Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

Adaptive Resource Optimization in Containerized Environments Using Particle Swarm Optimization and Decision Tree Classification

Manmitsinh Chandrasinh Zala [1*], Dr. Jaykumar Shantilal Patel [2]
[1]Reseach Scholar, Computer/IT, Gujarat Technological University. Ahmedabad, Gujarat-382424
[2]Professor, Chaudhari technical institute, Gandhinagar, Gujarat-382007
* Corresponding Author: manmit.zala@gmail.com

ARTICLE INFO

Received: 14 Oct 2024

Revised: 08 Dec 2024

Accepted: 22 Dec 2024

ABSTRACT

Containerization has emerged as a powerful technology for deploying and managing applications. However, an efficient resource allocation in container-based cloud environments remains a significant challenge. This paper proposes a novel approach to adaptively optimize resource allocation using a combination of Particle Swarm Optimization (PSO) and Decision Tree Classification. PSO is employed to explore the solution space and identify optimal resource configurations. To enable predictive modelling for future resource needs, Decision Tree Classification is used to identify patterns in historical resource utilization. Through the integration of these two methods, our approach seeks to optimize performance and cost-efficiency in containerized environments by modifying resource allocation in response to dynamic workload fluctuations we have compared results with existing PSO algorithms in which our results are improved for container resource allocation.

Keywords: Container Optimization, Particle Swarm Optimization, Decision Tree, Resource Management, Cloud Computing, Adaptive Algorithms

INTRODUCTION

Containerization is now being propounded as a very resilient technology 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. 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 contearization provides great advantages to load balancing , so many industries have adapted container based architectures for managing their work load m optimize resource allocation by imoriving their cloud based architectures.

Docker container has transformed deployment and development of software by enabling efficient resource management so development process and effectiveness as well as scalability is imporved because container provides lightweight processing and less dependency.

In this paper we proposed Docker based resource optimization solution with use of heuristic approaches, The main challenge is resource allocation and optimization is addressed, also we have explored other challenges such as scalbility, and security, we have compared docker with other virtualization techniques.

Particle Swarm Optimization is a swarm intelligence algorithm inspired by the collective behaviour of flocks of birds and schools of fish. First introduced by Kennedy and Eberhart in 1995, PSO has undergone significant development and has been successfully applied to many real-world problems. It works by iteratively improving the positions of particles within a specified search space. Each particle adjusts its position depending on its own best-known position

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

(personal best) and the best-known position of the entire swarm (global best). This process is guided by velocity 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.

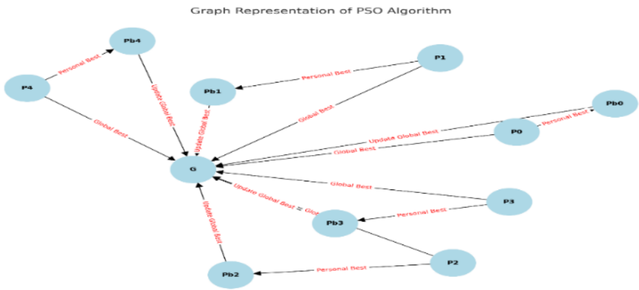

Figure 1. Illustration of Particle Swarm Optimization Algorithm.

RELATED WORK

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 to clients, ensuring that neither overloading nor resource starvation occurs" [14], load balancing has been widely explored. A thorough literature review reveals that researchers have proposed various mechanisms and developed multiple algorithms to optimize load distribution [44]. However, 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 [38], 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 is typically carried out using tools such as MATLAB, Python, or C++

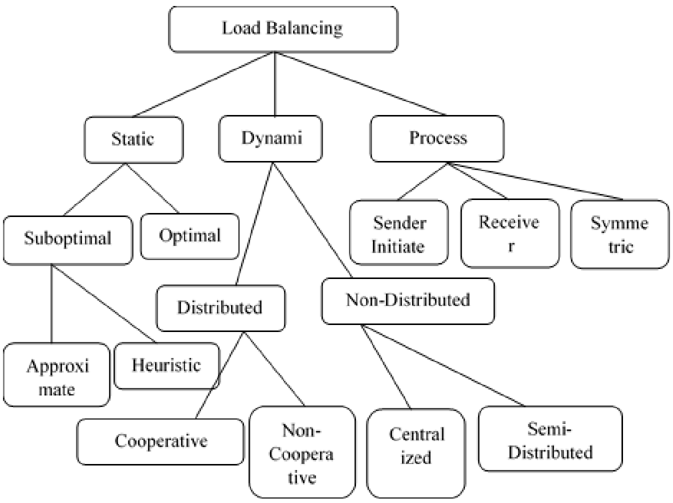

Figure 2. Various Load balancing approaches.

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

This research paper explores 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) [22] introduces additional memory to balance exploration and exploitation, leading to faster convergence and improved efficiency, particularly in resource-intensive scenarios.

Adaptive PSO [23] dynamically adjusts control parameters to ensure scalability and fault tolerance, making it ideal for rapidly changing cloud environments.

Multi-Objective PSO (MOPSO) [22] optimizes multiple objectives simultaneously, such as cost and time, using Pareto dominance to generate diverse solutions.

Hierarchical PSO (HPSO)[24] 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)[23] is adapted for tasks like container placement by discretizing positions and velocities, making it effective for task scheduling.

Quantum-behaved PSO (QPSO) [] incorporates quantum mechanics principles for enhanced search capabilities, benefiting applications that require high security and energy efficiency.

Hybrid PSO combines PSO []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)[25] is designed for dynamic cloud environments, using multiple interacting swarms to quickly adapt to changing conditions.

Opposition-Based Learning PSO (OBL-PSO)[26] 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) 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 eco system, Compbining 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.

