# OPTIMIZED RESOURCE ALLOCATION FOR DOCKER CONTAINER BASED CLOUD ENVIRONMENTS USING HYBRID ADAPTIVE PSO AND DECISION TREE MODEL

**A Synopsis**

For the award of

**Doctor of Philosophy**

In

Computer/ IT Engineering

Submitted By:

**Manmitsinh Chandrasinh Zala**
**[199999913537]**

**Supervisor:**

Dr. Jaykumar Shantilal Patel
Professor
Hillwoods Academy of Higher Education (HAHE)
Gujarat Technological University
India

Submitted to

**Gujarat Technological University**
Ahmedabad, Gujarat, India.

Supervisor
Dr.jaykumar S. patel

DPC Member 1
Dr. Harshad B. Bhadka

DPC Member 2
Dr. Shivani A. Trivedi

# Contents

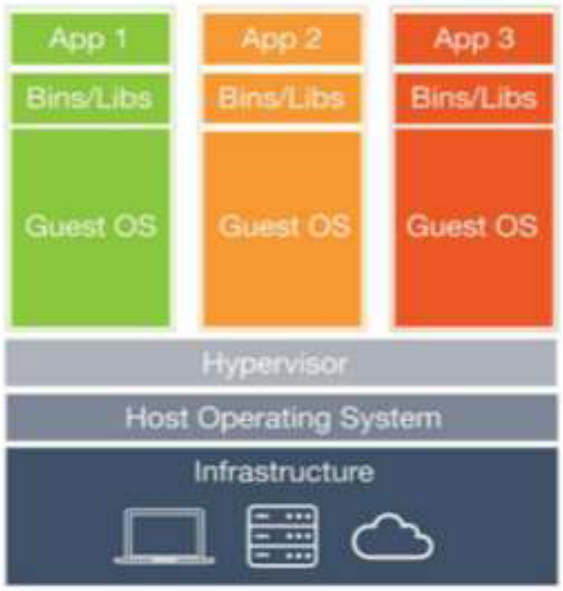
# 1 Abstract

In Cloud computing containerized applications are light weight virtualization where good performance , energy efficiency and cost depends on mainly how workloads are distributed and balanced across various swarm nodes , Docker swarm is lightweight design for virtualization, Docker swarm provides static scheduling methods ,which lacks real time dynamics often leading to imbalanced CPU usage ,This work introduces a hybrid intelligent load-balancing framework that combines a Decision Tree (DT)–based cost model with an Adaptive Particle Swarm Optimization (APSO) algorithm for context-aware, dynamic container placement. The DT component continuously classifies node state using live telemetry—CPU utilization, memory pressure, and task density—while APSO adjusts inertia and velocity parameters on the fly to refine the search process and mitigate premature convergence, improving performance over conventional PSO. The framework is implemented on a 1-master/2-worker Docker Swarm cluster running on Ubuntu VirtualBox, instrumented with Prometheus and cAdvisor for detailed monitoring and Grafana for real-time visualization. A mixed workload comprising CPU-intensive factorial computations and controlled memory operations are executed under three strategies: the native Swarm scheduler, a classical PSO-based scheduler, and the proposed DT+APSO model. Analysis of CPU utilization, memory footprint, variance, and 95% confidence intervals shows that DT+APSO lowers overall CPU utilization by 18–20% compared to the default Swarm scheduler and 10–12% relative to standard PSO, while keeping memory usage stable and reducing node-level variance. These gains translate into energy savings through reduced dynamic CPU power consumption. The modular design supports seamless integration into existing Swarm deployments and provides a scalable foundation for future multi-objective optimization involving latency, energy, and cost trade-offs, demonstrating the practical benefits of machine-learning-guided metaheuristic scheduling for container orchestration.

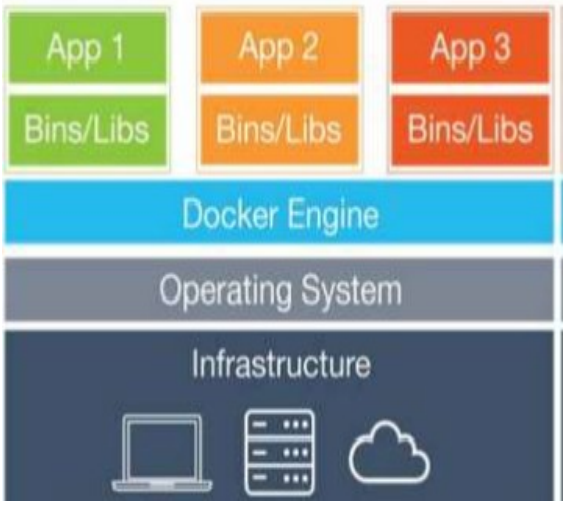
**Keywords:** Load Balancing, Resource Allocation, Heuristic Approach, Docker Container, Cloud Computing, Adaptive PSO, cAdvisor, Decision Tree, Docker Swarm, Grafana, Prometheus.

## 2 Brief description on the state of the art of the research topic

Containers, as a lightweight technology for virtualization applications, have recently revolutionized the management of cloud applications. In modern digital systems, cloud computing has become the core platform for digital services. Containerization is now being propounded as a very resilient technology for packaging applications and all their dependencies into portable units. In contrast to traditional virtual machines, it offers strengths in fast deployment, utilization of resources, and ease of portability [15]. Since it uses the same host operating system's kernel, overhead reduction is achieved with faster boot times [33], which enables further efficiency in resources and more rapid application deployment. Although cloud computing has transformed the way we use and access computing resources, it also comes with several issues, among which load balancing is included. Load balancing is one of the most important techniques in distributing incoming traffic across various servers to gain optimal performance, reliability, and scalability. [10]Recent trends in research and studies are pointing out the evolution of cloud technologies from VM-based architectures towards container-based approaches, and also throws light on the challenges and opportunities related to load balancing in containerized environments. We can say that containerization provides great advantages to load balancing, so many industries have adapted container-based architectures for managing their work load optimize resource allocation by improving their cloud-based architectures. Docker containers have transformed the deployment and development of software by enabling efficient resource management, so the development process and effectiveness, as well as scalability, are improved because containers provide lightweight processing and less dependency. In this paper, we proposed a Docker-based resource optimization solution with the use of heuristic approaches. The main challenge is resource allocation and optimization are addressed, also Other challenges such as scalability and security have also been explored,[9][22] along with a comparative analysis of Docker and other virtualization techniques. Particle Swarm Optimization (PSO) is a swarm intelligence algorithm inspired by the collective behavior of flocks of birds and schools of fish. First introduced by James Kennedy and Russell Eberhart in 1995[14], PSO has undergone significant development and has been successfully applied to a wide range of real-world problems. The algorithm operates by iteratively improving the positions of particles within a defined search space, where each particle updates its position based on its own best-known position as well as the best-known positions of other particles in the swarm.This process is guided by velocity [14]vectors that determine the magnitude and direction of particle movement. Recent efforts to improve PSO performance have been in parameter tuning and hybridization with other techniques, but these approaches often overlook the evolving nature of the optimization process. Thus, they lack a developed methodology to deal with the hard problems and may persist with weaknesses. Cloud computing encompasses numerous challenges, with load balancing standing as a crucial problem among them.Defined as "a technique, method, or strategy to efficiently manage resource utilization and allocate resources toclients, ensuring




(a) Virtual machine architecture




(b) Container Architecture

that neither overloading nor resource starvation occurs" [25], load balancing has been widelyexplored. A thorough literature review reveals that researchers have proposed various mechanisms and developednmultiple algorithms to optimize load distribution [10]. nowever, within the realm of containerized technology—still in its developmental stage—a definitive, universally effective approach to load balancing remains yet unsolved. In literature survey we have identified and classified load balancing approaches as shown in figure2. In this research paper, we focus on particle Swarm Optimization (PSO)-based algorithms for effective resource allocation in container-based cloud computing systems. As demonstrated in [2][7][12], PSO has been shown to outperform Ant Colony Optimization (ACO) in various scenarios. Cloud computing offers a wide range of PSO-based variations, with Standard PSO serving as the foundational model. This standard approach is commonly employed for basic load balancing and resource allocation in cloud environments. It relies on key parameters such as inertia weight, cognitive coefficients, and social coefficients to guide the search process and facilitate convergence toward optimal solutions. While Standard PSO is known for its speed and efficiency in handling straightforward tasks, it often struggles when applied to complex, high-dimensional problem spaces typical of cloud systems. The implementation of these algorithms are typically carried out using tools such as MATLAB, Python, or C++ . [10]

