ISSN: 2093-5374 / E-ISSN: 2093-5382

# Container Load Placement for Deep Learning Application Using Whale Optimization

Taufiq Odhi Dwi Putra[1*], Royyana Muslim Ijtihadie[2], and Tohari Ahmad[3*]

[1*]Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia.
6025221005@student.its.ac.id, https://orcid.org/0009-0000-6456-1737

[2]Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia.
roy@its.ac.id, https://orcid.org/0000-0001-7168-1235

[3*]Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia.
tohari@its.ac.id, https://orcid.org/0000-0002-3390-0756

Received: February 15, 2024; Revised: March 30, 2024; Accepted: May 10, 2024; Published: June 29, 2024

**Abstract**

The deployment and scaling of deep learning applications in distributed computing environments pose significant challenges, particularly in the context of containerized virtualization. Efficient placement and management of Docker containers are critical to optimizing resource utilization, minimizing latency, and ensuring the scalability of deep learning models across clusters of machines. In this paper, we will compare five methods of container placement that are implemented within a scheduling method named Differentiate Quality of Experience Scheduling (DQoES). The container placement methods to be compared include the default Docker Swarm container placement, Discrete Whale Optimization Container Placement (DOWCP), proposed whale optimization, a proposed hybrid with DWOCP, and a proposed hybrid with proposed whale optimization. Based on the experimental results, the method that demonstrates better performance than both the default Docker Swarm container placement and DWOCP is the proposed hybrid with proposed whale optimization.

**Keywords:** Application, Cloud Computing, Container Placement, Deep Learning, Task Scheduling.

## 1 Introduction

Due to the rapid development of deep learning applications, the need for GPU resources has increased rapidly. Deep learning applications development has become an important part for advancements in various sectors, for example automated customer service systems in medical diagnostics (Marakala et al., 2022), image super sampling to decrease render scale or input image file (Sarker, 2021). These applications rely heavily on deep learning models, which are computationally intensive and necessitate robust cloud-based resources for efficient operation. However, a critical challenge emerges in the form of efficiently scheduling (Ahmad et al., 2022) these deep learning tasks within cloud environments. This challenge is not solely a matter of computational resource allocation but also involves optimizing the scheduling (Hu et al., 2020; Sravana et al., 2022) process to meet diverse user experience (UX) demands.

*Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, volume: 15, number: 2 (June), pp. 183-201. DOI: *10.58346/JOWUA.2024.I2.013*

*Corresponding authors: [1,3]Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia.

Different applications warrant varying levels of responsiveness and precision, making a one-size-fits-all approach to task scheduling inadequate. For instance, real-time analysis in financial trading systems demands rapid processing with minimal latency, whereas batch processing in data analytics can afford more flexibility in scheduling (Asl & Asl, 2022).

Some deep learning models (Ay et al., 2019; Sihag et al., 2021) need a large amount of VRAM (Choi & Lee, 2021), for example Super Resolution General Adversarial Network (SRGAN) (Ledig et al., 2017; Iman et al., 2023). SRGAN is a deep learning model utilized to enhance the resolution of an image by a factor of four compared to its original size. In the process of generating high-resolution images, the SRGAN model requires substantial VRAM (Video Random Access Memory) (Silvano et al., 2023; Srinivasareddy et al., 2021; Bobir et al., 2024).

The Differentiate Quality of Service Scheduler (DQoES) is a task scheduler method capable of meeting different quality of service requirements across various types of tasks (Mao et al., 2022; Rathi et al., 2024). In the DQoES scheduler method, container placement on worker nodes utilizes the container placement method present in Docker Swarm, where a new container is placed on the node with the fewest containers. This container placement method does not consider the load of each existing container nor the available capacity of each node's resources (Kutlu et al., 2021; Johnson et al., 2021; Shadadi et al., 2022)

In this paper, we will compare the performance of several container placement methods in the context of load distribution across each node and running time. Experiments for each container placement method will be conducted on the DQoES scheduler architecture. The container placement methods to be compared include:

- Default Docker Swarm, which is the method used in the DQoES method (Mao et al., 2022).
- Discrete Whale Optimization Container Placement (DWOCP), a container placement method using whale optimization for energy-efficient and efficient container placement (Al-Moalmi et al., 2021).
- Proposed Whale Optimization, a modification of the DWOCP method.
- Hybrid Whale Optimization, a combined method of whale optimization with a procedure conducted before implementing whale optimization. This hybrid whale optimization method will be tested on both the DWOCP and Proposed Whale Optimization methods.

The remainder of this paper is as follows: Section 2 provides a comprehensive review of the literature, highlighting previous studies that have explored container placement methods. In Section 3, we describe the methodology employed in our research. This includes a detailed explanation of the study design, experiment design, and proposed method for container placement. Section 4 presents the results of our experiment. It systematically reports the findings from the data analysis, using figures. This section evaluates the outcomes in the context of our research questions and hypotheses, offering insights into container placement for deep learning applications. Section 5 discusses the implications of our findings. It interprets the results in light of the existing literature, considering both the theoretical and practical implications. This section also addresses the limitations of the study and suggests directions for future research.

## 2 Related Work

In the realm of cloud computing and data center management, the optimization of container deployment within Container-as-a-Service (CaaS) environments emerges as a critical challenge, particularly in balancing power consumption and resource utilization (Al-Moalmi et al., 2021; Saadawi et al., 2024;

Babenko et al., 2021). Al-Moalmi et al paper contributes to the growing body of literature by addressing the complexity of initial container and Virtual Machine (VM) placement on physical machines (PMs) within CaaS contexts. Unlike previous works, which predominantly employ simple heuristics for container placement and more sophisticated strategies for VM allocation separately, Al-Moalmi et al study introduces an innovative approach that integrates these processes. Leveraging the Whale Optimization Algorithm (WOA), the proposed methodology innovatively tackles the placement of containers and VMs as a singular optimization challenge, aiming for an optimal allocation of resources that simultaneously minimizes power usage and maximizes resource efficiency. The uniqueness of this approach lies in its consideration of the heterogeneity among containers, VMs, and PMs, a factor often overlooked in existing methodologies. Comparative analysis with recent methods across various heterogeneous environments showcases the proposed algorithm's effectiveness, marking a significant advancement in the optimization of resource allocation and energy efficiency in cloud data centers.

Container placement (CP) within Container as a Service (CaaS) frameworks presents a sophisticated challenge that significantly impacts energy efficiency in cloud computing environments (Zhang et al., 2019). Traditional approaches to this problem often rely on linear models for server energy consumption, which fail to distinguish between different CP strategies in homogeneous hosting setups, leading to energy inefficiencies. Zhang et al research advances the discourse by illustrating the potential for energy conservation through the optimization of CP under a nonlinear energy consumption model. Employing a genetic algorithm (GA)-based strategy, Zhang et al study seeks to navigate the complexities of optimizing CP for enhanced energy efficiency. However, it acknowledges the limitations of conventional GAs, particularly under conditions of high virtual machine (VM) resource utilization, which tend to degrade performance.

To address these challenges, the paper introduces an innovative improved genetic algorithm (IGA), which incorporates dual exchange mutation operations and a novel control function for selectively applying these operations, thus optimizing the search for energy-efficient CP solutions. Through rigorous experimental comparison with standard CP strategies like spread and binpack, as well as other optimization algorithms including First Fit, Particle Swarm Optimization (PSO), and conventional GAs, the proposed IGA demonstrates superior performance in achieving energy savings. Furthermore, the results affirm the IGA's ability to mitigate performance degradation associated with high VM resource utilization, offering new, more efficient CP solutions. Zhang et al work not only contributes to the optimization of CP for energy efficiency but also broadens the understanding of applying advanced genetic algorithms in the context of CaaS, marking a significant step forward in the quest for more sustainable cloud computing practices.