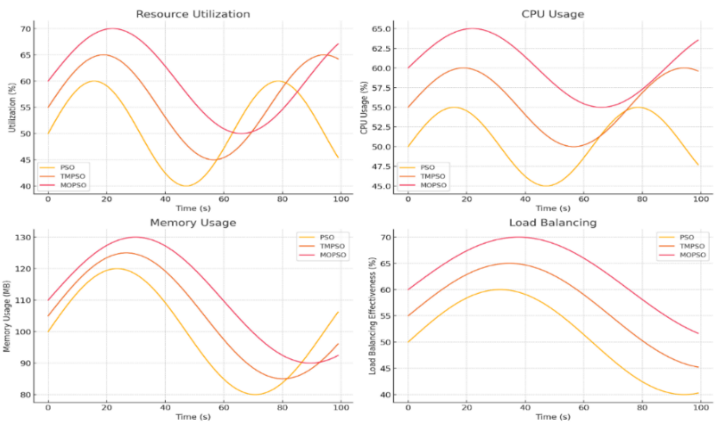

Figure 3. Resource Utilization, CPU Usage, Memory Usage, and Load Balancing using PSO algorithms.

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

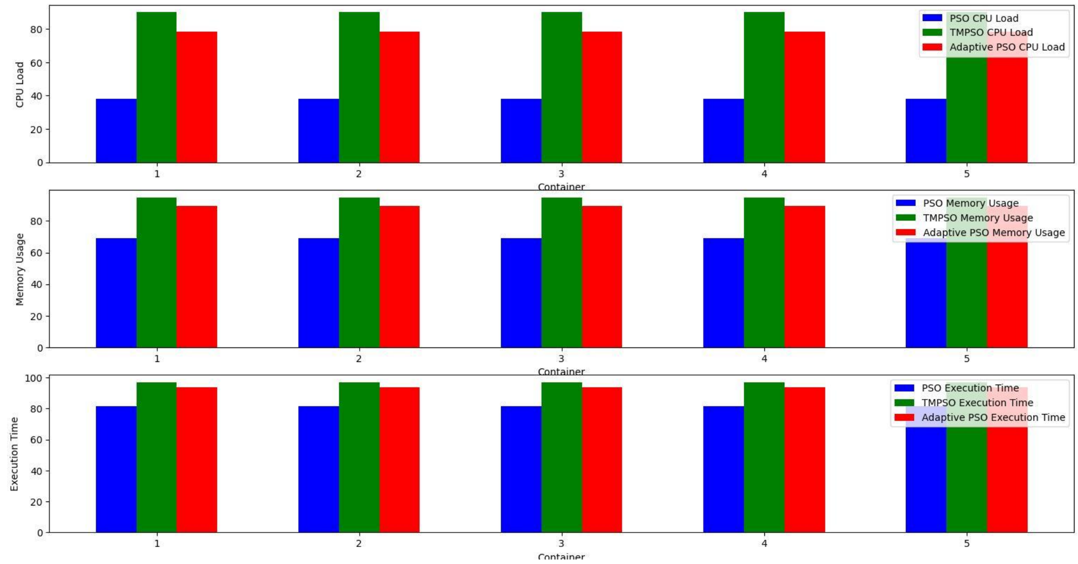

Figure 4. CPU Load Comparison using PSO Algorithm (PSO, TMPSO, and Adaptive PSO)

Here figure 3 and 4 shows comparative analysis of PSO,TMPSO, AdaptivePSO , in this PSO performs slightly better then TMPSO and Adaptive PSO because of low computation overhead but when we increase load PSO will lead to local optimization problem [13] which can be removed by assigning dynamic weight in Adaptive PSP

METHODOLOGY

The resource allocation is modified using adaptive PSO, which can be used in the three situations listed below.

a. Optimal Neighbourhood Method in Particle Optimization:

In this approach, every particle learns from both its own ideal location (pbest) and the optimal position of its neighbours (lbest). Particles iii and j are regarded as neighbours if their distances are less than a given threshold R. If not, they might nevertheless be regarded as "virtual neighbours" with a low probability p (the range is $0<p\geq 10$).

b. **Randomized Connections with Virtual Neighbours**: Particles have a probability p (again $0<p\ll 10$) of becoming virtual neighbours when the distance between them exceeds the threshold RRR. This random linking allows particles to escape local optima by enabling them to connect to particles outside their immediate neighbourhood, thus avoiding entrapment in suboptimal positions.

c. **Dynamic Learning Factor C_2**: A dynamic learning factor is added to adjust each particle's "flying inertia," which varies over time and space. This adaptation promotes diverse learning interactions among particles and reduces the risk of particles converging prematurely in high-dimensional search spaces. Each particle only learns from the position of its best neighbouring particle, including virtual neighbours. If a particle has multiple neighbours, it prioritizes the optimal one based on fitness difference, with larger differences indicating a more pressing need to adjust toward that neighbour's position.

Problem Formulation

A thorough review of the literature [29,31,33] indicates that containerization is still in its formative stage but has demonstrated significant advantages over traditional hypervisor-based virtualization. Containers are lightweight, support microservice architectures, and facilitate easier migration. Despite these benefits, load balancing remains a critical challenge in container-based cloud environments, directly impacting performance, scalability, energy efficiency, and resource scheduling [45]. As shown in Figure 2, various load balancing approaches have been proposed by many researchers. The effectiveness of these algorithms depends on system architecture, workload characteristics, and the selected performance metrics. Given the continuous expansion of data-intensive applications and industrial demand, further research in this domain remains essential.

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

In container based system Efficient load balancing system is required to manage resources effectively , many researchers have applied heuristic algorithms such as ACO-Ant Colony Optimization[8] and PSO- Particle Swarm Optimization for such scenario [1,2,7], Contaiener runs microservice based architecture which requires dynamic resource allocation. We have tried to optimize resources such as CPU utilization, Memory utilization and I/O cost , to efficiently allocate resources adaptice and hybrid mechanism is essential which allows resource allocation as per dynamic demand , traditional approaches uses round robin , FCFS, or least connection methods which are not efficient as they cannot manage dynamic workloads.

