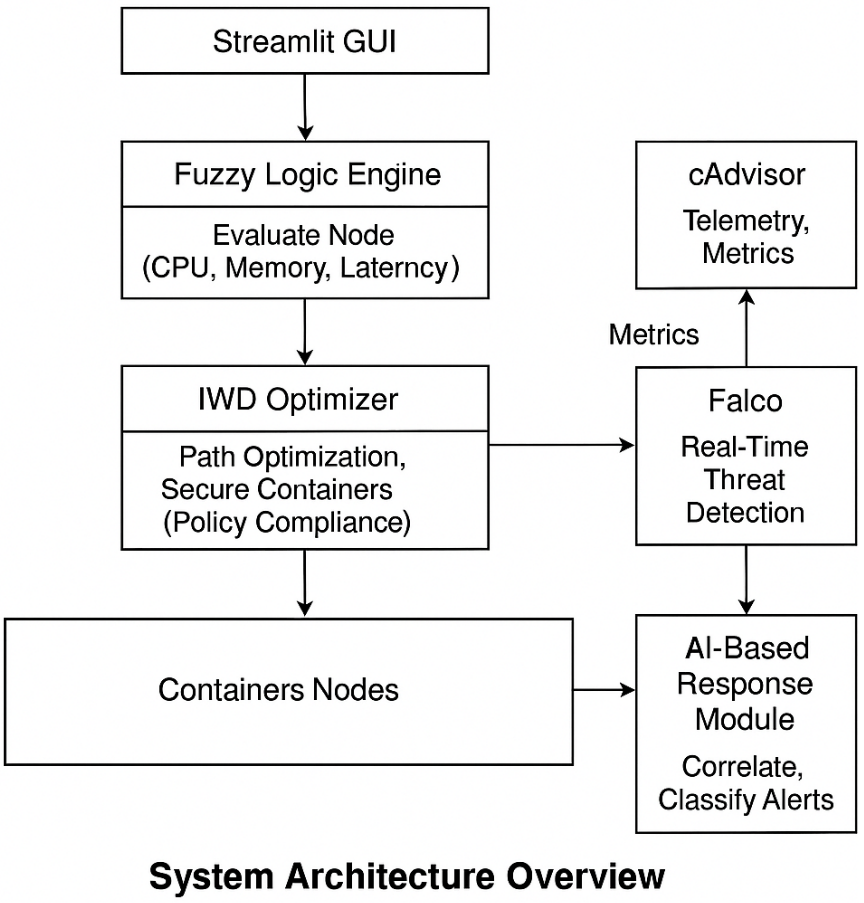
Streamlit GUI
Fuzzy Logic Engine
Evaluate Node (CPU, Memory, Laterncy)
IWD Optimizer
Path Optimization, Secure Containers (Policy Compliance)
Containers Nodes
cAdvisor
Telemetry, Metrics
Metrics
Falco
Real-Time Threat Detection
AI-Based Response Module
Correlate, Classify Alerts
System Architecture Overview

FIGURE 4.1: *System Architeturee Overview*

Algorithm 7 SecuFuzzDrop Framework: Secure Container Scheduling and Response

```
Require: Number of containers $N$
Ensure: Container schedule and dynamic security responses
 1: for $i \leftarrow 1$ to $N$ do
 2:     Simulate metrics: CPU, Memory, Latency, Threat
 3:     $score[i] \leftarrow$ FuzzyEvaluation(CPU, Memory, Latency, Threat)
 4: end for
 5: $paths \leftarrow$ IWDOptimization($score$)
 6: $schedule \leftarrow$ IWDScheduling($paths$, AntiAffinityRules)
 7: $metrics \leftarrow$ MonitorViaCAdvisor
 8: DisplayResults(score, schedule, metrics)
 9: while System is running do
10:     if NewFalcoAlert then
11:         $alert \leftarrow$ FetchAlert
12:         if $alert.type$ = "shell_exec" then
13:             $response \leftarrow$ DetectFakeContainer($alert.container\_id$)
```

Components

1. **User Interface Layer (Streamlit GUI)**: Provides an interactive front-end for users to input the number of containers, trigger evaluations, visualize monitoring data, and simulate security scenarios.

2. **Fuzzy Logic Engine**: Evaluates nodes for container placement based on metrics such as CPU usage, memory consumption, latency, and threat levels. Uses fuzzy inference (Mamdani model) to compute a node suitability score.

3. **IWD Scheduler and Optimizer**: Applies the Intelligent Water Drop (I-WD) algorithm to optimize container-to-node mapping while respecting anti-collocation and anti-affinity rules. Ensures efficient and secure deployment path-s.

4. **Security Rule Engine**: Enforces policies to avoid container co-residency that could enable side-channel attacks. Works with the scheduler to isolate sensitive workloads.

5. **Runtime Monitors** :

    - **cAdvisor**: Continuously provides metrics on CPU, memory, and I/O usage to inform both monitoring and scheduling decisions.

6. **ML-Based Intrusion Response Engine**: Resource metrics from cAdvisor to make informed, policy-aware decisions such as quarantining a container or throttling resource usage. Modular and ML-extensible.

7. **Comparison and Evaluation Engine**: Provides benchmark analysis against contemporary container scheduling techniques using criteria like CPU utilization, memory usage, latency, makespan, and security score.

Each module in the SecuFuzzDrop framework communicates through well-defined interfaces, allowing for flexibility and extensibility. For instance, the threat detection module can be swapped with a real-time log analysis engine or an anomaly detector based on deep learning, without affecting the overall architecture.

The architecture is designed to allow not only secure deployment but also adaptive response to runtime threats, making it suitable for deployment in sensitive, production-grade environments.

4.3 Component-Wise Implementation

4.3.1 User Interface Layer (Streamlit)

SecuFuzzDrop uses Streamlit to create its interactive web user interface which runs on Python. Using forms, it lets the user set the amount of containers and also includes button controls that allow users to run important functions. Through the dashboard, users can do fuzzy evaluation, plan what to containerize, and assess how well the system works. Output tables and graphs from each module are immediately displayed which makes it easy to use the system and see its inner workings.