The burgeoning field of containerization in cloud computing, particularly with Docker's prominence for operating system level virtualization, calls for innovative solutions to optimize resource utilization within data centers (Zhang et al., 2018). Zhang et al research paper expands on existing literature by addressing a critical gap in the systematic collaboration between container placement (CP) and Virtual Machine (VM) placement towards Physical Machines (PMs). Previous studies have largely isolated the consideration of CP and VM placement, leading to inefficient physical resource utilization characterized by a scattered distribution of containers. In response, Zhang et al study introduces a novel "Container-VM-PM" architecture, pioneering a strategy that concurrently considers the placement of containers, VMs, and PMs. By developing and implementing a fitness function for the selection process of VMs and PMs, the proposed strategy showcases a marked improvement in optimizing physical resource utilization over existing methods. Simulation experiments reinforce the effectiveness of this approach,

positioning it as a superior solution for enhancing efficiency in data center operations and setting a new precedent for research in the domain of cloud computing resource management.

The emerging trend of containerization, specifically through the Container as a Service (CaaS) model, underscores the need for efficient container placement strategies in cloud computing environments (Hussein et al., 2019). Hussein et al study acknowledges the limitations of traditional approaches that focus separately on virtual machine (VM) placement on physical machines (PMs) and container or task placement on VMs, often resulting in suboptimal resource utilization. Addressing this gap, the research introduces a novel placement algorithm that aims to optimize resource utilization across both VMs and PMs, with a specific focus on improving the efficiency in terms of CPU cores and memory size. By incorporating scheduling heuristics such as Best Fit (BF) and Max Fit (MF), alongside a meta-heuristic approach using Ant Colony Optimization based on Best Fit (ACO-BF), the study presents a comprehensive solution to the container placement challenge. The experimental findings highlight the effectiveness of the proposed ACO-BF algorithm over traditional BF and MF heuristics, showcasing a significant enhancement in the resource utilization of both VMs and PMs within cloud environments. Hussein et al work not only advances the field of cloud computing resource management but also contributes to the broader discourse on optimizing cloud service models for enhanced operational efficiency.

The advent of Linux Containers (LXC) has introduced a paradigm shift in computing resource allocation and isolation, emphasizing high performance and lightweight virtualization alternatives (U-Chupala et al., 2017). U-Chupala et al study explores the inherent advantages of LXC, particularly its lower resource overhead and reduced container migration time compared to traditional virtual machines (VMs). These attributes make LXC an ideal candidate for frequent container placement modifications, a concept yet to be fully leveraged by conventional container scheduling mechanisms. Current strategies primarily focus on identifying the optimal placement for new containers, which then remain static throughout the container's lifecycle. However, this approach becomes less effective for long-lived containers, where initial placement may not remain optimal due to dynamic changes in the cluster (Chatterjee et al., 2024).

Addressing this challenge, the research introduces a novel scheduling mechanism termed container rebalancing, designed to enhance LXC cluster utilization through continuous, minimal-interference adjustments in container placement. By integrating a rebalancing process with the scheduling mechanism, the proposed method aims to achieve a balanced utilization across all hosts in the LXC cluster. The feasibility and effectiveness of container rebalancing are demonstrated through simulations using Google's cluster data, which highlight a notable improvement in both container scheduling rates and overall cluster utilization. The study presents container rebalancing as a promising approach for optimizing container placement in LXC environments, offering significant implications for the efficiency and flexibility of cloud computing resource management.

In recent advancements within cloud-native applications, the security threats posed by the weak isolation of lightweight containers have become a significant concern. These vulnerabilities have led to the rise of co-resident threats, which can easily propagate among applications through microservice calls, thereby jeopardizing the integrity and security of numerous cloud-native systems. Addressing this pressing issue, the research detailed by Zhou et al provides a comprehensive analysis of the mechanisms through which container co-resident threats spread within cloud-native applications (Zhou et al., 2023). It introduces an innovative approach to mitigate these threats while simultaneously enhancing load balancing. The study pioneers a Deep Q-Network (DQN)-based Microservice Container Placement Algorithm (MsCPA) designed to optimize both the containment of co-resident threats and load

distribution among cloud-native applications. Empirical simulation results underscore the efficacy of the proposed algorithms, demonstrating an average reduction in the threat propagation range by 18.06% and an improvement in load balancing performance by 13.97%. This work not only sheds light on the critical issue of container security within cloud-native ecosystems but also offers a viable solution that marries enhanced security with efficient resource utilization.

The research conducted by Lv et al tackles the complex issue of container distribution in large-scale data centers, a problem that is exacerbated by the dual challenges of container placement and reassignment (Lv et al., 2019). Their study delves into these issues within a real industrial context, uncovering a fundamental conflict between reducing communication costs and achieving balanced resource utilization. To address these challenges, we introduce two novel algorithms. The first, an Efficient Communication Aware Worst Fit Decreasing algorithm, is designed for the container placement phase. It strategically places a set of new containers into data centers, optimizing for communication efficiency. The second, a two-stage algorithm named Sweep & Search, aims at the container reassignment phase, refining the initial distribution of containers by facilitating their migration among servers to optimize resource utilization.

Implemented and rigorously tested within Baidu's data centers, our proposed solutions have demonstrated superior performance compared to existing state-of-the-art strategies. Evaluation results are impressive, showing up to 70% better performance in specific metrics and an overall service throughput increase of up to 90%. This research not only provides actionable insights into optimizing container distribution in data centers but also significantly contributes to enhancing the efficiency and effectiveness of service deployment in large-scale industrial environments.

# 3 Research Methodology

In our research methodology for this paper, we incorporate five key components: the scheduler module, manager worker system, whale optimization module, AI module, and container placement methods. Within the container placement methods, two approaches are proposed: the proposed whale optimization and the proposed hybrid method.

### Scheduler Module

The Differentiated Quality of Experience Scheduler (DQoES) is a scheduling method for Docker containers for deep learning inference (Mao et al., 2022).

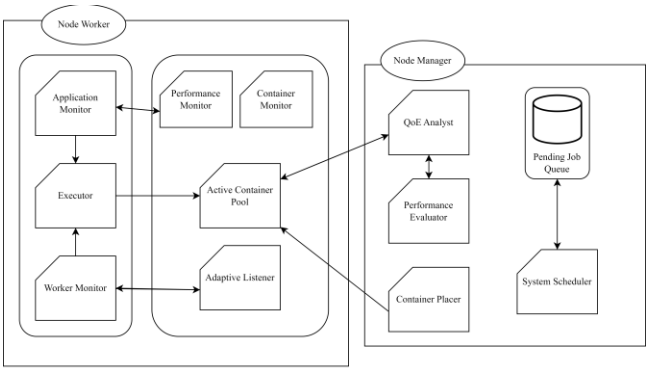


Figure 1: Scheduler Module

DQoES features a manager-worker architecture, a distributed model architecture where there are nodes acting as managers and others serving as workers. For optimal implementation, the DQoES scheduler architecture includes pending job queues to prevent processing from exceeding the capacity of the available resource nodes. The DQoES scheduler architecture, after the addition of the pending job queue, can be seen in the Figure 1.

**Manager-worker System**

In the DQoES scheduler framework, the Node Manager is assigned the roles of interacting with users, analyzing requests, and managing worker nodes in accordance with their respective clusters. Furthermore, the Node Manager is responsible for collecting Quality of Experience (QoE) metrics from targeted clients, subsequently forwarding these QoE data to the worker nodes, and overseeing the conditions of active workloads throughout the system in its entirety. The primary responsibility of worker nodes includes provisioning main computing resources such as CPUs, GPUs, and memory to execute tasks, store data, and relay their status back to the Node Manager. Deep learning applications are deployed on worker nodes with real-time monitoring of resource usage for each specific application. Utilizing the acquired QoE data, worker nodes adjust resource limits within each container to optimize outcomes and performance capabilities. The configuration of the manager-worker system utilized is depicted in Figure 2.