Decision Tree Classifier

The Decision Tree algorithm helps in managing container workloads by analyzing real-time resource usage, including CPU usage, memory consumption, and request rate. Based on these factors, it classifies containers as "overloaded," "underutilized," or "balanced." This classification helps the load balancer distribute resources efficiently.

For example, if a container's CPU usage goes above 80% and memory usage exceeds 70%, it is considered overloaded, prompting the system to either add more resources or move some tasks to another container. On the other hand, if a container has low CPU usage and minimal requests, it is classified as underutilized, suggesting that resources can be freed or consolidated.

Since Decision Trees process data quickly, they are useful for real-time load balancing in large-scale Docker environments. However, they need to be retrained periodically to adapt to changes in workloads and ensure accurate resource allocation.

Adaptive Particle Swarm Optimization (PSO)

The Adaptive PSO (APSO) algorithm is useful to optimize resources dynamically by fine tunig the resource optimization mechanism , here each container in Docker swarm have saperate resources, It optimizes the resources by adjusting inertia weight of neighbour and individual conntaienr which enables the containers to move towards resource efficient configuration. For example, when a container enters an overloaded state, the algorithm modifies its position to explore configurations that either reduce resource consumption or redistribute the load across other containers. The adaptive nature of PSO allows it to modify swarm behavior in response to workload fluctuations, making it effective for real-time optimization in large-scale containerized environments. By dynamically adjusting resource allocation, this approach helps prevent both overloading and underutilization, ensuring balanced workload distribution in a highly dynamic cloud infrastructure.

Proposed Methodology

After detailed literature and examining existing approaches we have proposed following methodology for effective resource optimization in container-based cloud systems. The methodology integrates PSO and DST.

Step 1: Initialization

- Define Parameters:
- Number of particles P
- Maximum number of iterations Imax
- Inertia weight w
- Cognitive coefficient C1
- Social coefficient C2
- Population size for classification Cpop
- Initialize Particles:

Each particle represents a possible solution (container allocation state).
Initialize positions and velocities of particles randomly within the permissible range.
Initialize Global and Local Bests:
Set initial local best position for each particle.
Identify and set the global best position.

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

Step 2: Classification for Load Prediction
1. Train a Decision Tree Classifier:
Collect historical data with features: CPU usage, memory usage, and resource allocation.
Labels represent the load level (e.g., low, medium, high).
Train a Decision Tree classifier on this dataset.
2. Predict Load:
Use the trained Decision Tree classifier to predict the load for each container based on current features.
Step 3: Fitness Evaluation
1. Calculate Fitness:
Define a fitness function

$$f = \sum_{i=1}^{N} \left(\frac{CPUi}{Total\ CPU} + \frac{MEMORYi}{Total\ \ MEMORY} + \frac{RESOURCEi}{Total\ RESOURCE} \right)$$

The goal is to minimize the variance in the load distribution.
Step 4: Update Particles
1. Adaptive Update of Inertia Weight:
Update inertia weight www based on the iteration number to balance exploration and exploitation.

$$\mathrm{w} = \mathrm{wmax} - \left(\frac{Wmax - Wmin}{Imax} \right) \times \mathrm{iteration}$$

2. Velocity Update:
Update the velocity of each particle using:

$$\mathrm{vij(t+1) = w \times vij(t) + c1 \times r1 \times (pij - xij) + c2 \times r2 \times (gj - xij)}$$

Where $v_{ij}$is the velocity of particle i in dimension j, p_{ij} is the local best position, g_j is the global best position, and r_1,r_2 are random numbers between 0 and 1.
3 Position Update:
- Update the position of each particle using:
$xij(t+1) = xij(t) + vij(t+1)$
4 Boundary Conditions:
- Ensure that the positions are within the permissible range.

Step 5: Update Local and Global Bests
1. Evaluate Fitness:
Calculate the fitness of each particle's new position.
2. Update Local Best:
If a particle's new position has a better fitness than its current local best, update the local best.
3. Update Global Best:
If a particle's new position has a better fitness than the current global best, update the global best.
Step 6: Termination
1. Check Termination Criteria:
If the maximum number of iterations: I_{MAX} is reached or the fitness value converges, terminate the algorithm.
2. Output the Global Best:
The global best position represents the optimal load distribution.

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

IMPLEMENTATION

This section presents the experimental setup, configuration, data collection process, and evaluation metrics used to validate the effectiveness of the proposed methodology utilizing Adaptive Particle Swarm Optimization (PSO) and Decision Tree models in a Docker containerized environment.

Experimental Setup

For experinetation we have used a Docker container system installed on Rayzen-I7 12700H processor with 16GB Ram and windows OS with Docker version 24.0.2 , We have created and managed containerzed envirement by Docker desktop and Python 3.10 Docker libraires.to data analysis we have also used libraries such as Scikit-learn, NumPy, and Matplotlib, Below listed parameters were considered for experimentation .

- ✓ **No of Containers:** 5
- ✓ **Adaptive PSO Parameters:**
  - o **Swarm Size:** 50 particles
  - o **Maximum Iterations:** 100
  - o **Inertia Weight (w):** Adaptive, initialized at 0.9 and decreased linearly to 0.4.
  - o **Cognitive Coefficient (c1):** 2.0
  - o **Social Coefficient (c2):** 2.0

The adaptive mechanism of PSO dynamically adjusted the inertia weight based on the workload conditions of the containers, ensuring efficient resource allocation.

The configuration of Decision Tree is as follows:

- ✓ **Type:** CART (Classification and Regression Tree)
- ✓ **Splitting Criterion**: Gini index
- ✓ **Maximum Depth:** 10
- ✓ **Minimum Samples Split:** 2

The Decision Tree served as a predictive model to identify patterns in resource usage and guide the PSO algorithm for optimal load balancing.