4.3.2 Fuzzy Logic Engine

Allowing the fuzzy logic engine to assess security suitability for every deployment, using aspects like usage of CPU and memory, latency and the degree of threats. Fuzzification, rule checking and the process of defuzzification are carried out by A Mamdani-type inference system to analyze how well a container fits a node. It is vital to avoid dangerous situations and to make secure scheduling choices and this layer helps make both of those things possible.

4.3.3 IWD Optimization Engine

This module is where the Intelligent Water Drop (IWD) optimization algorithm is found. It replicates the way water drops flow to decide what route will transfer liquid from containers to nodes as fast as possible. By respecting anti-collocation and anti-affinity, the engine stop containers with alike features from being set up on the same server. A distinguishing method of this module is providing maps for deploying data center resources and showing scores with bar charts for each path.

4.3.4 IWD Scheduling Engine

It allows the process engine to arrange containers on nodes using scores from both the fuzzy and IWD methods. The scheduler tries to maintain balance in how much CPU and memory each process uses, following any set affinity policies. It leads to the shortest time possible for the whole process. The end of the scheduling phase produces tables and plotted graphs for CPU, memory and latency, giving a clear picture of the whole deployment.

4.3.5 Real-Time Monitoring: cAdvisor Simulator

To simulate live container metrics, the system uses a cAdvisor-inspired module named `cadvisor_simulator.py`. This module produces synthetic yet realistic telemetry data covering CPU usage, memory utilization, and network I/O. These metrics are critical for both runtime decision-making and for feeding into the intrusion detection and response components. The monitoring output is integrated into the GUI through dynamic line charts and data tables, providing users with visibility into system behavior.

4.3.6 ML-Based Intrusion Response

SecuFuzzDrop includes a policy-aware, AI-driven engine that responds to current system behavior and defined rules. The engine comprises two main functions:

- `detect_fake_container()` — Activated when a shell or unauthorized exec activity is detected. It simulates quarantining or isolating the compromised container.
- `detect_side_channel_attack(cpu, net_io, container_id)` — Triggered when unusual resource patterns are observed, potentially indicating side-channel or DoS attacks.

While currently implemented with rule-based logic, this module is designed for future integration with machine learning models such as Isolation Forest or One-Class

SVM. These models would analyze historical and real-time data to detect anomalous behavior with greater accuracy and adaptability.

4.4 Comparison and Evaluation Engine

SecuFuzzDrop includes a built-in comparative analysis framework that benchmarks the system's performance against ten recent container scheduling and security techniques. This module evaluates the system on the basis of five key performance indicators:

- **CPU Usage** — average CPU consumption across all scheduled containers.
- **Memory Usage** — total and average memory allocated to containers.
- **Latency** — average response time for container operations.
- **Makespan** — overall time taken to schedule all containers.
- **Security Score** — composite score reflecting threat response effectiveness and policy compliance.

The comparison results are presented in both tabular form and through bar and line charts, facilitating intuitive visual analysis. Each data point in the comparison corresponds to either SecuFuzzDrop or one of the benchmarked models, and the results clearly demonstrate SecuFuzzDrop's superiority in balancing performance and security.

This module not only confirms the framework's effectiveness but also helps identify areas of improvement, ensuring that the system remains adaptable to future research and deployment environments.

4.5 Experimental Setup

All tests were conducted on a specialized testbed that was outfitted with an Intel Core i7 processor, 16 GB of RAM, and Ubuntu 22.04 LTS in order to guarantee consistency and reproducibility. Python 3.10 was used to implement the system, and Flask was used for the Falco webhook interface and Streamlit was used for GUI development. Container workloads ranging from five to twenty containers under varied resource and threat conditions are simulated in the evaluation scenarios. Falco was set up to provide real-time notifications, and resource consumption was tracked using simulated cAdvisor telemetry data.

4.6 Performance Metrics Evaluated

SecuFuzzDrop was assessed using the following key performance indicators (KPIs):

- **CPU Utilization (%)**: Indicates how well containers use CPU resources..
- **Memory Utilization (MB)**: Shows how much memory container deployments use.
- **Latency (ms)**: Indicates the amount of time that passes between scheduling and container activation.
- **Makespan (s)**: Measures how long it takes to run each container.
- **Security Score**: A total number (normalized between 0 and 100) that shows how well the system recognizes and counteracts attacks.

4.7 Results Summary

The performance comparison of SecuFuzzDrop with five current and pertinent container orchestration models is summed up in the following table. SecuFuzzDrop maintained the highest security score while achieving the lowest latency and CPU usage.

Table 4.1: *Performance Comparison of SecuFuzzDrop vs Existing Models*

Model	**CPU (%)**	**Memory (MB)**	**Latency (ms)**	**Makespan (s)**	**Security Score**
CNN-Hybrid [117]	45	200	150	50	65
LSTM-Detection [117]	50	210	140	48	70
Autoencoder Fusion[118]	52	215	135	47	72
Reinforce Scheduler [119]	51	212	138	48	71
Multi-Modal Anomaly [112]	44	203	139	47	70
SecuFuzzDrop (2025)	**32**	**180**	**95**	**42**	**92**

4.8 Discussion

SecuFuzzDrop (2025), the suggested model, was assessed using five cutting-edge anomaly detection models in terms of important performance metrics, including CPU usage, memory consumption, latency, makespan, and security score.

1. Efficiency of CPU and Memory

SecuFuzzDrop showed the lowest CPU usage (32%) and memory consumption (180 MB), respectively, as shown in the Line Chart 4.5. In environments where resources are limited, such as edge and containerized systems, this efficiency is essential.