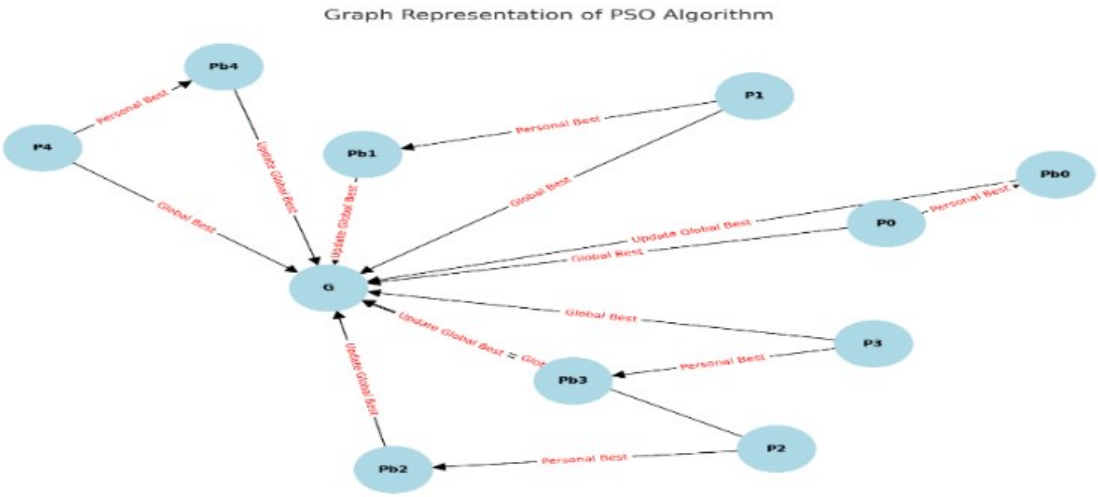



Figure 2: Illustration of Particle Swarm Optimization Algorithm

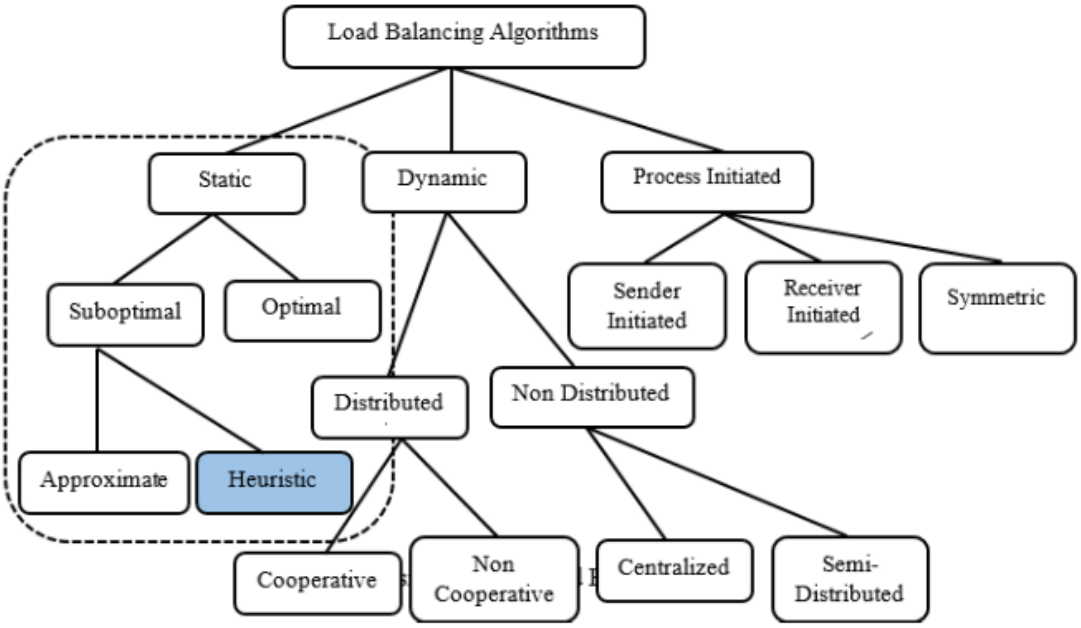



Figure 3: Types of Load Balancing Algorithms

Table 1: Comparative Analysis of Existing Container Scheduling and Load Balancing Approaches

| **Algorithm Name** | **Objectives** | **Advantage** | **Disadvantage** | **Metrics** | **Finding** |
|---|---|---|---|---|---|
| Container Scheduling Strategy based on Machine Learning in Microservice Architecture [5] | To improve performance using machine learning algorithms such as ANN and CSML. | Maintains load pressure and improves overall performance. | Does not fully consider inter-service relationships. | Response time, load pressure | Machine learning methods are important in microservice architecture and help regulate container load pressure through classification. |
| Accelerating Big Data Application using Lightweight Virtualization Framework on Enterprise Cloud [23] | To apply big data techniques in virtualization environments using Spark, RDD, and MapReduce. | Spark is much faster than Hadoop and is suitable for automatic application execution. | Suitable mainly for large applications; limited resources and research are available. | Total execution time, CPU utilization, disk utilization, memory utilization | Big data applications require efficient virtualization and scheduling to manage large amounts of data. |
| Chain Oriented Load Balancing in Microservice System [13] | To develop and maintain independent microservices that communicate through specific protocol requests. | Uses multi-priority queues and balances microservice chains while considering competition among microservices. | Shorter chains may be negatively affected; token bucket algorithm was used to address this issue. | Latency, processing capacity, efficiency | Very limited research focuses on latency-based scheduling in multi-queue algorithms. |

| Algorithm Name | Objectives | Advantage | Disadvantage | Metrics | Finding |
|---|---|---|---|---|---|
| Dynamic Priority Based Weighted Scheduling Algorithm (DPWS) [8] | To reduce high latency in chain-service requests caused by high concurrency. | Weighted fair queues dynamically improve the priority of chain-service requests. | Has little impact on leaf nodes, so full chain latency balancing is not achieved. | Latency, chain-service response, priority queues | Maintaining chain-service requests becomes difficult as concurrency and complexity increase. |
| A Decentralized System for Load Balancing of Containerized Microservice Architecture [19] | To build a swarm of tasks in the cloud and support live migration of Docker containers. | Container migration is faster than VM migration, and system calls enable load balancing. | Managing microservices through system function calls is complex. | Docker migration, microservice management | Monolithic enterprise systems are difficult to maintain and scale, motivating ESB-based architectures. |
| A New Docker Swarm Scheduling Strategy [5] | To build a new Swarm-based scheduling technique and create SLA classes for dynamic computation of containers according to SLA class and machine load. | Provides an economic model with multiple SLA classes such as Premium, Advance, and Best Effort. | Further improvement is needed in consolidation methods to dynamically decide the number of active machines for reducing energy consumption. | Energy consumption, SLA, low cost | Premium class provides highest priority for long services, Advance class supports short services with limited budget, and Best Effort class is intended for low-cost microservice solutions. |

| Algorithm Name | Objectives | Advantage | Disadvantage | Metrics | Finding |
|---|---|---|---|---|---|
| Kubernetes Resource Scheduling Scheme [20] | To study the orchestration functionality of Google Kubernetes. | Various algorithms were tested, and hybrid methods such as ACO+PSO showed efficient results. | Pods are not dynamically selected according to pod count, which affects low-usage cost efficiency. | Efficiency, resource allocation | Hybrid metaheuristic algorithms can improve resource allocation efficiency in Kubernetes environments. |
| Dynamic Balance Strategy of High Concurrent Web Cluster based on Docker Container [18] | To develop a high-concurrency and easily extendable dynamic cluster balancing strategy that provides high availability and strong concurrency performance. | Performs better than traditional Round Robin and Weighted Round Robin scheduling strategies. | Applicable only in certain business scenarios. | Concurrency, availability | Although Round Robin is widely used, the proposed strategy gives better results for availability and concurrency. |

# 3 Definition of the Problem

”Optimized Resource allocation for Docker container based cloud environments using Hybrid Adaptive PSO and Decision Tree Model”. This research pexplores various advanced Particle Swarm Optimization (PSO) variants and their applications in resource allocation for container-based cloud computing systems. Several enhanced versions of PSO have been developed to improve efficiency, adaptability, and optimization performance in complex cloud environments. Two-Memory PSO (TMPSO) [17][1][3] introduces additional memory to balance exploration and exploitation, leading to faster convergence and improved efficiency, particularly in resource-intensive scenarios. Adaptive PSO [23][12] dynamically adjusts control parameters to ensure scalability and fault tolerance, making it ideal for rapidly changing cloud environments. Multi-Objective PSO (MOPSO) [26]optimizes multiple objectives simultaneously, such as cost and time, using Pareto dominance to generate diverse solutions. Hierarchical PSO (HPSO)[16] structures particles in a hierarchy to enhance resource utilization, which is useful for multi-level decision-making tasks like clustering and scheduling. Cooperative PSO (CPSO) enables collaboration among particles, improving solution quality and convergence speed in complex cloud-based applications. Discrete PSO (DPSO)[1] is adapted for tasks like container placement by discretizing positions and velocities, making it effective for task scheduling. Quantum-behaved PSO (QPSO) [[4] incorporates quantum mechanics principles for enhanced search capabilities, benefiting applications that require high security and energy efficiency. Hybrid PSO combines PSO [6] with other optimization techniques like Genetic Algorithms (GA) or Simulated Annealing (SA) to improve convergence and resource allocation in cloud computing. Dynamic Multi-Swarm PSO (DMS-PSO)[13]is designed for dynamic cloud environments, using multiple interacting swarms to quickly adapt to changing conditions. Opposition-Based Learning PSO (OBL-PSO)[23] enhances global search and prevents local optima trapping by integrating opposition-based learning strategies. Chaotic PSO (CPSO) incorporates chaos theory to avoid premature convergence, making it suitable for environments with fluctuating workloads. Constriction Factor PSO (CF-PSO)[13] [9] stabilizes convergence by applying a constriction factor, ensuring reliable performance across various cloud optimization problems. In this paper we have imlemente hybrid Adaptive PSO and Decesion tree classifier to improve resource allocation and load balancing in Docker container-based ecosystem. Combining both algorithm allows efficient resource management and historical data in decision tree improves future dynamic load requirements, we try to achieve efficient resource allocation and improved performance for Docker container.