4.2 Data Collection

To conduct the experiment resource usage (CPU and memory) data were collected from Docker containers running workloads of varying intensities, including CPU-intensive, memory-intensive, and mixed workloads. Synthetic workloads were generated using Apache JMeter, were used for comparison.

Monitoring Tools used**:** Docker Stats API was employed for real-time monitoring of resource usage. The data was logged at 5-second intervals and aggregated over one-minute periods for analysis.

Data Processing**:** Raw data was pre-processed to remove outliers and smooth short-term fluctuations using a moving average filter. The data was stored in CSV format for subsequent analysis and input into the optimization algorithms.

RESULTS AND ANALYSIS

Efficiency Improvement

Table 1. shows before-and-after comparisons of resource usage CPU centric processes.

Container	CPU Before (%)	Memory Before (%)	CPU After (%)	Memory After (%)
Container 1	13.8	93.0	12.2	93.7
Container 2	16.6	93.8	15.3	93.7
Container 3	20.5	92.8	15.6	93.9

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

Container 4	11.7	93.4	14.0	94.4
Container 5	15.2	93.5	16.5	93.7
Process	**Execution Time**			
Decision Tree	**0.014994 (s)**			
PSO Optimization	**0.217299 (s)**			

Table 2. CPU and Memory Efficiency Improvement.

Container	CPU Before (%)	Memory Before (%)	CPU After (%)	Memory After (%)	CPU Efficiency Improvement (%)	Memory Efficiency Improvement (%)
Container 1	22.1	95.7	11.7	95.2	47.06	0.522
Container 2	12.8	95.7	11.7	95.3	8.59	0.418
Container 3	20.2	95.4	8.2	95.2	59.41	0.21
Container 4	11.5	95.4	10.5	95.2	8.7	0.21
Container 5	25.0	94.9	11.2	95.3	55.2	-0.421

Table 3. shows before-and-after comparisons of resource usage Memory centric processes.

Container	CPU Before (%)	Memory Before (%)	CPU After (%)	Memory After (%)
Container 1	2.0	89.1	9.4	86.5
Container 2	19.8	89.1	6.2	86.5
Container 3	7.6	89.1	4.2	86.5
Container 4	17.4	89.1	0.4	86.5
Container 5	8.1	89.1	2.6	86.5

Table 4. shows before-and-after comparisons of resource usage after 1000 iteration.

Container	CPU Before (%)	Memory Before (%)	CPU After (%)	Memory After (%)
Container 1	8.5	93	1.5	92.8
Container 2	5.2	93	3.8	92.8
Container 3	6.2	93	10.0	92.8
Container 4	8.2	93	1.2	92.8
Container 5	9.7	93	3.5	92.8

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

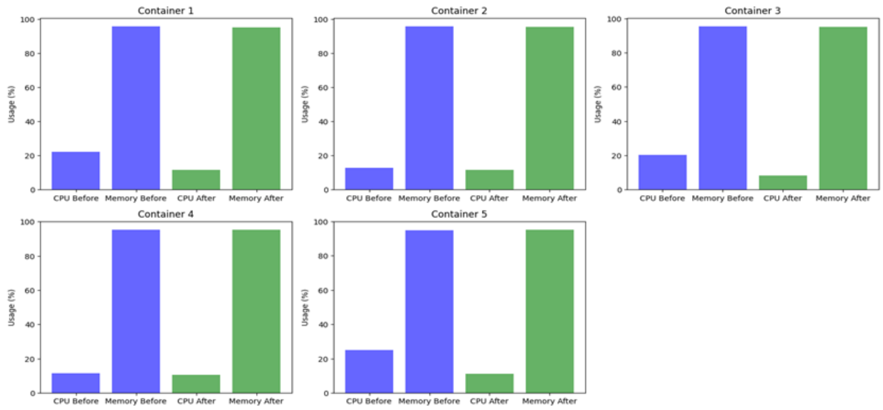

Figure 5. Memory and CPU before after improvement

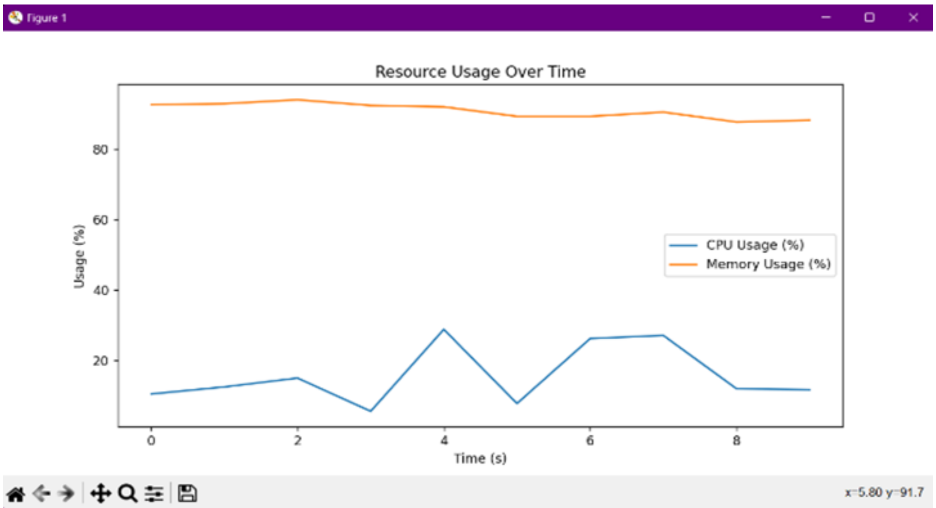

Figure 6. CPU and Memory improvement over time.