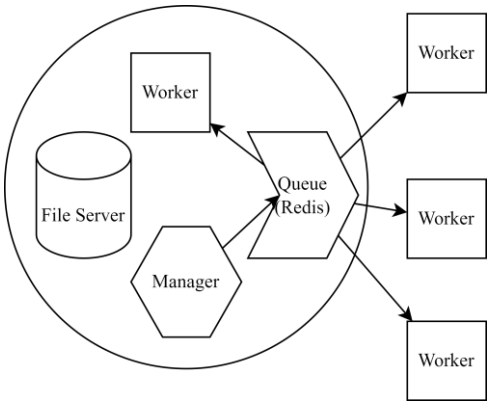


Figure 2: Manager-worker System

**AI Module**

Table 1: DIV2K Dataset used in this Paper for Training the SRGAN Models

| Filename Dataset | Usage |
|---|---|
| DIV2K_train_HR | Training |
| DIV2K_train_LR_bicubic_X4 | Training |
| DIV2K_valid_LR_bicubic_X4 | Testing |

In this paper, the SRGAN model is used as the framework processed by the worker nodes (Ledig et al., 2017). Several variations of the SRGAN model are utilized, including no prune, random unstructured, L1 norm, and L2 norm (Kim et al., 2022). These SRGAN models represent different types based on the method of layer-wise pruning aimed at reducing the complexity of the SRGAN model. The SRGAN models were trained using DIV2K dataset (Agustsson & Timofte, 2017), where the composition of training dataset and testing dataset can be seen on Table 1.

**Whale Optimization Module**

The whale optimization module was created from previous work conducted by Al-Moalmi et al (Al-Moalmi et al., 2021). A group of whale agents is utilized to search for the optimal solution within a search space. Each whale agent is represented as a matrix solution named S. S is a 2*L matrix, where L represents the number of containers. The first row in the matrix S represents the VM index, and the second row represents the PM index. If there are a total of m VMs, then the VM index values are ∈[1,m]. If there are a total of n PMs, then the PM index values are ∈[1,n]. Each column in the matrix S represents the solution for container placement, indicating on which VM and PM indexes the container is placed. A container will be placed on one and only one VM, while a VM will be placed on one and only one PM, thus allowing the VM and PM index values for placing containers to be the same or repeated. However, it is noted that the VM and PM index values must remain consistent if the same values are present.

The S matrix formed can be seen in equation 1, where i is the VM index, j is the PM index, and l represents the l-th container. To ensure that one VM is placed on one and only one PM, equation 2 is used to determine the PM index, which is utilized to place a VM. In the first iteration, to determine the location of each container, the VM and PM indexes are randomly assigned. The placement of containers is carried out as many times as there are whale agents, so that each whale agent has its own container layout solution in the form of the S matrix. Once all whale agents have their respective S matrices, the most optimal solution will be determined. The criteria for the optimal solution are the solution that requires the least number of VMs and PMs. If the total number of VMs and PMs across solutions is the same, the solution with the fewer number of PMs is considered more optimal.

$$S = \begin{bmatrix} x_{1,1}^{i}, & x_{1,2}^{i}, & x_{1,1}^{i}, & \dots & x_{1,L}^{i} \\ x_{2,1}^{ij}, & x_{2,2}^{ij}, & x_{2,3}^{ij}, & \dots & x_{2,L}^{ij} \end{bmatrix} \quad (1)$$

$$x_{2,l}^{ij} = \begin{cases} x_{2,l-n}^{ij}, & x_{1,l-n}^{i} = x_{1,l}^{i} \\ x_{2,l}^{ij}, & if\ not \end{cases} \quad (2)$$

If all whale agents already possess a solution and the optimal solution has been determined in the first iteration, for subsequent iterations, the value of each solution will be updated, where all solutions that are not optimal will be updated so their values will approach the optimal solution. Solution values are also updated based on a solution value selected at random. To update solution values, equation 3 is used, where S(t+1) represents the new solution value, $S_\eta$ is the optimal solution, S(t) represents the current solution value, $S_{rand}$ is a randomly chosen solution, prob is a random value determined from ∈ [0,1], b is a constant value in the DWO-CP algorithm search process, z is a random value from ∈ [-1,1]. The value of $Cf_1$ is obtained from equation 6, where a is an integer value that decreases linearly from 2 to 0, and rand is a decimal number obtained randomly within the range [0,1]. The value of $Cf_2$ is obtained from equation 7, with the same notation as in equation 6. The value of Di ⃗r is obtained from equation 4, and the value of Di ⃗r is obtained from equation 5.

$$S(t+1) = \begin{cases} S_\eta(\vec{t}) - C\overrightarrow{f_1} \times D\vec{\imath}r, & Prob < 0.5 \ \&\ |Cf_1| < 1 \\ S_{rand}(\vec{t}) - C\overrightarrow{f_1} \times D\vec{\imath}r, & Prob < 0.5 \ \&\ |Cf_1| \geq 1 \\ D\vec{\imath}r' \times e^{bz} \times \cos(2\Pi z) + S_\eta(\vec{t}), & if\ not \end{cases} \quad (3)$$

$$D\vec{\imath}r = \left|C\overrightarrow{f_2} \times S_{rand}(\vec{t}) - S(\vec{t})\right| \quad (4)$$

$$D\vec{\imath}r' = \left|S_\eta(\vec{t}) - S(\vec{t})\right| \quad (5)$$

$$C\overrightarrow{f_1} = 2\vec{a}.\overrightarrow{rand} - \vec{a} \quad (6)$$

$$C\overrightarrow{f_2} = 2.\overrightarrow{rand}, \quad (7)$$

The updated solution values must ensure that the VM index falls within the range [1, m], thus the new value $x_{1,l}^{i}$ is inserted into equation 8, where $x_{1,l}^{i}(t+1)$ represents the VM index value for the updated solution. The same applies to the PM index, which must be ensured to fall within the range [1, n], hence the value $x_{2,l}^{ij}$ is included in equation 9, where $x_{2,l}^{ij}(t+1)$ represents the PM index value for the updated solution.

$$x_{1,l}^{i}(t+1) = \begin{cases} x_{1,l}^{i}(t+1)\ mod\ m, & if\ x_{1,l}^{i}(t+1) \notin [1,m] \\ x_{1,l}^{i}(t+1), & if\ not \end{cases} \quad (8)$$

$$x_{2,l}^{ij}(t+1) = \begin{cases} x_{2,l}^{ij}(t+1)\ mod\ n, & if\ x_{2,l}^{ij}(t+1) \notin [1,n] \\ x_{2,l}^{ij}(t+1), & if\ not \end{cases} \quad (9)$$

**Container Placement Methods**

There are two methods that will be proposed: proposed whale optimization, and proposed hybrid. Proposed whale optimization is a modification of the Discrete Whale Optimization Container Placement (DWOCP) developed in previous research (Al-Moalmi et al., 2021). Proposed hybrid is a method that combines whale optimization with a procedure conducted before running whale optimization. The whale optimization used in this hybrid method includes both proposed whale optimization and DWOCP.

**Proposed Whale Optimization**

The proposed Whale optimization method is a modification of the Discrete Whale Optimization Container Placement method. The difference lies in the process of changing the solution matrix for a specified number of whale agents, utilizing matrix with the highest number of PMs and VMs, see equation 10. However, the selection of the updated optimal solution matrix used for container placement still uses matrix with the smallest number of PM and VM usage, according to equation 11.

$$S' = f(S, Pmax, Vmax) \quad (10)$$

$$S\eta = minS'g(S', Pmin, Vmin) \quad (11)$$

**Proposed Hybrid**

This hybrid method represents the integration of the whale optimization method with a preliminary procedure conducted prior to the implementation of whale optimization for container placement. This procedure ensures that all available nodes are allocated containers to process existing requests. If all nodes have been allocated containers and there remain unprocessed requests, then the remaining unprocessed requests will be assigned containers based on the whale optimization method. This paper will utilize both the Discrete Whale Optimization Container Placement (DWOCP) and the proposed whale optimization in implementing this proposed hybrid method.