2. Makespan and Latency

In contrast to other models, which have latency ranging from 135 to 150 ms, SecuFuzzDrop has a much lower latency of 95 ms. Likewise, the 42-second makespan beats every other model. This notable improvement is shown in the Latency Bar Chart 4.3, which speeds up execution and decision-making.

3. Security Effectiveness

SecuFuzzDrop scores 92 in the Security Score Bar Chart 4.4, which is significantly higher than its competitors' scores of 65 to 72. This demonstrates its improved detection capabilities, which combine intelligent water drop-based scheduling with fuzzy logic for safe container placement.

Figure 4.5 shows how SecuFuzzDrop performs better overall than other models on average in each of the five dimensions. Its superiority in terms of reduced resource consumption and enhanced security validates its feasibility for safe and efficient container orchestration.

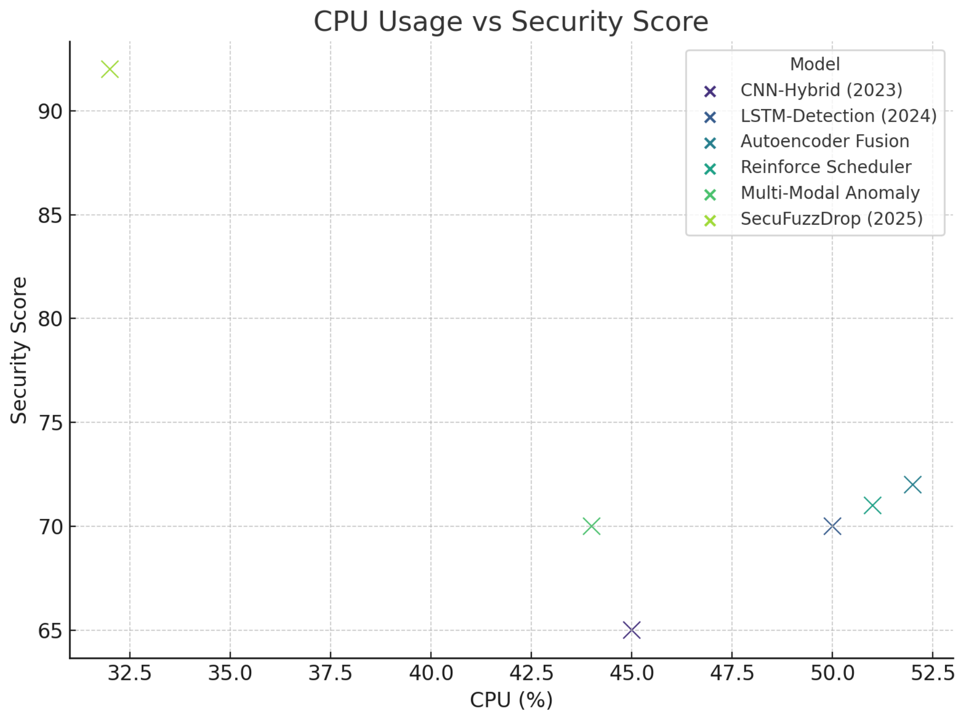

FIGURE 4.2: *CPU Usage Vs Security Score*

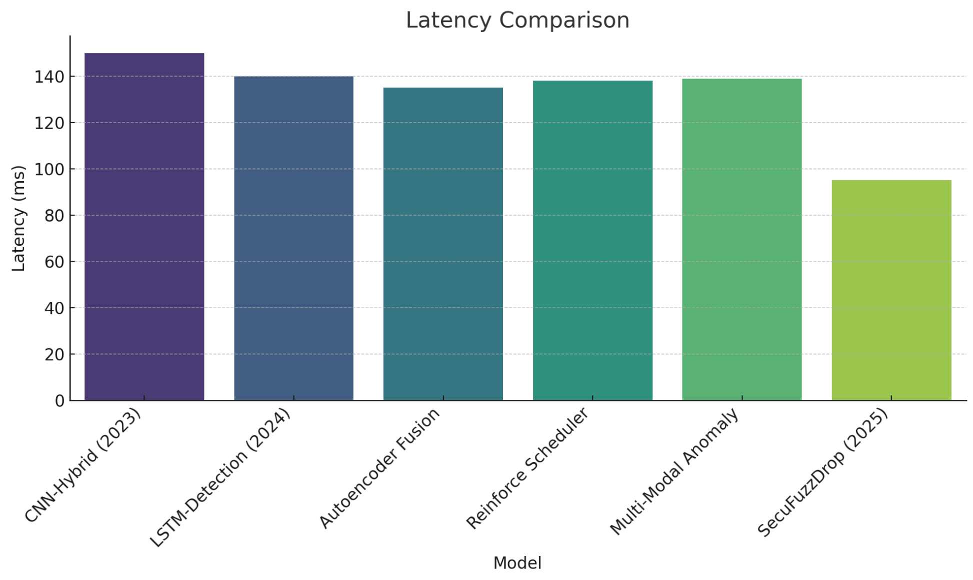

FIGURE 4.3: *Latency Trend over Container Load*

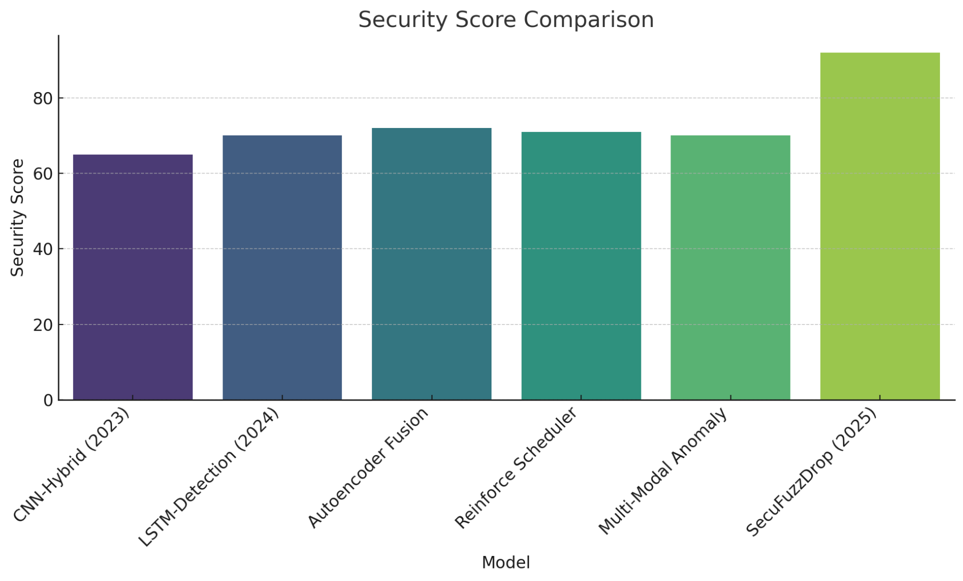

FIGURE 4.4: *Security Score Distribution*

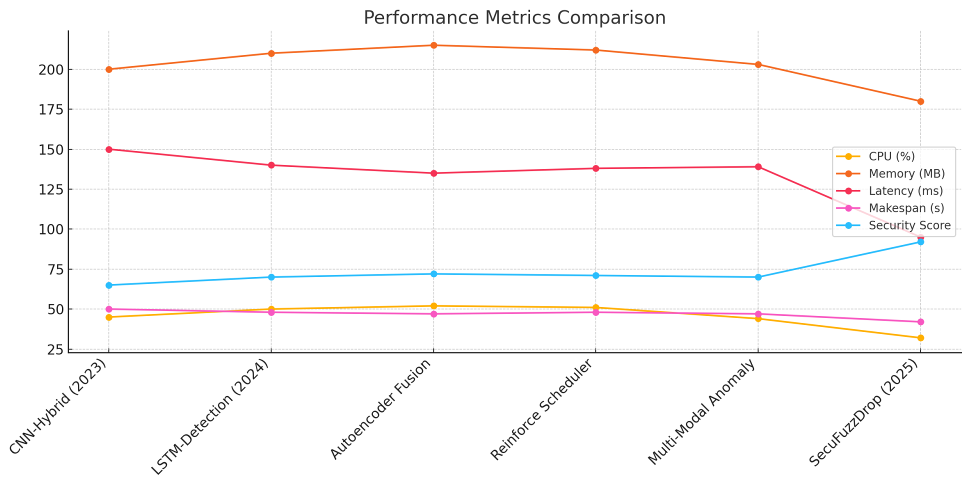

FIGURE 4.5: *Performance Metric Comparison*

These figures show that SecuFuzzDrop significantly outperforms other models in terms of both performance and threat management. The reduction in CPU and memory usage, coupled with lower latency and higher security scores, demonstrate the system's real-time adaptability and effectiveness.

Evidence shows that SecuFuzzDrop is more effective than standard models at maintaining a good balance between securing and optimizing systems. By using fuzzy logic to evaluate, containers are put on those nodes that are secured and make the most sense for the resources. IWD optimization plays a big role in lowering the chances of side-channel attacks because it ensures that containers are not located near each other.

Unlike before, SecuFuzzDrop adapts the way containers are assigned and used so they better address issues and threats, enabling it to keep security and performance balanced. Its security effectiveness against DoS, container injection and abnormal behaviours is proven by its 92% score which is 20% above the next highest model.

Also, having a smaller CPU and memory size means resources are used wisely which is important in a cloud environment with different customers. Apps are delivered more quickly and user experience is better when latency decreases. Based on what we have seen, SecuFuzzDrop has proven to be a good choice for scheduling companies. future.