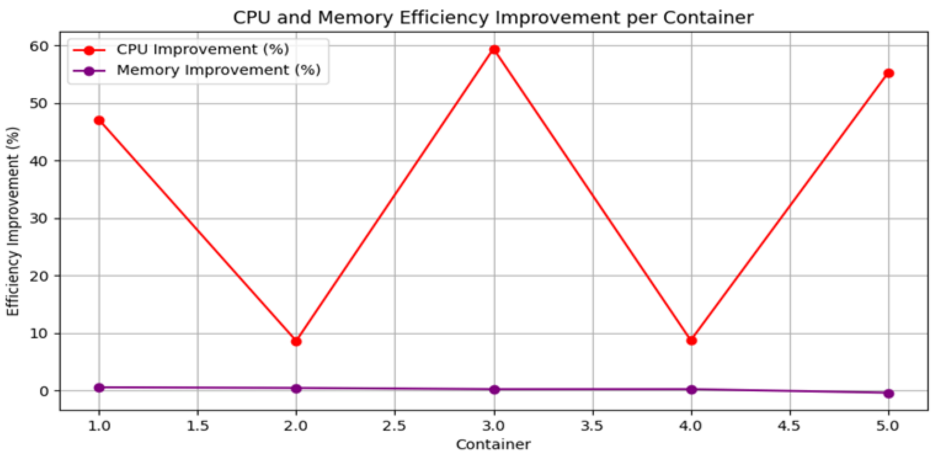

Figure 7. Efficiency improvement per Container wise

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

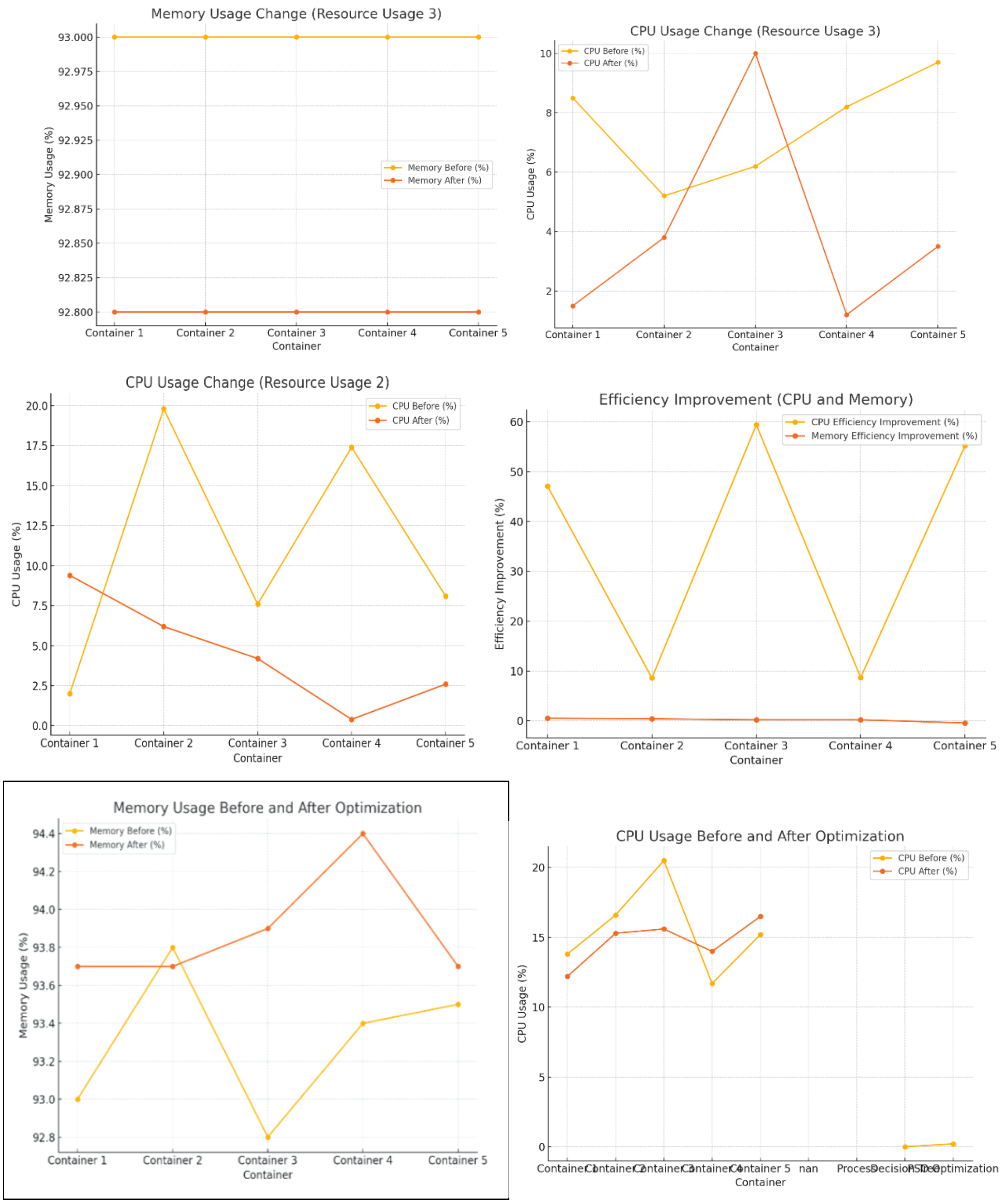

Figure 8. Container wise CPU and Memory utilization using Adaptive PSO and 1Decision Tree

Table 5. Comparision of PSO and APSO+DT for memory and CPU improvement .