# 4 Experiments and Results

In this research, we analyze the performance of container placement methods based on running time and task distribution. The container placement methods tested include the default docker swarm container placement method, Discrete Whale Optimization Container Placement (DWOCP) (Al-Moalmi et al., 2021), proposed whale optimization, proposed hybrid with DWOCP, and proposed hybrid with proposed whale optimization. The experiment utilizes 3 computer nodes where each computer is equipped with a GPU. Node A acts as both a manager and a worker, while Node B and Node C serve as workers, see

Figure 3. All three nodes communicate through Docker Swarm (Gao et al., 2016) and the deep learning application takes the form of an Application Programming Interface (API).

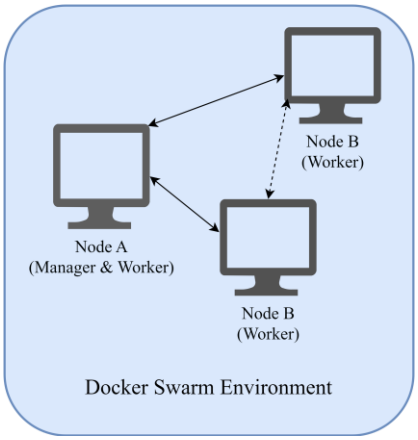


Figure 3: Experiment Architecture

**Running Time**

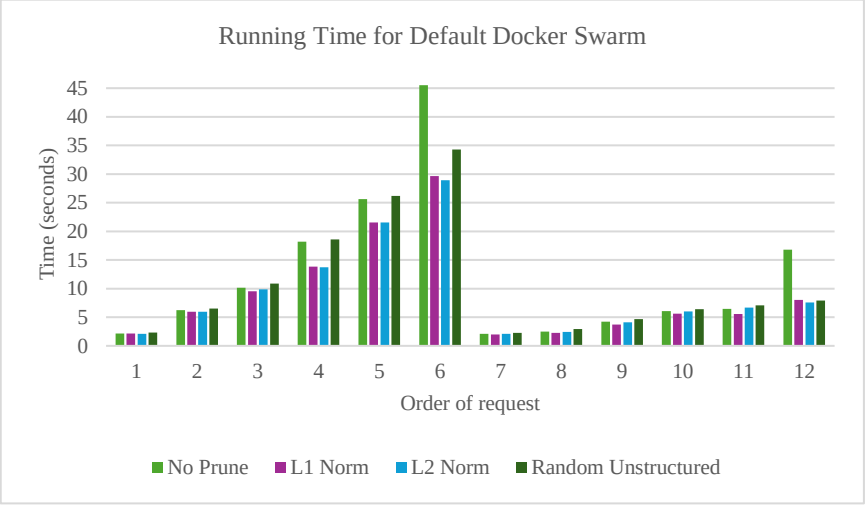


Figure 4: Running Time for Default Docker Swarm

Based on Figure 4, The "No Prune" condition consistently has the lowest running times across all orders of requests, indicating that this configuration or condition is the most efficient in terms of speed. The "L1 Norm" and "L2 Norm" conditions show similar patterns and are generally close in value, with "L2 Norm" occasionally having longer running times. This suggests that while both configurations have an impact on running time compared to "No Prune", their effects are somewhat similar to each other. The "Random Unstructured" condition exhibits the most variability and generally the highest running times, particularly spiking at the 7th order of request. This indicate a less predictable or less efficient process compared to the more structured "No Prune", "L1 Norm", and "L2 Norm" conditions. The spike for "Random Unstructured" at the 7th request is notably pronounced and could suggest an outlier or a specific instance where the process was significantly slower. Similarly, there's a smaller but still notable increase in running time for the "L2 Norm" at the 7th request, while the "L1 Norm" appears to be less affected.

Figure 5 shows there is a general increase in variability and range of the running times across all configurations compared to the previous graph in Figure 4 for default docker swarm. This may suggest that the container placement task is more complex or that the DWOCP process introduces more variability in computation time. "No Prune" continues to show the lowest running times on average, but the difference between "No Prune" and other SRGAN type seems less pronounced than in the DWOCP scenario, especially at order 4, where "No Prune" has a peak that surpasses the "L1 Norm". "Random Unstructured" displays the highest level of inconsistency, with running times ranging from moderately low to the highest recorded on the graph, peaking at order 11. "L1 Norm" and "L2 Norm" continue to exhibit similar patterns with peaks and valleys, though "L2 Norm" has the highest single value at order 4. Both “L1 Norm” and “L2 Norm” have a significant peak at this point, which might be an area of interest for further investigation. The graph in Figure 5 shows that at certain points, like the 4th and 11th orders of requests, all SRGAN types experience an increase in running time, which could suggest that those particular requests are inherently more demanding or that the DWOCP process encounters specific challenges at these points. The graph in Figure 5 could be instrumental for those looking to optimize container placement in terms of speed, indicating areas where certain algorithms may falter or succeed under the DWOCP strategy. It also emphasizes the importance of considering variability and maximum potential delay when choosing an algorithm for real-world applications.

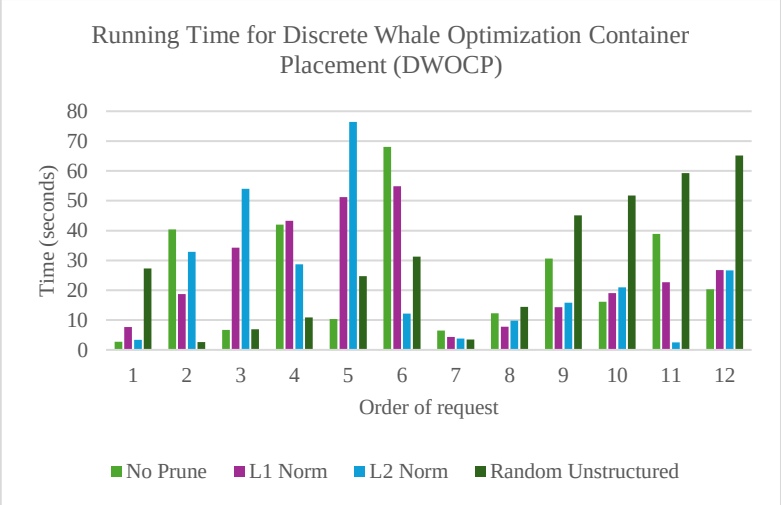


Figure 5: Running Time for Discrete Whale Optimization Container Placement (DWOCP)

Running time for proposed whale optimization can be seen on Figure 6. Similar to the previous charts in Figure 5, the "No Prune" condition tends to have lower running times compared to other configurations, but unlike using default docker swarm container placement method, it does not consistently have the lowest times. This suggests that while pruning may not always be the fastest, its performance is relatively stable. The "L1 Norm" and "L2 Norm" conditions show variability across different orders of request, with "L2 Norm" peaking significantly at the 5th and 11th requests. It is important to note the relative decrease in running time for "L2 Norm" at the 7th request compared to the graphs in Figure 4 and Figure 5, suggesting that the proposed optimization might be having an effect here. The "Random Unstructured" category, while still variable, does not reach the same extremes as seen in the DWOCP graph in Figure 5, indicating possibly more controlled performance under this proposed whale optimization method. Across all configurations, the graph in Figure 6 shows a trend where running times do not consistently increase or decrease with the order of request. This non-linear behavior indicates that the running time may be influenced by factors other than just the order of the

request, such as the complexity of the request itself or the specific optimization being applied, which means when using proposed whale optimization, the load request distributed more evenly across available resource node. The peaks at the 5th and 11th orders for "L2 Norm" suggest that there may be particular types of requests that are more challenging for this configuration under the proposed whale optimization.