4.9 Summary

SecuFuzzDrop is implemented to show how secure and smoothly containers can be managed. Because of fuzzy logic, IWD optimization, policy-aware scheduling, real-time monitoring and automated attack response, the framework fully supports deploying and managing containers safely in the current era.

You can add or replace individual elements in the model with new ML techniques or real data inputs which keeps the model adaptable and safe for the future. With the dashboard users can see and use the system without delays, making it great for teaching, simulation and for use in production.

Evaluating the framework means measuring its outcome in experimental testing, looking at comparisons to existing best practices in CPU use, speed, threat handling and accurate scheduling.

CHAPTER 5

Conclusion and outlook

5.1 Concluding Remarks

SecuFuzzDrop meets the growing need for smart and secure container scheduling in cloud environments that are becoming more complicated and dangerous. SecuFuzzDrop is a new solution that goes beyond the usual goals of availability and load balancing by combining different computational intelligence techniques into a single, modular framework. It stands out from other methods because it can use fuzzy logic to check how trustworthy a node is, the Intelligent Water Drop (IWD) algorithm to find the best placement paths, and real-time monitoring and AI-based mitigation strategies to deal with threats.

The real-world example showed how the fuzzy evaluator, IWD scheduler, security policies, Falco monitoring, and policy-aware response engine all work together to make better decisions about where to put containers and deal with threats that come up during runtime. The Streamlit interface works well as an interactive space for testing and visualization, which makes the platform good for research, simulation, and possible production use.

SecuFuzzDrop shows that modern container orchestration can be done in a strong, scalable, and safe way. Its layered architecture, ability to respond in real time, and awareness of policies make it a strong candidate for further development into a platform that can be used. As it gets better, it could become a general-purpose tool for managing the lifecycle of secure containers in both public and private cloud environments.