# 4 Objective and Scope

Existing Docker Swarm schedulers are not sufficiently adaptive to dynamic cloud workloads, often resulting in poor load balancing, node overloading, and inefficient resource utilization. This research addresses the problem by proposing a hybrid Adaptive PSO and Decision Tree model for intelligent and optimized resource allocation in container-based cloud environments. The primary objective of this research is to develop an intelligent and efficient resource allocation framework for Docker Swarm environments using a hybrid combination of Adaptive Particle Swarm Optimization (APSO) and Decision Tree techniques. The study aims to improve the overall performance of container orchestration by introducing an optimized mechanism for container placement across available worker nodes.

Another key objective is to enhance load balancing in the Docker Swarm cluster by distributing containers in a more balanced manner across worker nodes. This is expected to prevent overloading of individual nodes and improve the utilization of available computing resources.

The research also aims to reduce the imbalance in CPU and memory utilization within the cluster. By minimizing uneven resource consumption, the proposed model seeks to improve system stability, operational efficiency, and service reliability under varying workloads.

A further objective is to support node selection through Decision Tree-based suitability analysis. The Decision Tree model will be used to evaluate the fitness of each worker node based on system parameters and runtime conditions, thereby assisting the APSO algorithm in making more informed placement decisions.

Finally, the study aims to evaluate the performance of the proposed hybrid APSO–Decision Tree model against the default Docker Swarm scheduling strategy and other conventional approaches. The comparison will be carried out using relevant performance metrics such as load distribution, CPU utilization, memory usage, response behavior, and overall resource efficiency.

## 4.1 Scope of the Study

The scope of this research is focused on resource allocation and load balancing in container-based environments based on Docker Swarmarm. The work is limited to the design and implementation of a hybrid optimization model that combines Adaptive Particle Swarm Optimization with Decision Tree-based node-suitability analysis to improve scheduling decisions.

The study covers the analysis of Docker Swarm architecture, container placement strategies, and resource-aware scheduling mechanisms. It includes monitoring of worker node resources, such as CPU and memory utilization, to support intelligent decision-making during container deployment.

The proposed approach is intended to operate within a cluster consisting of a manager node and multiple worker nodes, where containers are deployed as services and distributed dynamically. The research mainly addresses the problem of imbalance in resource usage caused by default or less intelligent scheduling techniques.

The scope also includes performance evaluation of the proposed model through experimental analysis and comparison with existing Docker Swarm scheduling and conventional optimization methods. The evaluation is restricted to selected performance parameters related to load balancing and resource utilization.

However, the study does not primarily focus on application-level optimization, network-level security mechanisms, or large-scale heterogeneous cloud federation. Its central emphasis remains on improving container placement efficiency, node selection accuracy, and balanced resource utilization in Docker Swarm clusters through the proposed hybrid model.

## 4.2 Objectives

To develop a hybrid Adaptive PSO and Decision Tree model for Docker Swarm-based resource allocation. To improve load balancing by optimizing container placement across worker nodes. To reduce CPU and memory utilization imbalance in the cluster. To support node selection using Decision Tree-based suitability analysis. To evaluate the proposed model against existing Docker Swarm and conventional methods.

## 4.3 Scope

The study focuses on Docker Swarm-based container resource allocation and load balancing. It proposes a hybrid APSO and Decision Tree approach for intelligent container placement. The work considers resource parameters such as CPU and memory utilization for node selection. The study is limited to manager-worker cluster architecture in Docker Swarm. Performance evaluation is carried out by comparing the proposed approach with existing scheduling methods. The research mainly addresses resource balancing and scheduling efficiency, not broader cloud security or application-layer issues.

# 5 Original contribution by the Thesis

The original contribution of this thesis lies in the development of a hybrid intelligent resource allocation framework for Docker Swarm environments by integrating Adaptive Particle Swarm Optimization (APSO) with Decision Tree-based node suitability analysis. Unlike conventional Docker Swarm scheduling approaches that rely on predefined or basic placement strategies, the proposed method introduces an adaptive, data-driven mechanism for selecting worker nodes and placing containers more efficiently under dynamic workloads.

A significant contribution of the thesis is the use of APSO for container placement optimization in a Docker Swarm cluster. The adaptive behavior of APSO improves the search capability of the standard PSO algorithm by dynamically adjusting optimization parameters, thereby enhancing convergence, avoiding premature stagnation, and producing better scheduling decisions for load balancing. This makes the proposed method more effective for volatile and real-time containerized environments.

Another important contribution is the incorporation of a Decision Tree model for node suitability evaluation. The Decision Tree helps identify the most appropriate worker nodes based on system-level features such as CPU usage, memory availability, and current load conditions. This combination of rule-based intelligence and adaptive optimization provides a more informed scheduling framework than methods based solely on heuristic or static allocation mechanisms.

The thesis also contributes by addressing the practical problem of imbalance in CPU and memory utilization across Docker Swarm worker nodes. The proposed model is designed to reduce uneven resource consumption, improve cluster stability, and enhance the overall efficiency of service deployment. As a result, the study provides a resource-aware and load-balanced orchestration strategy that can support better cluster performance.

A further original contribution is the comparative performance evaluation of the proposed hybrid APSO–Decision Tree model against default Docker Swarm scheduling and conventional methods. Through this analysis, the thesis demonstrates the effectiveness of the proposed approach in improving load distribution, node utilization, and container placement efficiency, thereby establishing its practical relevance in container-based cloud environments.

Overall, the thesis contributes a novel hybrid optimization and classification-based scheduling model for Docker Swarm, offering an intelligent, adaptive, and scalable solution for resource allocation and load balancing in container orchestration systems.

# 6 Methodology of Research, Results / Comparisons

Table 2: Comparative Analysis of PSO Variants

| **Algorithm Name** | **Usage** | **Parameters** | **Result** | **Findings** | **Tools Used** |
|---|---|---|---|---|---|
| Standard PSO [14] | Resource allocation, load balancing | Inertia weight, cognitive and social coefficients | Basic optimization with good convergence speed | Effective for simple problems but struggles with complex landscapes | MATLAB, Python, C++ |
| Two-Memory PSO (TMPSO) [17] | Improved resource management | Inertia weight, cognitive and social coefficients | Enhanced optimization with faster convergence | Better memory utilization and more efficient than standard PSO | MATLAB, Python, Apache Bench |
| Adaptive PSO [11] | Scalability, fault tolerance | Adaptive control parameters, learning rates | Highly scalable and robust against faults | Balances performance and resource use but requires dynamic adjustment | MATLAB, Python, Java |
| Multi-Objective PSO (MOPSO) [26] | Handling multiple objectives | Multiple objective functions | Efficient multi-objective optimization with diverse solutions | Suitable for complex problems with high efficiency | MATLAB, Python, R |
| Hierarchical PSO (HPSO) [4] | Task scheduling, data clustering | Hierarchical structure, social coefficients | Improved convergence and resource utilization | Suitable for hierarchical problems and clustering tasks | MATLAB, Python, Java |
| Cooperative PSO (CPSO) [15] | Enhanced collaboration among particles | Cooperative parameters, social coefficients | Better convergence and optimization | Effective for cooperative problem solving | MATLAB, Python |

| Algorithm Name | Usage | Parameters | Result | Findings | Tools Used |
|---|---|---|---|---|---|
| Discrete PSO (DPSO) [4] | Task scheduling, data clustering | Discrete position and velocity | Efficient scheduling and clustering | Suitable for discrete problems but limited by size | MATLAB, Python, C++ |
| Quantum-behaved PSO (QPSO) [24] | Security optimization, energy efficiency | Quantum potential well, particle position | High security and low energy consumption | Effective in specific applications, requires tuning | MATLAB, Python, C# |
| Hybrid PSO [21] | Job-shop scheduling | Combination of PSO with other algorithms | Improved convergence and resource utilization | Better for complex scenarios but computationally intensive | MATLAB, Python, Java |
| Dynamic Multi-Swarm PSO (DMS-PSO) [21] | Dynamic resource allocation | Dynamic parameters, swarm division | Responsive and efficient allocation | Adapts well to changing environments but complex | MATLAB, Python, R |