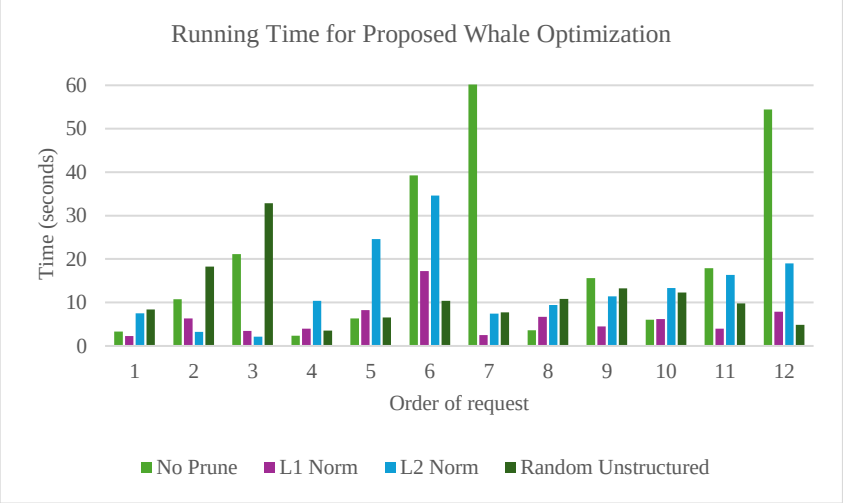


Figure 6: Running Time for Proposed Whale Optimization

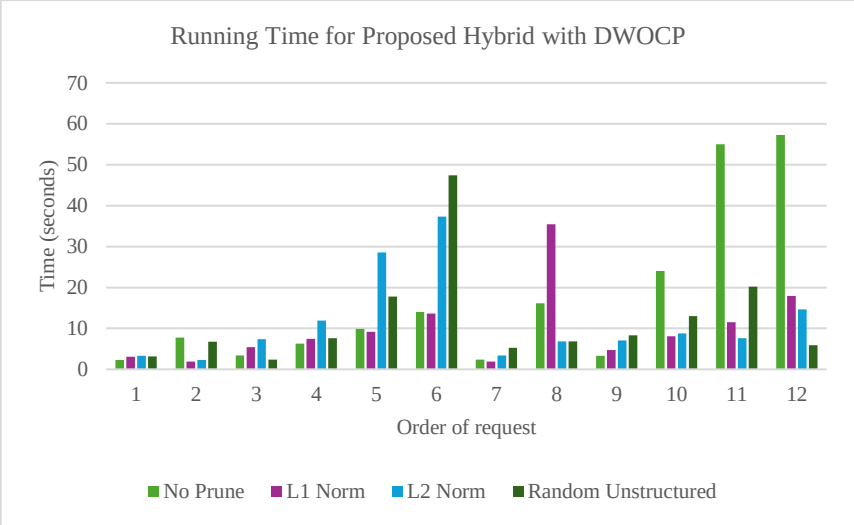


Figure 7: Running Time for Proposed Hybrid with DWOCP

Across all SRGAN types, the running times show a considerable range of variation for each order of request, see Figure 7. However, unlike the graph for DWOCP in Figure 5, the overall running times seem to be lower, suggesting that the hybrid approach may have improved efficiency. "No Prune" still generally has the lowest running times, but when using the proposed hybrid with DWOCP, it is advantage comparing when using DWOCP is the running time was reduced, especially in orders 5 and 9, where "L1 Norm" and "L2 Norm" have comparable or even lower running times. The "Random Unstructured" SRGAN type continues to show high variability and peaks, particularly at orders 6 and 12. This suggests that the unpredictability of “Random Unstructured” SRGAN type persists even in the proposed hybrid with DWOCP. The data in Figure 7 suggests that the hybrid approach combines

elements of both DWOC and proposed hybrid strategy, potentially balancing the strengths and weaknesses of each. It also implies that while "No Prune" is often the fastest, there are specific instances where "L1 Norm" and "L2 Norm" are competitive or even superior.

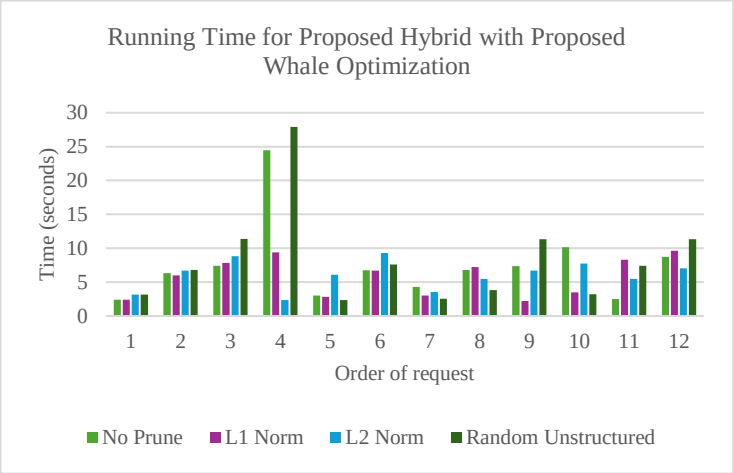


Figure 8: Running Time for Proposed Hybrid with Proposed Whale Optimization

Figure 8 shows the scale of running times is significantly lower than the graph in Figure 7, Figure 6, Figure 5, Figure 4, with the highest recorded time being just above 25 seconds, and most running times being under 15 seconds. This indicates a substantial improvement in running time compared to the previously analyzed strategies which are, DWOCP, proposed whale optimization, and proposed hybrid with DWOCP. The "No Prune" SRGAN type generally shows low running times, but unlike the graph in Figure 7, Figure 6, Figure 5, Figure 4, it does not always have the lowest time. This suggests that the improvements in the proposed hybrid method with proposed whale optimization may have reduced the relative benefit of not pruning. All SRGAN types exhibit peaks and valleys, but the "L2 Norm" has a notable peak at the 5th order of request, which is significantly higher than the other SRGAN types for that order. This could point to a specific scenario where "L2 Norm" is less efficient compared to other SRGAN types. The "Random Unstructured" SRGAN type does not show as dramatic peaks as in the other graphs in Figure 8, suggesting that the proposed hybrid with proposed whale optimization might help to mitigate the variability this SRGAN type showed previously. "L1 Norm" presents as a consistent alternative across most orders of request, without any extreme peaks, which may indicate reliability in the context of the proposed hybrid with proposed whale optimization. This iteration of the running time analysis suggests that the proposed hybrid with the proposed whale optimization may lead to a more consistent and generally improved running time across different methods, which could be very beneficial in practical applications. The reduction in running time and the relative performance of the methods is a result of an effective combination of the strengths of the proposed whale optimization and proposed hybrid.

**Task Distribution**

Figure 9 shows that Node C has the highest number of tasks allocated in all conditions except for the L1 Norm, where it shares the same number of tasks as Node B. Node A consistently has the fewest tasks across all conditions. The distribution of tasks is most uneven in the No Prune and Random Unstructured SRGAN types, where Node C has significantly more tasks compared to Nodes A and B. For the L1 Norm and L2 Norm SRGAN types, the tasks are more evenly distributed between Nodes B and C, with Node A still having the least tasks. This distribution imply that the task allocation behavior varies with

different types of SRGAN model, affecting how workload is distributed when using default docker swarm container placement. It's important to note that Node C seems to be the most utilized node in this setup.