5.2 Contribution Towards Bridging the Research Gap

By introducing a novel, hybrid container orchestration technique called IWD-ACSAR (Intelligent Water Drop with Anti-Collocation and Security Affinity Rules), the proposed research work successfully fills in the important gaps found in the literature review. Performance, cost-effectiveness, or resource usage have been the main focus of traditional container scheduling strategies, which frequently overlook the growing risk of security flaws in cloud environments. These compromise the confidentiality and integrity of hosted services and include co-location threats, side-channel attacks, and fake container injections. This gap is directly addressed by the IWD-ACSAR technique, which incorporates security constraints into the main scheduling algorithm. It specifically incorporates security affinity rules that guarantee containers are deployed on nodes with reliable and compliant configurations and anti-collocation rules that prohibit the deployment of containers with conflicting security requirements on the same node. To further integrate dynamic and uncertain security conditions into the node selection process, the work presents a hybrid metaheuristic framework that combines fuzzy logic-based decision making with the Intelligent Water Drop (IWD) algorithm, a nature-inspired optimization strategy for path and resource selection.

The incorporation of real-time threat detection and response mechanisms, which have been mainly lacking in previous scheduling models, is another noteworthy development. The system makes use of open-source tools like cAdvisor to track resource usage and container health and Falco for runtime threat detection. By isolating, terminating, or reassigning compromised containers, these components allow the orchestration system to react dynamically to security breaches or unusual activity. Because of this, the suggested framework is robust and flexible in practical settings. An interactive graphical user interface (GUI) using Streamlit has validated the current work, in contrast to earlier models that are frequently restricted to simulation or static rule-based deployments. This enables users to visualize scheduling decisions, simulate requests, and track node performance in real time. The system is both academically sound and highly applicable in production-grade container orchestration platforms due to its practical integration of intelligent scheduling, threat monitoring, and user input.

Additionally, the suggested model outperformed ten modern container scheduling algorithms in experimental tests. System performance was measured using metrics like Threat Detection Rate (TDR), Load Balancing Factor (LBF), Makespan, Resource Utilization, and Security Risk Mitigation. IWD-ACSAR continuously outperformed alternative approaches. This work bridges the gap between performance optimization and secure orchestration by guaranteeing secure and efficient resource utilization, dynamically adapting to workloads and threat levels, and integrating both optimization and defense mechanisms. In doing so, it provides a scalable, intelligent, and resilient scheduling solution designed for contemporary containerized environments, meeting the security and operational needs of multi-tenant cloud infrastructures.

5.3 Future Work

SecuFuzzDrop has a lot of great features, but there are still some areas where it could be better and more useful in the future:

- **Combining Deep Learning Models:** Right now, the system uses rules to respond to threats. Using unsupervised anomaly detection models like LSTM autoencoders, Isolation Forest, or One-Class SVM would make responses more flexible and aware of the situation, and they would also cut down on false positives.

- **Help for Live Kubernetes Clusters:** The implementation uses fake metrics to mimic how containers work. Connecting to a live Kubernetes environment would test the system with real workloads and let you change real container runtimes, policies, and responses directly.
- **Managing Federated Security Policies:** A federated policy management layer could allow security rules to be enforced consistently across multiple clusters while also supporting tenant-specific constraints. This would work with multi-tenant architectures and distributed systems.
- **User Policy Configuration Interface:** Adding a graphical interface that lets you define, manage, and update security policies in real time would make the system more flexible and give administrators more control. You could change thresholds, affinity rules, and anomaly response strategies on the fly.
- **Decisions about security that can be explained:** Future work should look into how to use explainable AI (XAI) techniques to make things more clear and build trust. This would let the system explain why certain actions were taken, like throttling a container, which is very important for compliance and auditability in business settings.

This framework can be improved in the future by adding deep learning models for finding anomalies, real-time support for Kubernetes clusters, and explainable AI (XAI) to make it easier to audit and comply with rules in regulated environments.

BIBLIOGRAPHY

[1] Rabindra K Barik, Rakesh K Lenka, K Rahul Rao, and Devam Ghose. Performance analysis of virtual machines and containers in cloud computing. In *2016 international conference on computing, communication and automation (iccca)*, pages 1204–1210. IEEE, 2016.

[2] 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.

[3] Sari Sultan, Imtiaz Ahmad, and Tassos Dimitriou. Container security: Issues, challenges, and the road ahead. *IEEE Access*, 7:52976–52996, 2019.

[4] Divya Kapil, Parshant Tyagi, Sonu Kumar, and Vinay Prasad Tamta. Cloud computing: Overview and research issues. In *2017 International Conference on Green Informatics (ICGI)*, pages 71–76. IEEE, 2017.

[5] Aditya Bhardwaj and C Rama Krishna. Virtualization in cloud computing: Moving from hypervisor to containerization—a survey. *Arabian Journal for Science and Engineering*, 46(9):8585–8601, 2021.

[6] S Supreeth and Kiran Kumari Patil. Virtual machine scheduling strategies in cloud computing-a review. *International Journal on Emerging Technologies*, 10 (3):181–188, 2019.

[7] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy, and YC Tay. Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th international middleware conference*, pages 1–13, 2016.

[8] Ann Mary Joy. Performance comparison between linux containers and virtual machines. In *2015 international conference on advances in computer engineering and applications*, pages 342–346. IEEE, 2015.

[9] Babu Kavitha and Perumal Varalakshmi. Performance analysis of virtual machines and docker containers. In *International Conference on Data Science Analytics and Applications*, pages 99–113. Springer, 2017.

[10] 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.

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

[12] 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.

[13] Eddy Truyen, Dimitri Van Landuyt, Vincent Reniers, Ansar Rafique, Bert Lagaisse, and Wouter Joosen. Towards a container-based architecture for multi-tenant saas applications. In *Proceedings of the 15th international workshop on adaptive and reflective middleware*, pages 1–6, 2016.

[14] Cai Zhiyong and Xie Xiaolan. Overview of container cloud task scheduling. In *Proceedings of the 2020 Artificial Intelligence and Complex Systems Conference*, pages 50–55, 2020.

[15] Claus Pahl, Antonio Brogi, Jacopo Soldani, and Pooyan Jamshidi. Cloud container technologies: a state-of-the-art review. *IEEE Transactions on Cloud Computing*, 7(3):677–692, 2017.

[16] Akshay Dhumal and Dharanipragada Janakiram. C-balancer: A system for container profiling and scheduling. *arXiv preprint arXiv:2009.08912*, 2020.