## 6.1 Proposed Algorithm

**Algorithm 1 Adaptive PSO with Decision Tree for Load Balancing**

**Initialization**

Define parameters:

Number of particles $P$, Maximum iterations $I_{max}$, Inertia weight $w$, Cognitive coefficient $C_1$, Social coefficient $C_2$, Population size for classification $C_{pop}$

Initialize positions and velocities of particles randomly within the permissible range

Initialize local best position for each particle

Identify and set the global best position

**Classification for Load Prediction**

Train a Decision Tree Classifier:

Collect historical data with features: CPU usage, memory usage, and resource allocation

Use load labels (e.g., low, medium, high) and train the classifier

Predict the load for each container using the trained classifier based on current features

**Fitness Evaluation**

Define the fitness function:

$$f = \sum_{i=1}^{N} \left( \frac{CPU_i}{\text{Total CPU}} + \frac{MEMORY_i}{\text{Total MEMORY}} + \frac{RESOURCE_i}{\text{Total RESOURCE}} \right)$$

11: Calculate the fitness for each particle

**Update Particles**

12: Adaptive update of inertia weight:

$$w = w_{\max} - \frac{(w_{\max} - w_{\min})}{I_{max}} \times \text{iteration}$$

13: Update velocity for each particle:

$$v_{ij}(t+1) = w \cdot v_{ij}(t) + c_1 \cdot r_1 \cdot (p_{ij} - x_{ij}) + c_2 \cdot r_2 \cdot (g_j - x_{ij})$$

14: Update position for each particle:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1)$$

15: Apply boundary conditions to ensure positions are within permissible range

**Update Local and Global Bests**

16: Evaluate fitness of each particle's new position

17: Update local best if the new position has better fitness

18: Update global best if the new position has the best overall fitness

**Termination**

19: **if** Maximum iterations $I_{max}$ reached or fitness converges **then**

20: Terminate the algorithm

21: **end if**

22: Output the global best position as the optimal load distribution =0

## 6.2 Methodology

The methodology adopted in this research is designed to evaluate an intelligent load balancing framework for Docker Swarm environments by combining container orchestration, real-time telemetry monitoring, Decision Tree-based node suitability estimation, Adaptive Particle Swarm Optimization (APSO), and an automated orchestration pipeline. The complete experimental workflow is implemented in a controlled Docker Swarm cluster and supported by continuous metric collection and comparative evaluation. The proposed framework aims to improve the allocation of service replicas across worker nodes while minimizing resource imbalance and enhancing scheduling efficiency.

The experimental platform consists of a three-node Docker Swarm cluster comprising one master node and two worker nodes. Each node is configured with 8 CPU cores, 8 GB RAM, and 20 GB memory under an Ubuntu operating system deployed through VirtualBox. The master node coordinates task scheduling and cluster control, whereas the two worker nodes execute the deployed containers. This cluster architecture provides a compact yet realistic environment for testing dynamic load balancing strategies in containerized cloud systems. The use of Docker Swarm enables distributed scheduling, service replication, resource sharing, and overlay networking among participating nodes.

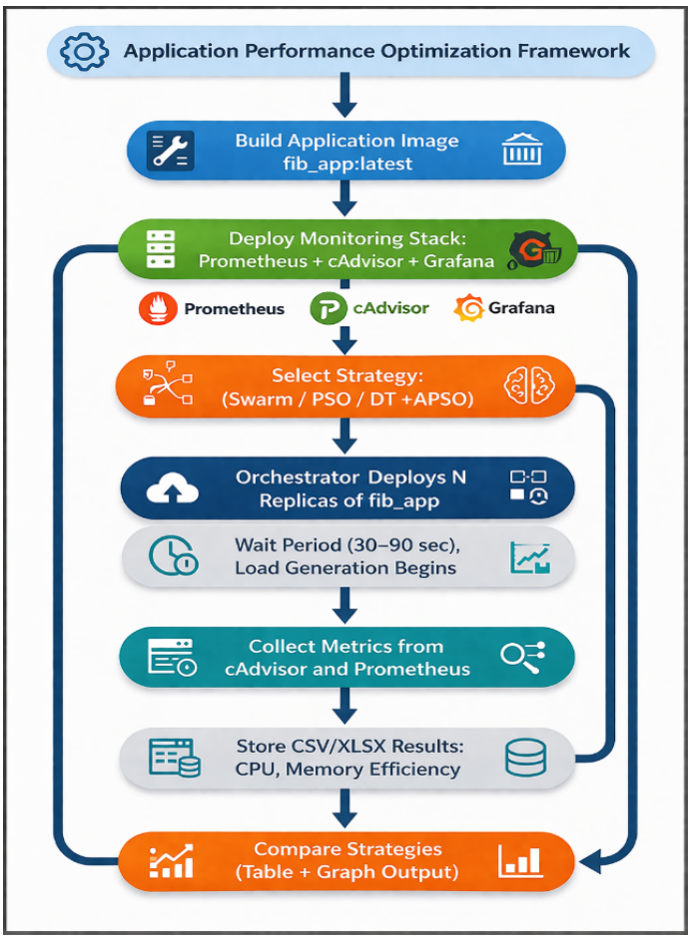



Figure 4: Praposed Architecture

## 6.3 System Architecture

The experimental testbed used in this study is shown in Table 3.

Table 3: Experimental Testbed

| Node | IP | CPU | RAM | Memory |
|---|---|---|---|---|
| Master | 10.0.4.42 | 8 Cores | 8 GB | 20 GB |
| Worker1 | 10.0.4.43 | 8 Cores | 8 GB | 20 GB |
| Worker2 | 10.0.4.49 | 8 Cores | 8 GB | 20 GB |

The architecture includes one master node and two worker nodes. All nodes run Ubuntu under VirtualBox and participate in the Swarm overlay network. The proposed system supports distributed container scheduling, resource sharing, real-time container metrics through cAdvisor, and monitoring and visualization via Prometheus and Grafana. The complete architecture consists of five major modules: application containers, monitoring stack, load-balancing algorithms, Python orchestrator, and result storage with visualization.

The application container used in the experiments is a CPU-intensive factorial computation

service tagged as `fib_app:latest`. The monitoring stack, consisting of Prometheus, Grafana, and cAdvisor, is deployed as a Docker stack. The scheduling strategies considered include the default Docker Swarm scheduler, a standard PSO scheduler, and the proposed Decision Tree with APSO scheduler. A Python-based orchestrator automates deployment, waiting, data collection, removal, and comparison across all strategies. The experimental outputs are stored in `.csv` and `.xlsx` formats and visualized

## 6.4 Application Workload

To generate a meaningful workload, a CPU-intensive factorial computation service named `fib_app` is deployed across the cluster. This workload continuously produces computational stress and therefore serves as an effective benchmark for analyzing CPU saturation, load distribution fairness, scheduler responsiveness, optimization convergence, and Decision Tree-driven node suitability. In addition to CPU-intensive behavior, controlled memory allocation is introduced within the containers to examine memory-aware scheduling performance. Thus, the application workload provides a suitable basis for measuring both computational and memory balancing under repeated execution conditions. :c

## 6.5 Monitoring Framework

A real-time monitoring framework is integrated into the system using Prometheus, cAdvisor, and Grafana. Prometheus periodically collects telemetry from all nodes and containers, including CPU utilization, memory usage, running container counts, and node-level performance statistics. cAdvisor is deployed globally across the Swarm nodes to expose container-specific CPU, memory, and I/O statistics, while Grafana is used to visualize runtime variations such as CPU peaks, memory heatmaps, replica migration patterns, and node load changes. This monitoring framework ensures that all scheduling outcomes are supported by observed runtime measurements rather than static assumptions.

## 6.6 Scheduling Strategies

Three scheduling strategies are considered in the experimental design. The first is the default Docker Swarm scheduler, which serves as the baseline approach. It uses internal heuristics such as spread, bin-pack, or random placement, but it does not consider historical telemetry, future overload risk, or predictive node suitability.

The second strategy is a standard PSO-based scheduler in which each particle represents a candidate replica distribution across worker nodes. The objective of optimization is to minimize CPU variance across nodes and thereby improve load balancing. However, this strategy uses fixed inertia and learning coefficients, which restrict adaptability under changing workload conditions and may lead to premature convergence.