Based on Figure 10, under the "No Prune" SRGAN type, Node A is assigned the most tasks, slightly more than Nodes B and C. The "L1 Norm" SRGAN type shows a more balanced distribution of tasks among the three nodes compared to "No Prune," with Node C receiving the highest number and Node A receiving the lowest. In the "L2 Norm" SRGAN type, Node C is allocated the most tasks, which is a significant increase compared to Node A and B. The "Random Unstructured" SRGAN type has a less uniform distribution, with Node A handling the most tasks, followed by Node C, and Node B managing the least. Comparing graph in Figure 10 to the graph in Figure 9, it is apparent that the allocation strategy impacts the distribution of tasks across nodes, which might reflect different optimization techniques used in the DWOCP approach. The allocation appears to vary with different optimization conditions, suggesting that each condition influences task assignment differently.

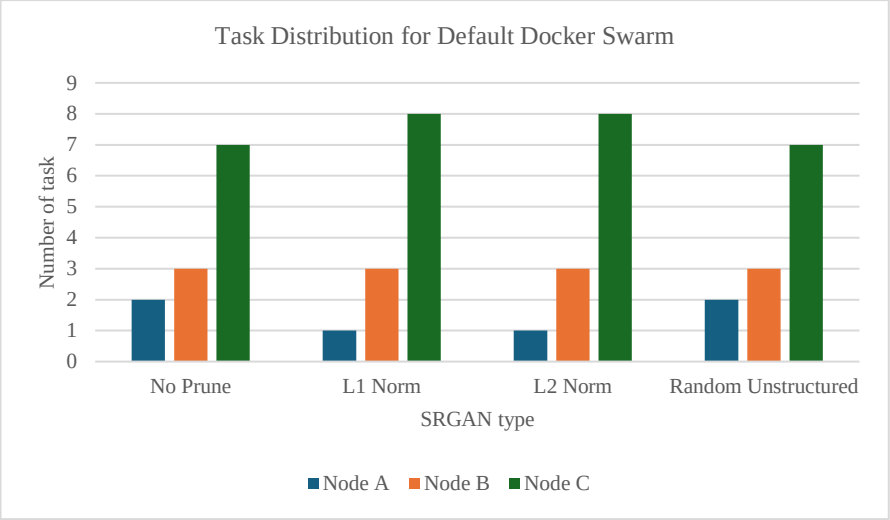


Figure 9: Task Distribution for Default Docker Swarm

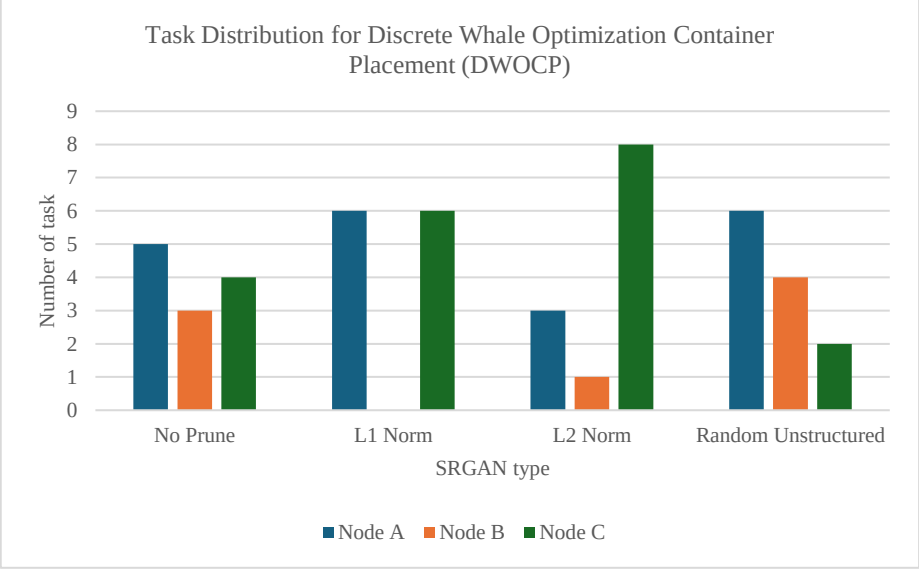


Figure 10: Task Distribution for Discrete Whale Optimization Container Placement (DWOCP)

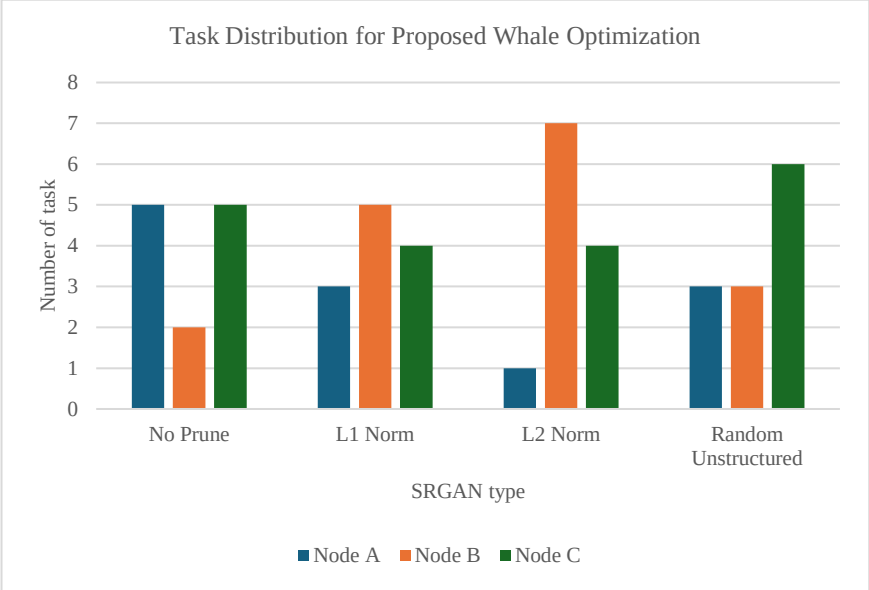


Figure 11: Task Distribution for Proposed Whale Optimization

Under the "No Prune" SRGAN type, the tasks are distributed quite evenly among the three nodes, with Node A having a slightly higher number of tasks, see Figure 11. For the "L1 Norm" SRGAN type, Node B is assigned the most tasks, while Node C has the fewest. In the "L2 Norm" SRGAN type, Node B has the least number of tasks, with Node A having the most. The "Random Unstructured" SRGAN type shows a significant increase in tasks for Node C, while Node A and Node B have a comparatively lower and almost equal distribution of tasks. From these observations, we can infer that the Proposed Whale Optimization strategy results in a variable distribution of tasks depending on the specific SRGAN type, suggesting that the optimization algorithm influences task allocation differently in each scenario. The distribution under the "No Prune" SRGAN type appears to be the most balanced among the three nodes, whereas the other SRGAN type show more variability in the number of tasks assigned to each node.