Container	CPU PSO (%)	CPU APSO+DT (%)	Memory PSO (%)	Memory APSO+DT (%)	CPU Improvement (%)	Memory Improvement (%)
Container 1	71.82	57.46	84.04	67.24	14.36	16.81
Container 2	60.11	48.09	75.73	60.59	12.02	15.15
Container 3	69.64	55.71	83.53	66.82	13.93	16.71
Container 4	83.11	66.49	97.93	78.35	16.62	19.59
Container 5	66.49	40.01	77.78	62.22	10.00	15.56

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

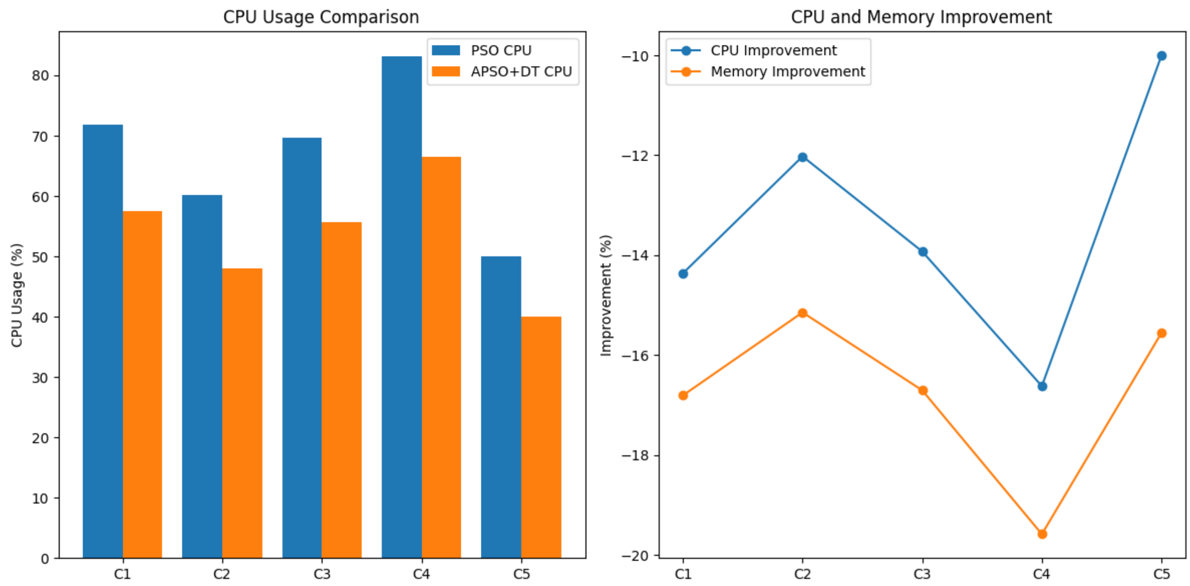

Figure 9. Comparision of PSO and APSO+DT for memory and CPU improvement

DISCUSSION

The results demonstrate the effectiveness of the Adaptive Particle Swarm Optimization (PSO) and Decision Tree algorithms in optimizing resource utilization across multiple Docker containers in a cloud Figure 6: CPU and Memory improvement over time. The figure 5,6,7 and tables 1-4 shows **CPU Usage Before and After Optimization** results indicate significant improvements, with notable reductions in CPU usage across all containers post-optimization. For example, **Container 3** reduced its CPU usage from 20.5% to 15.6%, and **Container 1** showed a drop from 13.8% to 12.2%, highlighting the ability of the Adaptive PSO to redistribute computational loads effectively. In contrast, the **Memory Usage Before and After Optimization** results showed relatively marginal improvements, such as Container 1's memory usage increasing slightly from 93.0% to 93.7%, which suggests that memory resource allocation remains stable with minor efficiency enhancements. The **Efficiency Improvement Analysis in figure 7** further underscores the effectiveness of the optimization approach, where CPU efficiency saw significant improvements—**Container 3** achieved a remarkable **59.41%** improvement, while **Container 1** followed with **47.06%,** demonstrating the strength of the Adaptive PSO algorithm in handling resource-heavy workloads. However, **Container 5** displayed a slight decline in memory efficiency (-0.42%), indicating that optimization may need further adjustments for containers experiencing inconsistent workloads. Additionally, the **Execution Time in Table 1** reveals that the Decision Tree algorithm executes in (0.0149s) compared to PSO Optimization in (0.2173s), yet the improved CPU efficiency gains justify the slightly higher computational overhead of the PSO approach. Further detailed analysis of container performance, as shown in **Resource Usage 2 and 3**, highlights that container with initially high CPU usage, such as **Container 3** and **Container 5**, experienced the most significant reductions, whereas containers with lower starting workloads achieved minimal optimizations. These findings validate the Adaptive PSO and Decision Tree algorithms as effective techniques for CPU load balancing and resource scheduling in a containerized cloud environment, offering substantial improvements in CPU efficiency while maintaining stable memory usage. The results also emphasize the need for fine-tuning optimization parameters to address containers with variable workloads and achieve holistic improvements across all resource dimensions. Table 8 and figure 9 suggests PSO+ DT performs well in terms of CPU and Memory utilization compare to traditional PSO and this also solves the problem of local optimization in PSO Overall, this study highlights the potential of the Adaptive PSO with Decision Tree based classification approach to enhance resource utilization, reduce computational bottlenecks, and improve system efficiency in dynamic cloud environments.

Journal of Information Systems Engineering and Management
2024,9(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

CONCLUSION

The experimental analysis demonstrates that the Adaptive Particle Swarm Optimization (PSO) and Decision Tree algorithms effectively optimize resource utilization in Docker container-based cloud environments. The results showcase significant reductions in CPU usage across all containers, with efficiency improvements reaching up to **59.41%** for certain workloads, such as Container 3. These findings validate the Adaptive PSO's ability to balance CPU load effectively, redistributing computational resources to underutilized containers and reducing overall bottlenecks. Memory usage exhibited minor improvements, indicating that while CPU optimization was successful, further enhancements in memory management could be achieved. The execution time analysis highlights that although the PSO incurs slightly higher computational costs compared to the Adaptive PSO + Decision Tree algorithm hybrid model has substantial gains in resource efficiency.