## 6.7 Proposed Decision Tree with APSO Model

The proposed hybrid scheduler combines Decision Tree-based node cost estimation with Adaptive Particle Swarm Optimization. In this framework, a Decision Tree classifier is trained online using features such as CPU utilization, memory usage, active container count, historical load slopes, and CPU throttling rate. The classifier produces a node cost score representing the future overload risk of each worker node. Nodes with lower cost values are considered more suitable for receiving new service replicas. This predictive component allows the scheduling process to consider both present resource states and near-future behavior.

The APSO component uses the Decision Tree-derived information during replica allocation. Let $x_i(t)$ denote the position of particle $i$ at iteration $t$, $v_i(t)$ denote its velocity, $p_i(t)$ denote its personal best position, and $g(t)$ denote the global best solution. The standard PSO velocity update is expressed as

$$v_i(t+1) = \omega(t)v_i(t) + c_1(t)r_1\big(p_i(t) - x_i(t)\big) + c_2(t)r_2\big(g(t) - x_i(t)\big) \tag{1}$$

and the position update is given by

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2}$$

where $r_1$ and $r_2$ are uniformly distributed random values in the range $[0, 1]$. Unlike conventional PSO, APSO modifies the inertia weight dynamically according to fitness improvement. If the global improvement is small, a higher inertia value is maintained to encourage exploration; if the improvement is large, the inertia weight is reduced to support exploitation and faster convergence.

The improvement in global fitness is computed as

$$\Delta f(t) = \left| \frac{f(g(t-1)) - f(g(t))}{f(g(t-1))} \right| \tag{3}$$

and the adaptive inertia function is defined as

$$\omega(t) = \omega_{\min} + (\omega_{\max} - \omega_{\min})e^{-\alpha \Delta f(t)} \tag{4}$$

where $\alpha$ is the adaptation control constant. Similarly, the cognitive and social coefficients are adjusted adaptively over iterations according to

$$c_1(t) = c_{1,\max} - (c_{1,\max} - c_{1,\min})\,\frac{t}{T_{\max}} \tag{5}$$

$$c_2(t) = c_{2,\max} - (c_{2,\max} - c_{2,\min})\,\frac{t}{T_{\max}} \tag{6}$$

This strategy ensures that earlier iterations emphasize exploration, while later iterations focus on exploitation and convergence. The final APSO update equation is therefore written as

$$v_i(t+1) = \omega(t)v_i(t) + c_1(t)r_1\big(p_i(t) - x_i(t)\big) + c_2(t)r_2\big(g(t) - x_i(t)\big) \tag{7}$$

The main strengths of APSO in this framework include responsiveness to changing workloads, avoidance of stagnation, mathematically stable replica allocation, and effective integration of the Decision Tree cost into the fitness calculation. The final replica allocationcorrected to an integercted to preserve the total number of replicas.

## 6.8 Mathematical Model of Decision Tree (CART) for Container Load Classification

In this study, a Classification and Regression Tree (CART) model is employed to classify containers based on their resource utilization. The model uses CPU and memory usage as input features to categorize the system load into three classes: Low, Medium, and High. This classification assists in identifying node suitability for optimized container placement.

### 6.8.1 Feature Representation

Each container is represented by a feature vector:

$$\mathbf{x} = [x_1, x_2] \tag{8}$$

where $x_1$ denotes CPU usage (%) and $x_2$ denotes memory usage (%). The output variable is defined as:

$$y \in \{\text{Low}, \text{Medium}, \text{High}\} \tag{9}$$

### 6.8.2 CART Splitting Criterion

The CART model constructs a binary decision tree by recursively partitioning the dataset. At each node, the best split is selected based on the Gini impurity index.

$$Gini(D) = 1 - \sum_{k=1}^{K} p_k^2 \tag{10}$$

where $p_k$ represents the probability of class $k$ in dataset $D$, and $K = 3$ corresponds to the three classes.

For a candidate split, dividing the dataset into left ($D_L$) and right ($D_R$) subsets, the weighted Gini index is computed as:

$$Gini_{split} = \frac{|D_L|}{|D|} Gini(D_L) + \frac{|D_R|}{|D|} Gini(D_R) \tag{11}$$

The optimal split $(j, s)$ is selected by minimizing:

$$\min_{j,s} \; Gini_{split}(j, s) \tag{12}$$

### 6.8.3 Decision Rules for Load Classification

Based on predefined thresholds, the container load is classified as follows:

$$f(x_1, x_2) = \begin{cases} \text{High}, & x_1 > 80 \land x_2 > 75 \\ \text{Low}, & x_1 < 30 \land x_2 < 25 \\ \text{Medium}, & \text{otherwise} \end{cases} \tag{13}$$

where $x_1$ and $x_2$ represent CPU and memory usage, respectively.

### 6.8.4 Region-Based Representation

The decision tree partitions the feature space into regions:

$$R_1 = \{x : x_1 > 80 \land x_2 > 75\} \Rightarrow \text{High} \tag{14}$$

$$R_2 = \{x : x_1 < 30 \land x_2 < 25\} \Rightarrow \text{Low} \tag{15}$$

$$R_3 = \text{Remaining region} \Rightarrow \text{Medium} \tag{16}$$

### 6.8.5 Probability-Based Prediction

For each region $R_m$, the class probability is computed as:

$$P(y = k \mid x \in R_m) = \frac{N_k}{N_m} \tag{17}$$

where $N_k$ is the number of samples belonging to class $k$ and $N_m$ is the total number of samples in region $R_m$. The predicted class is given by:

$$\hat{y} = \arg\max_k P(y = k \mid x) \tag{18}$$

### 6.8.6 Illustrative Example

Consider a dataset of five containers as shown in Table 4.

Table 4: Sample Container Dataset

| Container | CPU (%) | Memory (%) | Class |
|---|---|---|---|
| C1 | 85 | 80 | High |
| C2 | 20 | 20 | Low |
| C3 | 50 | 40 | Medium |
| C4 | 90 | 85 | High |
| C5 | 25 | 20 | Low |

The Gini impurity of the root node is computed as:

$$Gini(D) = 1 - \left(\frac{2}{5}^2 + \frac{2}{5}^2 + \frac{1}{5}^2\right) = 0.64 \tag{19}$$

After splitting on CPU usage, the impurity reduces significantly, indicating an effective partition.

## 6.9 Integration with APSO Framework

The Decision Tree output is incorporated into the Adaptive PSO framework by assigning numerical cost values to each class:

$$Cost = \begin{cases} 3, & \text{High} \\ 2, & \text{Medium} \\ 1, & \text{Low} \end{cases} \tag{20}$$

This cost is integrated into the fitness function:

$$Fitness = \sum_{i=1}^{N} \left( \frac{CPU_i}{CPU_{total}} + \frac{MEM_i}{MEM_{total}} + \lambda \cdot Cost_i \right) \tag{21}$$

where $\lambda$ is a weighting factor controlling the influence of predicted load.

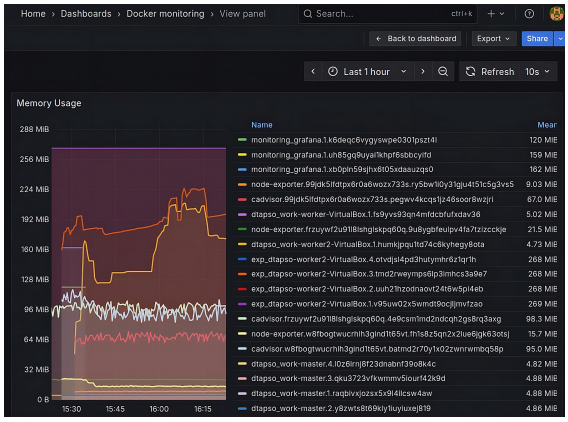
The integration of CART with APSO enhances predictive scheduling by combining classification-based node suitability with adaptive optimization, thereby improving load balancing efficiency in Docker Swarm environments.

## 6.10 Automated Orchestrator Pipeline

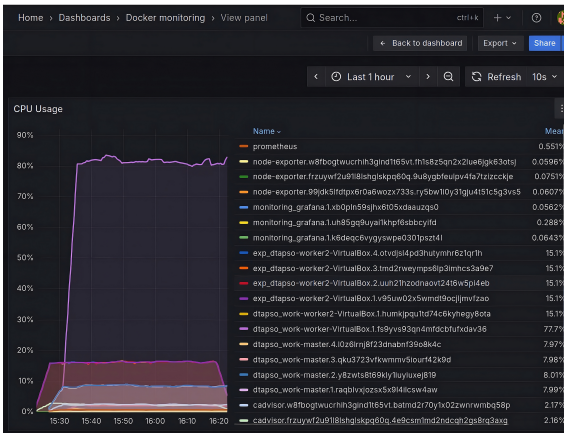
To ensure reproducibility and consistency, a Python-based orchestrator controls the complete experiment pipeline. The orchestrator manages service deployment, stabilization time, monitoring collection, service removal, and repeated evaluation for each scheduling strategy. All experimental outputs are exported in `.csv` and `.xlsx` formats for subsequent analysis, while Grafana dashboards provide real-time visual support during execution. This automation guarantees that all strategies are evaluated under identical workload and infrastructure conditions.