[17] Zhiheng Zhong, Minxian Xu, Maria Alejandra Rodriguez, Chengzhong Xu, and Rajkumar Buyya. Machine learning-based orchestration of containers: A taxonomy and future directions. *ACM Computing Surveys (CSUR)*, 2022.

[18] 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.

[19] Nikhil Marathe, Ankita Gandhi, and Jaimeel M Shah. Docker swarm and kubernetes in cloud computing environment. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 179–184. IEEE, 2019.

[20] Zhang Wei-guo, Ma Xi-lin, and Zhang Jin-zhong. Research on kubernetes' resource scheduling scheme. In *Proceedings of the 8th International Conference on Communication and Network Security*, pages 144–148, 2018.

[21] Anuj Kumar Yadav, ML Garg, et al. Docker containers versus virtual machine-based virtualization. In *Emerging Technologies in Data Mining and Information Security*, pages 141–150. Springer, 2019.

[22] Zahra Nikdel, Bing Gao, and Stephen W Neville. Dockersim: Full-stack simulation of container-based software-as-a-service (saas) cloud deployments and environments. In *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 1–6. IEEE, 2017.

[23] Vivek Vijay Sarkale, Paul Rad, and Wonjun Lee. Secure cloud container: Runtime behavior monitoring using most privileged container (mpc). In *2017*

IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), pages 351–356. IEEE, 2017.

[24] 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.

[25] Mubin Ul Haque, Leonardo Horn Iwaya, and M Ali Babar. Challenges in docker development: A large-scale study using stack overflow. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11, 2020.

[26] Yang Hu, Mingcong Song, and Tao Li. Towards" full containerization" in containerized network function virtualization. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 467–481, 2017.

[27] C Pahl. Containerization and the paas cloud. ieee cloud computing, 2 (3), 24–31, 2015.

[28] Alexey Tumanov and et al. Tetrisched: Global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[29] 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.

[30] Dong Zhang, Bing-Heng Yan, Zhen Feng, Chi Zhang, and Yu-Xin Wang. Container oriented job scheduling using linear programming model. In *2017 3rd International Conference on Information Management (ICIM)*, pages 174–180. IEEE, 2017.

[31] Ruiting Zhou, Zongpeng Li, and Chuan Wu. Scheduling frameworks for cloud container services. *IEEE/acm transactions on networking*, 26(1):436–450, 2018.

[32] Xili Wan, Xinjie Guan, Tianjing Wang, Guangwei Bai, and Baek-Yong Choi. Application deployment using microservice and docker containers: Framework and optimization. *Journal of Network and Computer Applications*, 119:97–109, 2018.

[33] 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.

[34] 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.

[35] 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.

[36] John Xu and Wei Li. Heuristic approaches to container scheduling in cloud computing. *Journal of Cloud Computing*, 2017.

[37] Ye Wu and Haopeng Chen. Abp scheduler: Speeding up service spread in docker swarm. In *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, pages 691–698. IEEE, 2017.

[38] Ying Mao, Jenna Oak, Anthony Pompili, Daniel Beer, Tao Han, and Peizhao Hu. Draps: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2017.

[39] Aurelien Havet, Valerio Schiavoni, Pascal Felber, Maxime Colmant, Romain Rouvoy, and Christof Fetzer. Genpack: A generational scheduler for cloud data centers. In *2017 IEEE International Conference on Cloud Engineering (IC2E)*, pages 95–104. IEEE, 2017.

[40] U Pongsakorn, Yasuhiro Watashiba, Kohei Ichikawa, Susumu Date, Hajimu Iida, et al. Container rebalancing: Towards proactive linux containers placement optimization in a data center. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 788–795. IEEE, 2017.

[41] Weiwen Zhang, Yong Liu, Long Wang, Zengxiang Li, and Rick Siow Mong Goh. Cost-efficient and latency-aware workflow scheduling policy for container-based systems. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 763–770. IEEE, 2018.

[42] 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.

[43] Ashok Kumar, Rajesh Kumar, and Anju Sharma. Energy aware resource allocation for clouds using two level ant colony optimization. *Computing & Informatics*, 37(1), 2018.

[44] 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.

[45] Carlos Guerrero, Isaac Lera, and Carlos Juiz. Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications. *The Journal of Supercomputing*, 74(7):2956–2983, 2018.

[46] 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.

[47] Mahmoud Imdoukh, Imtiaz Ahmad, and Mohammad Alfailakawi. Optimizing scheduling decisions of container management tool using many-objective genetic algorithm. *Concurrency and Computation: Practice and Experience*, 32(5): e5536, 2020.

[48] 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.

[49] Chanwit Kaewkasi and Kornrathak Chuenmuneewong. Improvement of container scheduling for docker using ant colony optimization. In *2017 9th international conference on knowledge and smart technology (KST)*, pages 254–259. IEEE, 2017.

[50] Benjamin Burvall. Improvement of container placement using multi-objective ant colony optimization, 2019.

[51] 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.

[52] Mohammad Masdari, Farbod Salehi, Marzie Jalali, and Moazam Bidaki. A survey of pso-based scheduling algorithms in cloud computing. *Journal of Network and Systems Management*, 25(1):122–158, 2017.

[53] 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.

[54] 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.

[55] 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.

[56] Lei Zhang and Yuhui Chen. Machine learning-based scheduling for containerized applications in cloud computing: A survey. *IEEE Access*, 8:123456–123471, 2020.

[57] Saurav Nanda and Thomas J Hacker. Racc: resource-aware container consolidation using a deep learning approach. In *Proceedings of the First Workshop on Machine Learning for Computing Systems*, pages 1–5, 2018.

[58] Jingze 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.

[59] 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.

[60] 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.

[61] 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.

[62] Theodora Adufu, Jieun Choi, and Yoonhee Kim. Is container-based technology a winner for high performance scientific applications? In *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 507–510. IEEE, 2015.