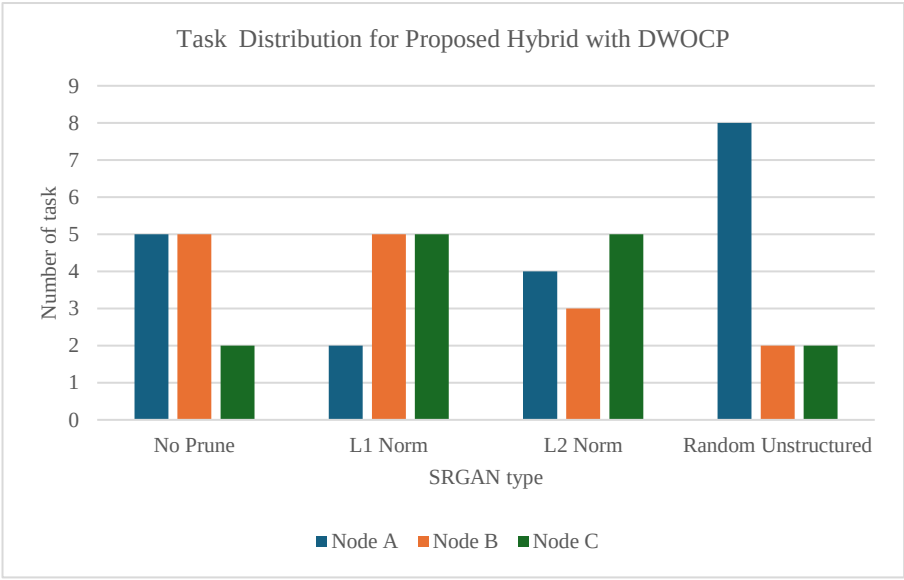


Figure 12: Task Distribution for Proposed Hybrid with DWOCP

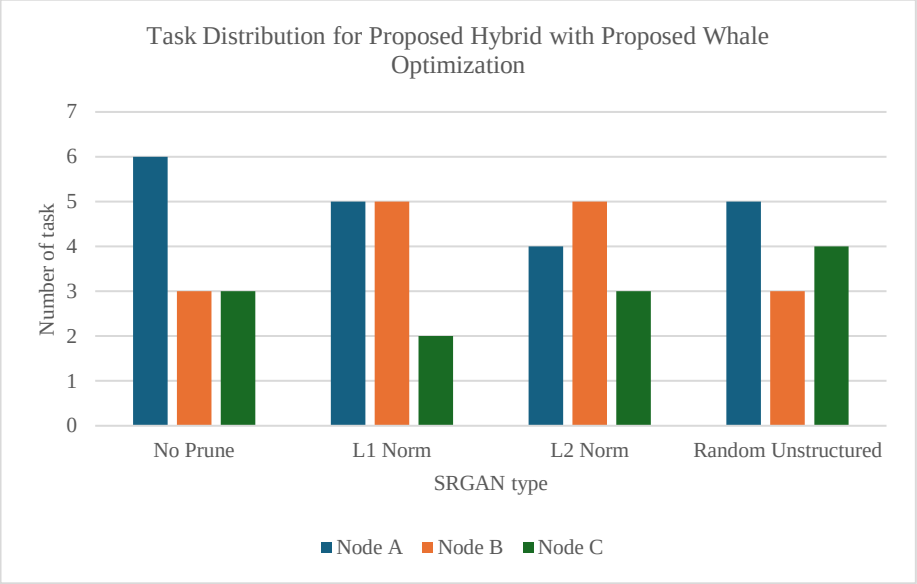


Figure 13: Task Distribution for Proposed Hybrid with Proposed Whale Optimization

Figure 12 shows that in "No Prune" SRGAN type the task distribution is even across all nodes, each having 4 tasks. For "L1 Norm" SRGAN type Node A has the fewest tasks it is 2, while Node B and Node C both have 3 tasks each. When using "L2 Norm" SRGAN type Node C has the fewest tasks it is 2, Node A has 3 tasks, and Node B has the most with 4 tasks. For "Random Unstructured" SRGAN type Node A has a significant majority of tasks which is 8, while Node B and Node C have only 1 task each. The Proposed Hybrid with DWOCP method indicate a hybrid approach to task distribution, combining features of the proposed hybrid and DWOCP methods visualized in the graphs Figure 12. The even distribution in the No Prune SRGAN type and the high task count for Node A in the Random Unstructured SRGAN type are particularly noteworthy. In a broader context, comparing graph in Figure 12 with the graph in Figure 9, Figure 10, and Figure 11 may suggest how different algorithms or heuristics for task distribution in a difference container placement methods can result in varying efficiencies and workload distributions.

Based on Figure 13, for "No Prune" SRGAN type Node A is allocated the most tasks, with the count being 6. Nodes B and C have a fewer number of tasks, with 3 and 2 tasks respectively. When using "L1 Norm" SRGAN type the tasks are evenly distributed among the three nodes, with each node handling 4 tasks. For "L2 Norm" Node B is allocated the most tasks at 5, with Node A handling 3 and Node C only 2. In "Random Unstructured" SRGAN type Node C has the highest task count at 5, Node B has 3, and Node A has the least with 1 task. For proposed hybrid with proposed whale optimization container placement method the allocation strategy adapts based on the SRGAN type. No single node is consistently overloaded or underloaded across all conditions. Particularly in the L1 Norm SRGAN type, the workload is perfectly balanced across all nodes. There is a clear variability in task distribution for the L2 Norm and Random Unstructured SRGAN types, indicating a dynamic approach to task allocation. When comparing between graph in Figure 13 with graphs in Figure 9, Figure 10, Figure 11, and Figure 12, The graph in Figure 13 indicates an attempt to balance the workload more effectively across the nodes. The fact that no single node is always handling the most or least tasks suggests an optimization that may prevent bottlenecks and enhance the overall performance of the system. The equal distribution in the L1 Norm SRGAN type may indicate an optimized scenario for task handling.

# 5 Conclusions and Future Work

In this research, we compared five container placement methods in the context of running time and task distribution. The container placement methods compared include the default Docker Swarm container placement method, Discrete Whale Optimization Container Placement (DWOCP), proposed whale optimization, proposed hybrid with DWOCP, and proposed hybrid with proposed whale optimization. Based on the experimental results obtained, the proposed whale optimization method has better performance than the DWOCP method, but it still cannot surpass the container placement method of the default Docker Swarm. The proposed hybrid with DWOCP method is slightly better than the DWOCP method, and the proposed hybrid with proposed whale optimization method has better performance than both the DWOCP method and the container placement method from the default Docker Swarm.

The graph in Figure 6 demonstrates the potential improvements and challenges with the proposed whale optimization. It highlights where further optimization or investigation might be needed, especially for the "L2 Norm" at specific request orders. It also underscores the relative stability of the "No Prune" SRGAN type and shows a more moderate performance range for "Random Unstructured". Figure 6 also shows the presence of peaks at the 5th and 11th orders for "L2 Norm" indicates that certain kinds of requests might pose more difficulties for this setup when applying the suggested whale optimization technique, meriting additional exploration.

# Acknowledgment

This research was supported by Indonesian Endowment Fund for Education/Lembaga Pengelola Dana Pendidikan (LPDP), Ministry of Finance Indonesia for providing the financial support.

# References

[1] Agustsson, E., & Timofte, R. (2017). Ntire 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 126-135. https://doi.org/10.1109/CVPRW.2017.150

[2] Ahmad, I., AlFailakawi, M. G., AlMutawa, A., & Alsalman, L. (2022). Container scheduling techniques: A Survey and assessment. *Journal of King Saud University - Computer and Information Sciences, 34*(7), 3934–3947. https://doi.org/10.1016/j.jksuci.2021.03.002

[3] Al-Moalmi, A., Luo, J., Salah, A., Li, K., & Yin, L. (2021). A whale optimization system for energy-efficient container placement in data centers. *Expert Systems with Applications, 164*, 113719. https://doi.org/10.1016/j.eswa.2020.113719

[4] Asl, T. M., & Asl, T. S. (2022). Strategy Optimization for Responding to Primary, Secondary and Residual Risks Considering Cost and Time Dimensions in Petrochemical Projects. *Archives for Technical Sciences, 2*(27), 33-48.