## 6.11 Results

The experimental results demonstrate that the proposed hybrid Decision Tree with APSO scheduler provides improved load distribution compared with both the default Docker Swarm scheduler and the standard PSO-based method. Under the CPU-intensive factorial workload, the baseline Docker Swarm scheduler exhibits uneven container placement because its internal heuristics do not incorporate predictive node analysis or adaptive optimization. As a result, one worker node may receive a higher concentration of demanding containers, leading to unbalanced CPU usage and increased resource skew across the cluster. A similar trend is observed for memory utilization, where the absence of predictive resource-awareness limits the scheduler's ability to achieve uniform deployment. The standard PSO-based strategy improves load distribution




(a) Figure 3. Grafautilizationilization




(b) Figure 4:Grafana CPU utilization

Table 5: Total CPU utilization per node

| Strategy | Node | CPU Sum |
|---|---|---|
| DT+APSO | Total | **41.5333** |
| | 10.0.4.42 | 10.5146 |
| | 10.0.4.43 | 20.3312 |
| | 10.0.4.49 | 10.6875 |
| PSO | Total | **45.8479** |
| | 10.0.4.42 | 11.6747 |
| | 10.0.4.43 | 22.4355 |
| | 10.0.4.49 | 11.7377 |
| Swarm-Def. | Total | **39.1625** |
| | 10.0.4.42 | 11.4539 |
| | 10.0.4.43 | 15.9302 |
| | 10.0.4.49 | 11.7784 |

Table 6: Total memory utilization per node

| Strategy | Node | Memory Sum |
|---|---|---|
| DT+APSO | Total | **1626.2452** |
| | 10.0.4.42 | 664.5458 |
| | 10.0.4.43 | 569.5854 |
| | 10.0.4.49 | 392.1139 |
| PSO | Total | **1798.8996** |
| | 10.0.4.42 | 737.7496 |
| | 10.0.4.43 | 626.9549 |
| | 10.0.4.49 | 434.1951 |
| Swarm-Def. | Total | **1795.0890** |
| | 10.0.4.42 | 734.0947 |
| | 10.0.4.43 | 628.6942 |
| | 10.0.4.49 | 432.3000 |

Table 7: CPU utilization strategy-wise on each node

| Node | Strategy | CPU Sum |
|---|---|---|
| 10.0.4.42 | Total | **33.6432** |
| | DT+APSO | 10.5146 |
| | PSO | 11.6747 |
| | Swarm-Def. | 11.4539 |
| 10.0.4.43 | Total | **58.6969** |
| | DT+APSO | 20.3312 |
| | PSO | 22.4355 |
| | Swarm-Def. | 15.9302 |
| 10.0.4.49 | Total | **34.2036** |
| | DT+APSO | 10.6875 |
| | PSO | 11.7377 |
| | Swarm-Def. | 11.7784 |

by searching for an optimized replica placement that minimizes CPU variance across the worker nodes. In several experimental runs, this method produces better balance than the default scheduler. However, the fixed inertia and fixed learning coefficients reduce its ability to react to changing workload characteristics. Consequently, the optimizer may converge prematurely or become trapped in suboptimal allocations, which limits its practical effectiveness in dynamic containerized environments.

The proposed DT-APSO method produces the most stable and balanced results among all compared strategies. The integration of Decision Tree-based node cost estimation into the fitness function enables the scheduler to identify nodes that are more likely to become overloaded in the near future. This predictive capability reduces risky placements and improves fairness in replica allocation across the worker nodes. The adaptive behavior of APSO further improves the optimization process by modifying the search trajectory as the experiment progresses, thereby avoiding stagnation and promoting better convergence toward balanced solutions. Consequently, CPU peaks are reduced, memory usage becomes more evenly distributed, and overall cluster stability improves.

The monitoring framework confirms the effectiveness of the proposed approach. Grafana dashboards show smoother node load variations and better-balanced replica placement when DT-APSO is used. The telemetry captured through Prometheus and cAdvisor indicates that the proposed model avoids excessive concentration of containers on a single worker node and maintains more stable system behavior across repeated experiments. Because the orchestrator executes all strategies under identical conditions, the observed differences can be attributed directly to the scheduling logic rather than environmental variation.

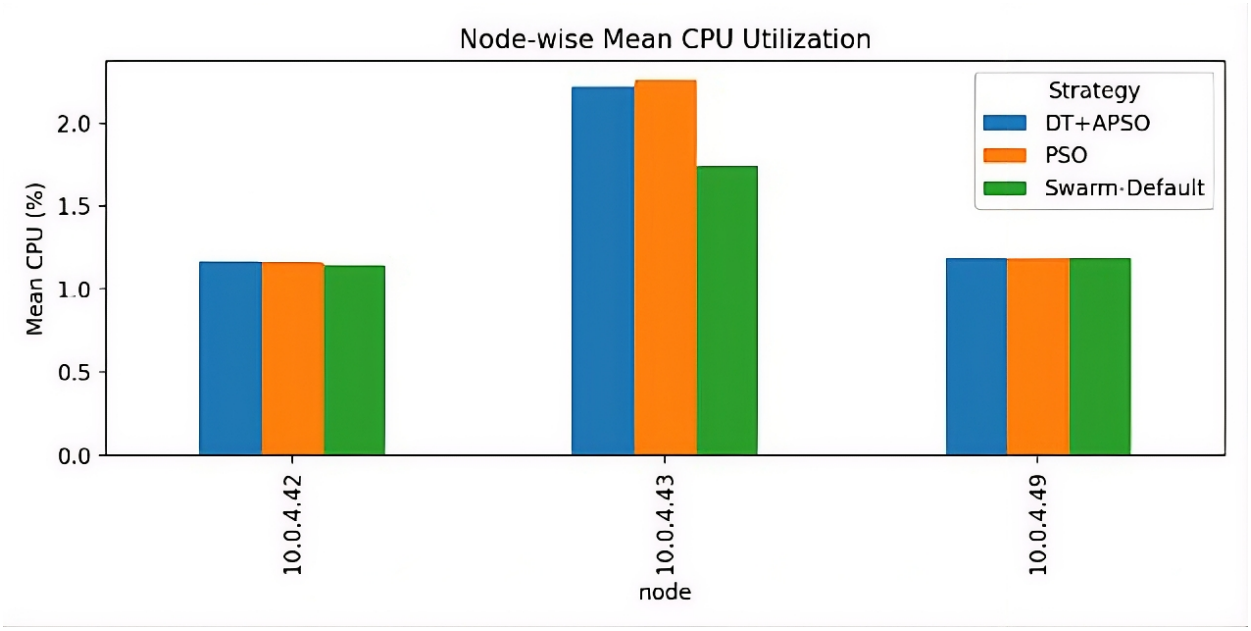



Figure 6: Node wise CPU Utilization

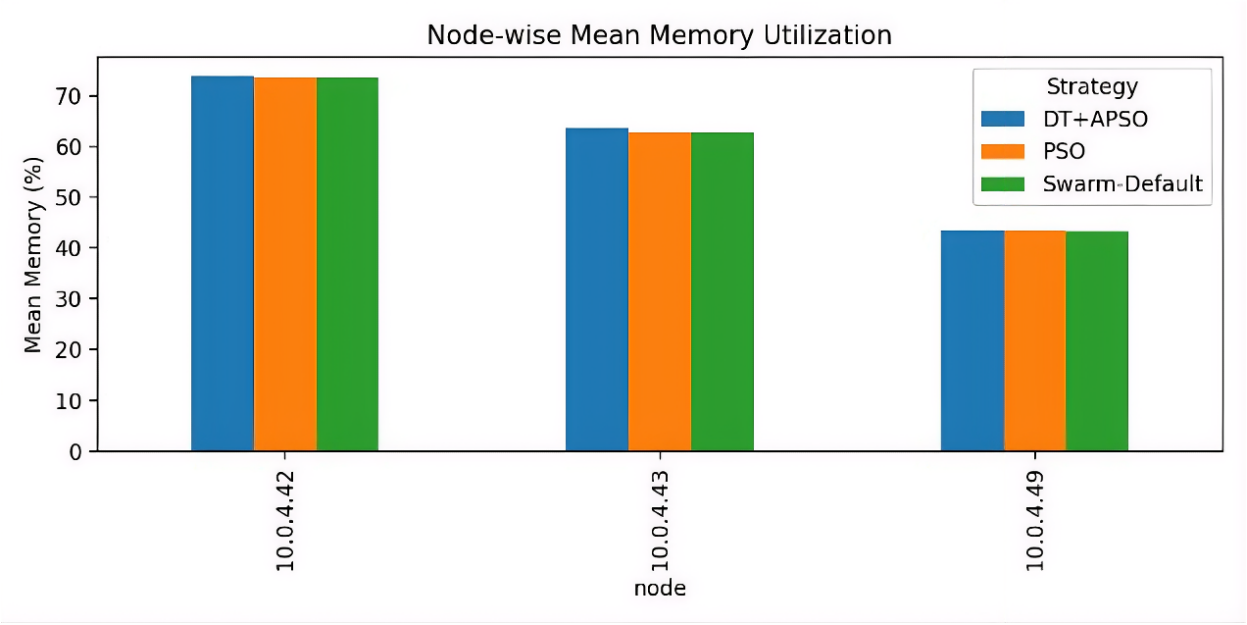



Figure 7: Node wise Memory Utilization

## 6.12 Comparison

A comparative assessment of the three scheduling strategies reveals clear differences in adaptability, predictive capability, and load balancing performance. The default Docker Swarm scheduler is the simplest to deploy and requires no additional optimization or learning model. However, its dependence on internal heuristics makes it unsuitable for dynamic environments where future overload risk and historical resource behavior are important. It therefore serves primarily as a baseline for comparison.