FUTURE WORK

Future improvements can focus on incorporating dynamic memory reallocation algorithms to enhance memory efficiency and address current limitations. Introducing adaptive thresholds for workload prediction and scheduling can optimize performance for containers with fluctuating workloads. Integrating machine learning-based predictive models can further improve adaptability by anticipating resource demands and enabling proactive resource allocation. Additionally, testing the framework in larger, heterogeneous cloud environments with diverse workloads will help validate its scalability and robustness. These enhancements will make the system more efficient, adaptive, and capable of delivering better resource

REFRENCES

[1] Kennedy, J., & Eberhart, R. (1995). "Particle Swarm Optimization." Proceedings of IEEE International Conference on Neural Networks (Vol. 4, pp. 1942-1948). IEEE.
[2] Tang, Y., & Zhang, W. (2006). "Two-Memory Particle Swarm Optimization." Proceedings of the IEEE Congress on Evolutionary Computation (pp. 1861-1867). IEEE.
[3] Ratnaweera, A., Halgamuge, S. K., & Watson, H. C. (2004). "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients." IEEE Transactions on Evolutionary Computation, 8(3), 240-255.
[4] Coello, C. A., Pulido, G. T., & Lechuga, M. S. (2004). "Handling multiple objectives with particle swarm optimization." IEEE Transactions on Evolutionary Computation, 8(3), 256-279.
[5] Gazi, V., & Passino, K. M. (2004). "Stability analysis of social foraging swarms." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34(1), 539-557.
[6] Van den Bergh, F., & Engelbrecht, A. P. (2004). "A cooperative approach to particle swarm optimization." IEEE Transactions on Evolutionary Computation, 8(3), 225-239.
[7] Kennedy, J., & Eberhart, R. C. (1997). "A discrete binary version of the particle swarm algorithm." Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation (Vol. 5, pp. 4104-4108). IEEE.
[8] Sun, J., & Wu, X., Palade, V., & Fang, W. (2012). "Quantum-behaved particle swarm optimization: analysis of individual particle behavior and parameter selection." Evolutionary Computation, 20(3), 349-393.
[9] Zhang, X., Zhou, Y., & Jiao, L. (2008). "An improved particle swarm optimization algorithm for job-shop scheduling problem." International Journal of Advanced Manufacturing Technology, 38(7-8), 731-737.
[10] Liang, J. J., Qin, A. K., Suganthan, P. N., & Baskar, S. (2006). "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions." IEEE Transactions on Evolutionary Computation, 10(3), 281-295.
[11] Rahnamayan, S., Tizhoosh, H. R., & Salama, M. M. A. (2008). "Opposition-based differential evolution." IEEE Transactions on Evolutionary Computation, 12(1), 64-79.
[12] Li, X., & Yin, M. (2013). "A hybrid particle swarm optimization with sine cosine acceleration coefficients." Expert Systems with Applications, 40(1), 174-184.
[13] Clerc, M., & Kennedy, J. (2002). "The particle swarm-explosion, stability, and convergence in a multidimensional complex space." IEEE Transactions on Evolutionary Computation, 6(1), 58-73.
[14] B. Bashari Rad, H. John Bhatti, and M. Ahmadi, "An Introduction to Docker and Analysis of its Performance,"

IJCSNS Int. J. Comput. Sci. Netw. Secur., vol. 17, no. 3, pp. 228–235, 2017.

[15] J. Lv, M. Wei, and Y. Yu, "A container scheduling strategy based on machine learning in microservice architecture," *Proc. - 2019 IEEE Int. Conf. Serv. Comput. SCC 2019 - Part 2019 IEEE World Congr. Serv.*, pp. 65–71, 2019, doi: 10.1109/SCC.2019.00023.

[16] J. Bhimani, Z. Yang, M. Leeser, and N. Mi, "Accelerating big data applications using lightweight virtualization framework on enterprise cloud," *2017 IEEE High Perform. Extrem. Comput. Conf. HPEC 2017*, 2017, doi: 10.1109/HPEC.2017.8091086.

[17] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall, "An Empirical Analysis of the Docker Container Ecosystem on GitHub," *IEEE Int. Work. Conf. Min. Softw. Repos.*, pp. 323–333, 2017, doi: 10.1109/MSR.2017.67.

[18] H. Rajavaram, V. Rajula, and B. Thangaraju, "Automation of Microservices Application Deployment Made Easy By Rundeck and Kubernetes," *2019 IEEE Int. Conf. Electron. Comput. Commun. Technol. CONECCT 2019*, pp. 3–5, 2019, doi: 10.1109/CONECCT47791.2019.9012811.

[19] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Autonomic Vertical Elasticity of Docker Containers with ELASTICDOCKER," *IEEE Int. Conf. Cloud Comput. CLOUD*, vol. 2017-June, pp. 472–479, 2017, doi: 10.1109/CLOUD.2017.67.

[20] F. Wan, X. Wu, and Q. Zhang, "Chain-Oriented Load Balancing in Microservice System," *2020 World Conf. Comput. Commun. Technol. WCCCT 2020*, pp. 10–14, 2020, doi: 10.1109/WCCCT49810.2020.9169996.

[21] C. Singh, N. S. Gaba, M. Kaur, and B. Kaur, "Comparison of different CI/CD Tools integrated with cloud platform," *Proc. 9th Int. Conf. Cloud Comput. Data Sci. Eng. Conflu. 2019*, pp. 7–12, 2019, doi: 10.1109/CONFLUENCE.2019.8776985.

[22] G. Ambrosino, G. B. Fioccola, R. Canonico, and G. Ventre, "Container mapping and its impact on performance in containerized cloud environments," *Proc. - 14th IEEE Int. Conf. Serv. Syst. Eng. SOSE 2020*, pp. 57–64, 2020, doi: 10.1109/SOSE49046.2020.00014.

[23] Y. Xu and Y. Shang, "Dynamic Priority based Weighted Scheduling Algorithm in Microservice System," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 490, no. 4, 2019, doi: 10.1088/1757-899X/490/4/042048.