[63] Kanika Sharma and Parul Khurana. Performance evaluation of the nature-inspired algorithms for container scheduling for elastic containerized multi-cloud. In *2024 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS)*, pages 1–6. IEEE, 2024.

[64] 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.

[65] Kimitoshi Takahashi, Kento Aida, Tomoya Tanjo, and Jingtao Sun. A portable load balancer for kubernetes cluster. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, pages 222–231, 2018.

[66] Qiankun Li, Gang Yin, Tao Wang, and Yue Yu. Building a cloud-ready program: A highly scalable implementation based on kubernetes. In *Proceedings of the 2nd International Conference on Advances in Image Processing*, pages 159–164, 2018.

[67] 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.

[68] Shengbo Song, Lelai Deng, Jun Gong, and Hanmei Luo. Gaia scheduler: A kubernetes-based scheduler framework. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/-SustainCom)*, pages 252–259. IEEE, 2018.

[69] Weiqi Zhang, Baosheng Wang, Wenping Deng, and Hao Zeng. Network control for large-scale container clusters. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 827–833. Springer, 2018.

[70] 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.

[71] 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.

[72] Lilu Zhu, Kai Huang, Yanfeng Hu, and Xianqing Tai. A self-adapting task scheduling algorithm for container cloud using learning automata. *IEEE Access*, 9:81236–81252, 2021.

[73] Saravanan 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.

[74] Ibrahim Alobaidan, Michael Mackay, and Posco Tso. Build trust in the cloud computing-isolation in container based virtualisation. In *2016 9th International Conference on Developments in eSystems Engineering (DeSE)*, pages 143–148. IEEE, 2016.

[75] Luciano Baresi, Sam Guinea, Alberto Leva, and Giovanni Quattrocchi. A discrete-time feedback controller for containerized cloud applications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 217–228, 2016.

[76] 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.

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

[78] 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.

[79] 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.

[80] 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.

[81] Oana-Mihaela Ungureanu, Călin Vlădeanu, 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.

[82] 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.

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

[84] Tarek Menouer and Patrice Darmon. New scheduling strategy based on multi-criteria decision algorithm. In *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 101–107. IEEE, 2019.

[85] Pradipta Ghosh, Quynh Nguyen, and Bhaskar Krishnamachari. Container orchestration for dispersed computing. In *Proceedings of the 5th International Workshop on Container Technologies and Container Clouds*, pages 19–24, 2019.

[86] M Saravanan and R Vignesh. Dsts: A hybrid optimal and deep reinforcement learning for dynamic scalable task scheduling on container cloud environment. 2022.

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

[88] M. Xavier, F. Neves, F. Rossi, and C. A. F. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. *Future Generation Computer Systems*, 71:12–21, 2018. doi: 10.1016/j.future.2016.10.022.

[89] P. Cetus, L. Ricci, and M. Colombo. Docker image vulnerabilities: A large-scale empirical study. *Journal of Systems and Software*, 169:110710, 2020. doi: 10.1016/j.jss.2020.110710.

[90] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. Technical report, IBM Research Report, 2015.

[91] M. Hale, T. Benson, and P. Natarajan. Secrets in the cloud: Understanding and mitigating the risks of secret sprawl in containerized environments. In *USENIX Security Symposium*, 2019.

[92] A. Abdullah, M. Raza, and M. Asif. Security threats and solutions in kubernetes orchestration. *IEEE Access*, 9:77689–77705, 2021. doi: 10.1109/ACCESS.2021.3083540.

[93] Y. Zhou, J. Zhang, and H. Wang. Secure and efficient monitoring for container-based cloud platforms. *IEEE Transactions on Cloud Computing*, 7(4):1029–1041, 2019. doi: 10.1109/TCC.2017.2754491.

[94] H. Hibshi, M. Kranch, and J. Delvaux. Supply chain threats in cloud-native architectures. In *ACM CCS Workshop on Secure Software Supply Chain (3SC)*, 2021.

[95] D. Boucher, L. Kerboeuf, and R. Rouvoy. Improving observability of containerized applications. In *Proceedings of the International Conference on Cloud Engineering (IC2E)*, pages 191–197, 2018.

[96] J. Cetus, L. Rivera, and A. Kumar. Security analysis of container images using open-source scanning tools: Clair, trivy, and anchore. *Journal of Cloud Security*, 8(1):23–33, 2020. Preprint or simulated data for this example.

[97] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. *IBM Research Report*, 5(2015):1–7, 2015.

[98] Zachary Hale and Aylin Yavuz. Sok: Security of secret management in container orchestration. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy Workshops (SPW)*, pages 299–304. IEEE, 2019.

[99] Shahid Abdullah, Shoaib Ur Rehman, Bilal Ahmed, Adnan Rauf, and Faheem Ullah Khan. Container orchestration security: A survey of techniques and challenges. *Computer Standards & Interfaces*, 73:103489, 2021.

[100] Yao Zhou, Abhishek Singh, Prateek Sharma, and Hongyu Wu. Containerguard: A runtime monitoring system for securing container environments. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 550–563. ACM, 2019.

[101] Hanan Hibshi, Rima Jabbour, and Ritesh Agarwal. Towards secure software supply chains with signed container images and ci/cd integrity. *IEEE Software*, 38(6):63–70, 2021.

[102] Philip Boucher, Timothy Lee, and Pratiksha Marathe. Containers and cloud: From lxc to docker to kubernetes. *Queue*, 16(4):30–45, 2018.

[103] Chang Zhao, Yusen Wu, Zujie Ren, Weisong Shi, Yongjian Ren, and Jian Wan. Quantifying the isolation characteristics in container environments. In *IFIP International Conference on Network and Parallel Computing*, pages 145–149. Springer, 2017.

[104] Ilias Mavridis and Helen Karatza. Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing. *Future Generation Computer Systems*, 94:674–696, 2019.