The standard PSO-based scheduler introduces optimization-driven scheduling and achieves better balancing performance than the baseline. By minimizing CPU variance, it improves replica placement and reduces some of the imbalance observed under default Swarm scheduling. Nevertheless, the absence of predictive learning and adaptive control parameters reduces its robustness when workload conditions change over time. Its fixed inertia and learning factors do not provide the most effective balance between exploration and exploitation across the full optimization cycle.

In contrast, the proposed DT-APSO scheduler combines classification-based prediction with adaptive optimization. The Decision Tree model contributes a forward-looking estimate of node suitability, while APSO dynamically refines the search process in response to optimization progress. This dual mechanism enables the scheduler to make more informed and more effective container placement decisions than the other two approaches. In terms of comparative behavior, the proposed method achieves stronger reduction in CPU and memory imbalance, higher adaptability to workload variation, and more stable replica distribution across the cluster. Its ability to integrate real-time monitoring data into both prediction and optimization makes it especially suitable for practical containerized cloud environments. index=18

Overall, the comparative analysis indicates that the default Docker Swarm scheduler is useful only as a basic benchmark, the standard PSO method offers moderate improvement, and the proposed DT-APSO strategy provides the most effective solution for intelligent load balancing in the studied Docker Swarm cluster. These findings justify the selection of the hybrid DT-APSO model as the principal contribution of the research and establish its superiority for dynamic resource allocation in container-based distributed systems.

# 7 Achievements with respect to objectives

The research work carried out in the present study has achieved significant progress with respect to the objectives formulated at the beginning of the doctoral work. The primary objective of the research was to investigate efficient load balancing and resource optimization mechanisms in Docker container cloud environments using intelligent optimization techniques. In this direction, the study successfully explored the applicability of Particle Swarm Optimization (PSO)-based approaches for dynamic container placement and resource management. A comparative framework was established to evaluate the performance of different load balancing strategies in terms of CPU utilization, memory utilization, and node-wise workload distribution.

One of the major achievements of the research is the successful implementation and analysis of PSO-based load balancing in a containerized cloud setup. The work demonstrated that metaheuristic optimization techniques are capable of improving resource allocation decisions compared to conventional scheduling approaches. Experimental results indicated that PSO-based strategies can provide better workload distribution across Docker nodes and reduce the possibility of resource overloading on individual hosts.

A further important achievement of the study is the development of an adaptive hybrid framework combining Decision Tree classification with Adaptive Particle Swarm Optimization (DT+APSO). This hybrid approach enhanced the intelligence of the scheduling mechanism by integrating predictive decision-making with optimization-based search. The Decision Tree component contributed to the identification of suitable nodes based on resource conditions, while APSO improved the allocation process by dynamically adapting the search behavior. The proposed DT+APSO framework showed improved balancing capability and more stable utilization patterns in comparison with standard PSO and Docker Swarm default scheduling.

The research has also achieved meaningful outcomes in terms of experimental validation. A Docker Swarm-based environment was used for implementation and performance evaluation under realistic distributed conditions. The analysis of CPU and memory utilization across multiple worker nodes confirmed that the proposed strategy can effectively reduce uneven resource concentration and improve balanced utilization of available infrastructure. This validates the practical relevance of the proposed method for container orchestration and cloud resource scheduling.

Another notable achievement is the publication of research findings in reputed Scopus-indexed journals. The published papers arising from this thesis document the evolution of the work from comparative PSO-based load balancing to adaptive resource optimization using hybrid intelligent techniques. These publications provide academic validation of the novelty, continuity, and contribution of the research work.

Overall, the achievements of the present research confirm that the objectives of the thesis have been substantially addressed. The study has contributed a comparative understanding of PSO-based load balancing, proposed an enhanced hybrid DT+APSO approach for containerized envi-

ronments, validated the method experimentally, and disseminated the outcomes through quality research publications. These achievements establish a strong foundation for the final thesis and indicate the potential of the proposed framework for further extension in intelligent cloud resource management.

# 8 Conclusion and Future Work

This study proposes a hybrid Decision Tree and Adaptive Particle Swarm Optimization (DT+APSO) framework for efficient container load balancing in Docker Swarm clusters. The approach improves resource allocation by combining intelligent node selection with adaptive optimization based on real-time system conditions. Experimental observations show better CPU and memory balance, improved replica placement, and more stable cluster performance compared to conventional methods. These outcomes confirm that DT+APSO is a practical and effective solution for dynamic containerized cloud environments.

## 8.1 Conclusion

This work reports a Hybrid Decision Tree and Adaptive Particle Swarm Optimization (DT+APSO) approach for container load balancing in Docker Swarm clusters. Proposed scheduler reports to perform better than the Default Swarm placement strategy and a static PSO algorithm in term of resource allocation and efficiency. Study shows that it has achieved a large reduction in average CPU utilization as reported in Fig.11 and also that experiments have very low variance in CPU use across nodes which in turn indicates a more even load with no single node over utilized fig 1. Also noted that memory use with DT+APSO was more stable, which also did away with the spike and inconsistent use patterns research saw in the base methods The DPSO algorithm which used a different approach was able to better utilize available nodes which in turn reduced the number of unnecessary replicas for the work load thus improving resource use and performance of applications. The results shows of using a smart integration of decision tree modeling into an adaptive optimization strategy. The decision tree element put forth a very complex model of what term "cost" which in turn evaluated each host's performance for new containers based on present metrics. This model which in turn ran the swarm scheduler did so through a process of filtering and ranking nodes past simple heuristics and thus in the task of placing containers it did what it could to avoid resource contention and at the same time take into account dynamic conditions. Also the adaptive PSO element in the system tuned its parameters (like inertia weight and acceleration coefficients) in to the present state of the system. This adaptation which in turn brought about a fine balance between exploration and exploitation had the result that saw very quick convergence on to optimal or near optimal placement solutions as work load patterns changed. In practice the DT+APSO scheduler used live data from Prometheus and cAdvisor which in turn means that all decisions were based on current CPU and memory use data. Surely, experiments have achieved all of the above with the transparent inclusion of our solution in Docker Swarm which present through the orchestration API and does not require modification of the Swarm manager or Docker engine. Thus report that the in depth load balancing features may be put into production use without breaking present container orchestration processes.

## 8.2 Future Work

While the DT+APSO approach does very well in terms of CPU and memory based load balancing There is room for improvement. A good next step would be to apply this to multi objective optimization which in turn will allow the scheduler to look at not only CPU and memory but also network bandwidth, task latency and throughput at the same time. Future work will look at the implementation of the DT+APSO framework in Kubernetes which seen as the preeminent container orchestration platform. Also will look at including SLA aware scheduling which will see the system give priority to critical applications and at the same time see that response time and throughput requirements are met. Future work will look at the implementation of the DT+APSO framework in Kubernetes which is the preeminent container orchestration platform. Also can look at including SLA aware scheduling which will see the system give priority to critical applications and at the same time see that response time and throughput requirements are met. These in large part are what make the put forth approach practical, flexible and very much at home in real world cloud and edge settings. Experiments shows that they also enable the scheduler to do a better job of resource allocation, which in turn reduces waste, improves stability and does a better job of meeting operational requirements.

# 9 Copies of Papers published and a list of all publications arising from the thesis

As a part of the research work carried out under the present doctoral study, two research papers have been published in reputed Scopus-indexed journals. These publications reflect the progressive development of the research theme from a comparative analysis of Particle Swarm Optimization (PSO)-based load balancing methods in Docker container cloud environments to a more advanced hybrid framework integrating Particle Swarm Optimization with Decision Tree Classification for adaptive resource optimization. The published papers are directly related to the core objectives of the thesis and demonstrate the evolution of the proposed research methodology in the area of intelligent resource scheduling, load balancing, and optimization in containerized cloud computing environments. The copies of these papers are attached in the synopsis as supporting published work arising from the thesis.