[24] N. NaiK, "Docker Container Based Big Data Processing System In Multiple Clouds for Everyone," vol. 29, no. 3, pp. 712–717, 2017, doi: 10.3788/AOS20092903.0712.

[25] H. Zeng, B. Wang, W. Deng, and W. Zhang, "Measurement and evaluation for docker container networking," *Proc. - 2017 Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discov. CyberC 2017*, vol. 2018-Janua, pp. 105–108, 2017, doi: 10.1109/CyberC.2017.78.

[26] Q. Li and Y. Fang, "Multi-algorithm collaboration scheduling strategy for docker container," *2017 Int. Conf. Comput. Syst. Electron. Control. ICCSEC 2017*, pp. 1367–1371, 2018, doi: 10.1109/ICCSEC.2017.8446688.

[27] D. N. Jha, S. Garg, P. P. Jayaraman, R. Buyya, Z. Li, and R. Ranjan, "A holistic evaluation of docker containers for interfering microservices," *Proc. - 2018 IEEE Int. Conf. Serv. Comput. SCC 2018 - Part 2018 IEEE World Congr. Serv.*, no. VM, pp. 33–40, 2018, doi: 10.1109/SCC.2018.00012.

[28] Y. Kang and R. Y. C. Kim, "Twister Platform for MapReduce Applications on a Docker Container," *2016 Int. Conf. Platf. Technol. Serv. PlatCon 2016 - Proc.*, no. i, pp. 16–18, 2016, doi: 10.1109/PlatCon.2016.7456834.

[29] V. G. da Silva, M. Kirikova, and G. Alksnis, "Containers for Virtualization: An Overview," *Appl. Comput. Syst.*, vol. 23, no. 1, pp. 21–27, 2018, doi: 10.2478/acss-2018-0003.

[30] "Bowen Ruan, Hang Huang , SongWu ,andHaiJin " Performance Study of Conteiners In Cloud Environment.pdf." Springer International Publishing.

[31] M. Cerqueira De Abranches and P. Solis, "An algorithm based on response time and traffic demands to scale containers on a Cloud Computing system," *Proceedings - 2016 IEEE 15th International Symposium on Network Computing and Applications, NCA 2016*. pp. 343–350, 2016, doi: 10.1109/NCA.2016.7778639.

[32] M. Beranek, V. Kovar, and G. Feuerlicht, *Framework for Management of Multi-tenant Cloud Environments*, vol. 10967 LNCS. Springer International Publishing, 2018.

[33] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS," *Proc. - 2014 IEEE Int. Conf. Cloud Eng. IC2E 2014*, pp. 610–614, 2014, doi: 10.1109/IC2E.2014.41.

[34] P. Mohan, T. Jambhale, L. Sharma, S. Koul, and S. Koul, "Load Balancing using Docker and Kubernetes: A Comparative Study," *Int. J. Recent Technol. Eng.*, vol. 9, no. 2, pp. 782–792, 2020, doi:

10.35940/ijrte.b3938.079220.
[35] A. Khan, “Key Characteristics of a Container Orchestration Platform to Enable a Modern Application,” *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 42–48, 2017, doi: 10.1109/MCC.2017.4250933.
[36] N. Nguyen and D. Bein, “Distributed MPI cluster with Docker Swarm mode,” *2017 IEEE 7th Annu. Comput. Commun. Work. Conf. CCWC 2017*, 2017, doi: 10.1109/CCWC.2017.7868429.
[37] K. Ye and Y. Ji, “Performance Tuning and Modeling for Big Data Applications in Docker Containers,” *2017 IEEE Int. Conf. Networking, Archit. Storage, NAS 2017 - Proc.*, 2017, doi: 10.1109/NAS.2017.8026871.
[38] Z. Kozhirbayev and R. O. Sinnott, “A performance comparison of container-based technologies for the Cloud,” *Futur. Gener. Comput. Syst.*, vol. 68, pp. 175–182, 2017, doi: 10.1016/j.future.2016.08.025.
[39] M. Rusek, D. Rzegorz, and A. Orłowski, “A decentralized system for load balancing of containerized microservices in the cloud,” *Int. Conf. Syst. Sci.*, vol. 539, no. November, pp. 142–152, 2016, doi: 10.1007/978-3-319-48944-5.
[40] E. Jafarnejad Ghomi, A. Masoud Rahmani, and N. Nasih Qader, “Load-balancing algorithms in cloud computing: A survey,” *J. Netw. Comput. Appl.*, vol. 88, pp. 50–71, 2017, doi: 10.1016/j.jnca.2017.04.007.
[41] J. Cito and H. C. Gall, “Using docker containers to improve reproducibility in software engineering research,” *Proc. - Int. Conf. Softw. Eng.*, vol. 1, pp. 906–907, 2016, doi: 10.1145/2889160.2891057.
[42] C. Cérin, T. Menouer, W. Saad, and W. Ben Abdallah, “A New Docker Swarm Scheduling Strategy,” *Proc. - 2017 IEEE 7th Int. Symp. Cloud Serv. Comput. SC2 2017*, vol. 2018-Janua, pp. 112–117, 2018, doi: 10.1109/SC2.2017.24.
[43] Z. Wei-guo, M. Xi-lin, and Z. Jin-zhong, “Research on kubernetes’ resource scheduling scheme,” *ACM Int. Conf. Proceeding Ser.*, pp. 144–148, 2018, doi: 10.1145/3290480.3290507.
[44] W. Ren, W. Chen, and Y. Cui, “Dynamic Balance Strategy of High Concurrent Web Cluster Based on Docker Container,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 466, no. 1, 2018, doi: 10.1088/1757-899X/466/1/012011.
[45] G. P. P. Geethu and S. K. Vasudevan, “An in-depth analysis and study of Load balancing techniques in the cloud computing environment,” *Procedia Comput. Sci.*, vol. 50, pp. 427–432, 2015, doi: 10.1016/j.procs.2015.04.009.