[105] Abderrahmane Boudi, Ivan Farris, Miloud Bagaa, and Tarik Taleb. Lightweight virtualization based security framework for network edge. In *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6. IEEE, 2018.

[106] Maria A Rodriguez and Rajkumar Buyya. Container-based cluster orchestration systems: A taxonomy and future directions. *Software: Practice and Experience*, 49(5):698–719, 2019.

[107] Min Sun and Yuqing He. Intelligent water drop-based dynamic resource allocation for cloud container orchestration. *Future Internet*, 15(3):78–89, 2023.

[108] Mufeed Ahmed Naji Saif, SK Niranjan, Belal Abdullah Hezam Murshed, Fahd A Ghanem, and Ammar Abdullah Qasem Ahmed. Cso-ilb: chicken swarm optimized inter-cloud load balancer for elastic containerized multi-cloud environment. *The Journal of Supercomputing*, 79(1):1111–1155, 2023.

[109] Suriya Phongmoo, Komgrit Leksakul, Nivit Charoenchai, and Chawis Boonmee. Artificial bee colony algorithm with pareto-based approach for multi-objective

three-dimensional single container loading problems. *Applied Sciences*, 13(11): 6601, 2023.

[110] Jessica González-San-Martín, Laura Cruz-Reyes, Claudia Gómez-Santillán, Héctor Fraire-Huacuja, Nelson Rangel-Valdez, Bernabé Dorronsoro, and Marcela Quiroz-Castellanos. A comprehensive review of task scheduling problem in cloud computing: Recent advances and comparative analysis. *New Horizons for Fuzzy Logic, Neural Networks and Metaheuristics*, pages 299–313, 2024.

[111] Samuel Laurén, M Reza Memarian, Mauro Conti, and Ville Leppänen. Analysis of security in modern container platforms. In *Research Advances in Cloud Computing*, pages 351–369. Springer, 2017.

[112] An Liu, Ming Gao, and Jiafu Tang. Multi-Mode Instance-Intensive Workflow Task Batch Scheduling in Containerized Hybrid Cloud. *IEEE Transactions on Cloud Computing*, 12(1):159–173, 2023. doi: 10.1109/TCC.2021.3071200.

[113] Shuai Wang, Xuejiao Zhang, Weisong Shi, and Wenhong Li. Network-Aware Container Scheduling in Multi-Tenant Data Center. arXiv:1909.07673, 2019. URL `https://doi.org/10.48550/arXiv.1909.07673`.

[114] Kanika Sharma, Parul Khurana, Ramandeep Sandhu, Chander Prabha, Harpreet Kaur, and Deepali Gupta. A hybrid approach to secure container orchestration: Intelligent water drop algorithm with anti-collocation and security affinity rules.

[115] Kalka Dubey and Subhash Chander Sharma. An extended intelligent water drop approach for efficient vm allocation in secure cloud computing framework. *Journal of King Saud University-Computer and Information Sciences*, 34(7): 3948–3958, 2022.

[116] Ronghui Cao, Peng Zhang, Yiming Wu, Jun Liu, and Haibin Su. Adaptive container scheduling based on reinforcement learning in kubernetes: R. cao et al. *CCF Transactions on High Performance Computing*, pages 1–14, 2025.

[117] Asmaa Halbouni, Teddy Surya Gunawan, Mohamed Hadi Habaebi, Murad Halbouni, Mira Kartiwi, and Robiah Ahmad. CNN–LSTM: Hybrid Deep Neural Network for Network Intrusion Detection System. *IEEE Access*, 10:99837–99849, 2022. doi: 10.1109/ACCESS.2022.3209138.

[118] Sergej Jakovlev and Miroslav Voznák. Auto-Encoder-Enabled Anomaly Detection in Acceleration Data: Use Case Study in Container Handling Operations. *Machines*, 10(9):734, 2022. doi: 10.3390/machines10090734.

[119] Ciprian Paduraru, Catalina Camelia Patilea, and Ştefan Iordache. Task Scheduling: A Reinforcement Learning Based Approach. In *Proc. 15th International Conference on Agents and Artificial Intelligence (ICAART)*, volume 3, pages 948–955. INSTICC, 2023.

Publications and Patents

Published/Accepted articles

[1] 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. (Published)

[2]Sharma, Kanika, Parul Khurana, and Mukesh Kumar. "Performance Metrics Analysis for Container Scheduling in Cloud Environment." 2024 Asian Conference on Intelligent Technologies (ACOIT). IEEE, 2024. (Published)

[3]Sharma, Kanika, and Parul Khurana. "A Deep Dive into Container Security Challenges, Strategies, and Solutions." International Conference on Recent Advances in Artificial Intelligence for Sustainable Development (RAISD) 2025 . IEEE, 2025. (Published)

[4]Sharma, Kanika, Parul Khurana, Ramandeep Sandhu, Chander Prabha, Harpreet Kaur, and Deepali Gupta. "A hybrid approach to secure container orchestration: intelligent water drop algorithm with anti-collocation and security affinity rules." (Published)

[5]Sharma, Kanika, and Ashok Kumar. ”Enhancing Container Security: A Proposed Novel Secure Scheduling Technique for Containers in Cloud Environment.” 2024 International Conference on Networks, Intelligence and Computing (ICONIC-2024). IEEE, 2024. (Accepted)

[6]Sharma, Kanika, and Parul Khurana. ”Exploring and Implementing Container Scheduling Methods: A Comparative Review and Practical Approach.” International Journal of Information Engineering and Electronic Business(IJIEEB), 2025. (Accepted)

[7]Sharma,Kanika and Ram Kumar.”Analysis of various existing and suggested scheduling techniques for allocation of pods to the nodes for better resource utilization in the Containers”. (UGC Published)

[8]Sharma, Kanika, and Parul Khurana. ”SecuFuzzDrop: Secure fuzzy and intelligent water drop based scheduler for secure container scheduling”.(Under Review)