The first published paper is titled "Load Balancing using Particle Swarm Optimization based Algorithms in Docker Container Cloud Environment: A Comparative Analysis" and was published in the International Journal of Intelligent Systems and Applications in Engineering, Volume 12, Issue 4, in the year 2024. The paper is authored by Manmitsinh Chandrasinh Zala and Jaykumar Shantilal Patel and is available through the DOI 10.17762/ijisae.v12i4.6533. This work focuses on the comparative evaluation of PSO-based algorithms for load balancing in Docker container cloud environments. The study emphasizes the implementation and analysis of multiple PSO variants using performance parameters such as CPU usage, memory usage, and optimized load allocation. The contribution of this paper lies in establishing a strong analytical and experimental foundation for PSO-driven load balancing in containerized infrastructures, which forms an important base for the present thesis work.

The second published paper is titled "Adaptive Resource Optimization in Containerized Environments Using Particle Swarm Optimization and Decision Tree Classification" and was published in the Journal of Information Systems Engineering and Management, Volume 9, Issue 4s, in 2024. The paper is authored by Manmitsinh Chandrasinh Zala and Jaykumar Shantilal Patel and was published on December 30, 2024, with DOI 10.52783/jisem.v9i4s.11666. This paper presents an advanced hybrid approach in which Particle Swarm Optimization is combined with Decision Tree Classification to achieve adaptive and predictive resource optimization in containerized cloud environments. In this work, PSO is used to explore optimal resource allocation solutions, while Decision Tree Classification is applied to discover patterns in historical resource utilization data for better predictive decision-making. This publication represents a significant advancement over the earlier comparative PSO study and aligns more closely with the proposed direction of the thesis by incorporating intelligent, adaptive, and data-driven optimization mechanisms.

These two publications collectively demonstrate the systematic progression of the research undertaken in the thesis. The first paper establishes the effectiveness and comparative behavior

of PSO-based load balancing algorithms in Docker-based cloud systems, while the second paper extends this research by introducing a hybrid intelligent optimization framework that improves adaptability and predictive resource management. Thus, the publications not only validate the research contributions already achieved but also provide documented evidence of the novelty and continuity of the work carried out under the thesis.

List of Publications Arising from the Thesis

1. Zala, Manmitsinh Chandrasinh, and Patel, Jaykumar Shantilal, "Load Balancing using Particle Swarm Optimization based Algorithms in Docker Container Cloud Environment: A Comparative Analysis," International Journal of Intelligent Systems and Applications in Engineering, Vol. 12, No. 4, 2024, DOI: 10.17762/ijisae.v12i4.6533

2. Zala, Manmitsinh Chandrasinh, and Patel, Jaykumar Shantilal, "Adaptive Resource Optimization in Containerized Environments Using Particle Swarm Optimization and Decision Tree Classification," Journal of Information Systems Engineering and Management, Vol. 9, No. 4s, 2024, Published on December 30, 2024, DOI: 10.52783/jisem.v9i4s.11666.

The above publications are the outcome of the research work carried out during the doctoral study and are closely aligned with the objectives, methodology, and expected contributions of the thesis. The published papers substantiate the originality and academic significance of the research in the domain of Docker container cloud environments, intelligent load balancing, and adaptive optimization using Particle Swarm Optimization and machine learning techniques. Copies of these published papers are enclosed as annexures to the synopsis for reference and verification

# References


[1] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. Autonomic vertical elasticity of docker containers with elasticdocker. In *IEEE International Conference on Cloud Computing, CLOUD*, volume 2017-June, pages 472–479. IEEE Computer Society, 9 2017.

[2] Gaia Ambrosino, Giovanni B. Fioccola, Roberto Canonico, and Giorgio Ventre. Container mapping and its impact on performance in containerized cloud environments. In *Proceedings - 14th IEEE International Conference on Service-Oriented System Engineering, SOSE 2020*, pages 57–64. Institute of Electrical and Electronics Engineers Inc., 8 2020.

[3] Satyajit. Chakrabarti and Himadri Nath. Saha. *IEEE CCWC-2017 : 2017 IEEE 7th Annual Computing and Communication Workshop and Conference : 09-11 January, 2017, Las Vegas, USA*. IEEE, 2017.

[4] Xinying Chen. Multi-Objective and Parallel Particle Swarm Optimization Algorithm for Container-Based Microservice Scheduling. 2021.

[5] Christophe Cérin, Tarek Menouer, Walid Saad, and Wiem Ben Abdallah. A new docker swarm scheduling strategy. In *Proceedings - 2017 IEEE 7th International Symposium on Cloud and Service Computing, SC2 2017*, volume 2018-January, pages 112–117. Institute of Electrical and Electronics Engineers Inc., 7 2017.

[6] Vitor Goncalves da Silva, Marite Kirikova, and Gundars Alksnis. Containers for virtualization: An overview. *Applied Computer Systems*, 23:21–27, 5 2018.

[7] Rajdeep Dua, A. Reddy Raja, and Dharmesh Kakadia. Virtualization vs containerization to support paas. In *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014*, pages 610–614. Institute of Electrical and Electronics Engineers Inc., 9 2014.

[8] Marischa Elveny, Ari Winata, Baihaqi Siregar, and Rahmad Syah. Ilkogretim online-elementary education online. 19:744–751, 2020.

[9] Gopinath P.P. Geethu and Shriram K. Vasudevan. An in-depth analysis and study of load balancing techniques in the cloud computing environment. In *Procedia Computer Science*, volume 50, pages 427–432. Elsevier B.V., 2015.

[10] Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. Load-balancing algorithms in cloud computing: A survey, 6 2017.

[11] Gerhard Goos. *Cyberspace Safety and Security*. 2019.

[12] Devki Nandan Jha, Saurabh Garg, Prem Prakash Jayaraman, Rajkumar Buyya, Zheng Li, and Rajiv Ranjan. A holistic evaluation of docker containers for interfering microservices. In *Proceedings - 2018 IEEE International Conference on Services Computing, SCC 2018 - Part of the 2018 IEEE World Congress on Services*, pages 33–40. Institute of Electrical and Electronics Engineers Inc., 9 2018.

[13] Scopus Journal and Ugc Journal. International journal of recent technology and engineering (ijrte). 2020.

[14] Lianwan Li, Jianxin Chen, and Wuyang Yan. A particle swarm optimization-based container scheduling algorithm of docker platform. *ACM International Conference Proceeding Series*, pages 12–17, 2018.

[15] Jialei Liu, Shangguang Wang, Ao Zhou, Jinliang Xu, and Fangchun Yang. Sla-driven container consolidation with usage prediction for green cloud computing. *Frontiers of Computer Science*, 14:42–52, 2 2020.

[16] René Peinl and Florian Holzschuher. The docker ecosystem needs consolidation. In *CLOSER 2015 - 5th International Conference on Cloud Computing and Services Science, Proceedings*, pages 535–542. SciTePress, 2015.

[17] David Reis, Bruno Piedade, Filipe F Correia, João Pedro Dias, and Ademar Aguiar. Developing Docker and Docker-Compose Specifications : A Developers ' Survey. 10, 2022.

[18] Weizheng Ren, Wenkai Chen, and Yansong Cui. Dynamic balance strategy of high concurrent web cluster based on docker container. In *IOP Conference Series: Materials Science and Engineering*, volume 466. Institute of Physics Publishing, 12 2018.

[19] Athanasia Tsertou, Angelos Amditis, Evangelia Latsa, Ioannis Kanellopoulos, and Michael Kotras. Dynamic and synchromodal container consolidation: The cloud computing enabler. In *Transportation Research Procedia*, volume 14, pages 2805–2813. Elsevier B.V., 2016.

[20] Zhang Wei-guo, Ma Xi-lin, and Zhang Jin-zhong. Research on kubernetes' resource scheduling scheme. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 11 2018.

[21] Libin Wu and Hongbin Xia. Particle Swarm Optimization Algorithm for Container Deployment. *Journal of Physics: Conference Series*, 1544(1), 2020.

[22] Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, and Cesar A.F. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *Proceedings of the 2013 21st Euromicro International*

*Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013*, pages 233–240, 2013.

[23] Yan Xu and Yanlei Shang. Dynamic priority based weighted scheduling algorithm in microservice system. In *IOP Conference Series: Materials Science and Engineering*, volume 490. Institute of Physics Publishing, 4 2019.

[24] Y U Xue, Bing Xue, and Mengjie Zhang. Self-Adaptive Particle Swarm Optimization for Large-Scale. 13(5), 2019.

[25] Muhammad Zakarya. Energy and performance aware resource management in heterogeneous cloud datacenters. Technical report, 2017.

[26] Zhi-hui Zhan and Jun Zhang. Adaptive Particle Swarm Optimization. (60573066):227–228, 2008.