[5] Ay, B., Aydın, G., Koyun, Z., & Demir, M. (2019). A visual similarity recommendation system using generative adversarial networks. *In IEEE international conference on deep learning and machine learning in emerging applications (Deep-ML)*, 44-48. https://doi.org/10.1109/Deep-ML.2019.00017

[6] Babenko, V., Danilov, A., Vasenin, D., & Krysanov, V. (2021). Parametric Optimization of the Structure of Controlled High-voltage Capacitor Batteries. *Archives for Technical Sciences, 1*(24), 9–16

[7] Bobir, A. O., Askariy, M., Otabek, Y. Y., Nodir, R. K., Rakhima, A., Zukhra, Z. Y., Sherzod, A. A. (2024). Utilizing Deep Learning and the Internet of Things to Monitor the Health of Aquatic Ecosystems to Conserve Biodiversity. *Natural and Engineering Sciences, 9*(1), 72-83.

[8] Chatterjee, P., Siddiqui, S., Granata, G., Dey, P., & Abdul Kareem, R. S. (2024). Performance Analysis of Five U-Nets on Cervical Cancer Datasets. *Indian Journal of Information Sources and Services, 14*(1), 17–28.

[9] Choi, H., & Lee, J. (2021). Efficient use of GPU memory for large-scale deep learning model training. *Applied Sciences*, *11*(21), 10377. https://doi.org/10.3390/app112110377

[10] Gao, Y., Wang, H., & Huang, X. (2016). Applying docker swarm cluster into software defined internet of things. *In IEEE 8th International Conference on Information Technology in Medicine and Education (ITME)*, 445-449. https://doi.org/10.1109/ITME.2016.0106

[11] Hu, Y., Zhou, H., de Laat, C., & Zhao, Z. (2020). Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. *Future Generation Computer Systems*, *102*, 562-573. https://doi.org/10.1016/j.future.2019.08.025

[12] Hussein, M. K., Mousa, M. H., & Alqarni, M. A. (2019). A placement architecture for a container as a service (CaaS) in a cloud environment. *Journal of Cloud Computing*, *8*, 1-15. https://doi.org/10.1186/s13677-019-0131-1

[13] Iman, M. B., Qusay, A. A., Inass, S. H., & Refed, A. J. (2023). Mobile-computer Vision Model with Deep Learning for Testing Classification and Status of Flowers Images by using IoTs Devices. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 14*(1), 82-94.

[14] Johnson, C., Khadka, B., Ruiz, E., Halladay, J., Doleck, T., & Basnet, R. B. (2021). Application of deep learning on the characterization of tor traffic using time-based features. *Journal of Internet Services and Information Security*, *11*(1), 44-63.

[15] Kim, D. H., Lee, J. W., & Park, S. H. (2022). A Study on Model Compression Methods for SRGAN. *In IEEE International Conference on Electronics, Information, and Communication (ICEIC)*, 1-3. https://doi.org/10.1109/ICEIC54506.2022.9748707

[16] Kutlu, Y., & Camgözlü, Y. (2021). Detection of coronavirus disease (COVID-19) from X-ray images using deep convolutional neural networks. *Natural and Engineering Sciences*, *6*(1), 60-74.

[17] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., & Shi, W. (2017). Photo-realistic single image super-resolution using a generative adversarial network. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 4681-4690. https://doi.org/10.1109/CVPR.2017.19

[18] Lv, L., Zhang, Y., Li, Y., Xu, K., Wang, D., Wang, W., Li, M., Cao, X., & Liang, Q. (2019). Communication-Aware Container Placement and Reassignment in Large-Scale Internet Data Centers. *IEEE Journal on Selected Areas in Communications*, *37*(3), 540–555. https://doi.org/10.1109/JSAC.2019.2895473

[19] Mao, Y., Yan, W., Song, Y., Zeng, Y., Chen, M., Cheng, L., & Liu, Q. (2022). Differentiate quality of experience scheduling for deep learning inferences with docker containers in the cloud. *IEEE Transactions on Cloud Computing*. https://doi.org/10.1109/TCC.2022.3154117

[20] Marakala, V., Sriramakrishnan, G. V., Jakka, G., Shingadiya, C. J., Widiastuti, H. P., & Ortiz, G. G. R. (2022). Use of Deep Learning Application in Medical Devices. *In IEEE 4th International Conference on Inventive Research in Computing Applications (ICIRCA)*, 935-939. https://doi.org/10.1109/ICIRCA54612.2022.9985537

[21] Pongsakorn, U., Watashiba, Y., Ichikawa, K., Date, S., & Iida, H. (2017). Container rebalancing: Towards proactive linux containers placement optimization in a data center. *In IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, *1*, 788-795. https://doi.org/10.1109/COMPSAC.2017.94

[22] Rathi, S., Mirajkar, O., Shukla, S., Deshmukh, L., & Dangare, L. (2024). Advancing Crack Detection Using Deep Learning Solutions for Automated Inspection of Metallic Surfaces. *Indian Journal of Information Sources and Services*, *14*(1), 93–100.

[23] Saadawi, E. M., Abohamama, A. S., & Alrahmawy, M. F. (2024). IoT-based Optimal Energy Management in Smart Homes using Harmony Search Optimization Technique. *International Journal of Communication and Computer Technologies (IJCCTS), 12*(1), 1-20.

[24] Sarker, I. H. (2021). Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, *2*(6), 420. https://doi.org/10.1007/s42979-021-00815-1

[25] Shadadi, E., & Alamer, L. (2022). Hierarchical Parallel Processing for Data Clustering in GPU Using Deep Nearest Neighbor Searching. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, *13*(4), 155-168.

[26] Sihag, V., Vardhan, M., Singh, P., Choudhary, G., & Son, S. (2021). De-LADY: Deep learning-based Android malware detection using Dynamic features. *Journal of Internet Services and Information Security*, *11*(2), 34-45.

[27] Silvano, C., Ielmini, D., Ferrandi, F., Fiorin, L., Curzel, S., Benini, L., Conti, F., Garofalo, A., Zambelli, C., Calore, E., Schifano, S. F., Palesi, M., Ascia, G., Patti, D., Perri, S., Petra, N., Caro, D. D., Lavagno, L., Urso, T., Birke, R. (2023). *A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms*. https://doi.org/10.48550/arXiv.2306.15552

[28] Sravana, J., Indrani, K. S., Saranya, M., Kiran, P. S., Reshma, C., & Vijay, V. (2022). Realısatıon of Performance Optımısed 32-Bıt Vedıc Multıplıer. *Journal of VLSI Circuits and Systems*, *4*(2), 14-21.

[29] Srinivasareddy, S., Narayana, Y. V., & Krishna, D. (2021). Sector beam synthesis in linear antenna arrays using social group optimization algorithm. *National Journal of Antennas and Propagation (NJAP)*, *3*(2), 6-9.

[30] Zhang, R., Chen, Y., Dong, B., Tian, F., & Zheng, Q. (2019). A Genetic Algorithm-Based Energy-Efficient Container Placement Strategy in CaaS. *IEEE Access*, *7*, 121360–121373. https://doi.org/10.1109/ACCESS.2019.2937553

[31] Zhang, R., Zhong, A. M., Dong, B., Tian, F., & Li, R. (2018). Container-VM-PM architecture: A novel architecture for docker container placement. *In Cloud Computing–CLOUD 2018: 11th International Conference, Held as Part of the Services Conference Federation, SCF 2018, Seattle, WA, USA, Proceedings 11*, 128-140. Springer International Publishing.

[32] Zhou, D., Chen, H., & Cheng, G. (2023). A Security Containers Placement Algorithm Based on DQN for Microservices to Defend Against Co-Resident Threat. *In IEEE 8th International Conference on Computer and Communication Systems (ICCCS)*, 683-688. https://doi.org/10.1109/ICCCS57501.2023.10150672

## Authors Biography

Taufiq Odhi Dwi Putra is a Master student in Informatics Department, Institut Teknologi Sepuluh Nopember (ITS), Indonesia. He received the bachelors's degree in Informatics Department from Sebelas Maret University, Indonesia. His current research interests include cloud computing, development operation, artificial intelligence operation, and computer infrastructure.

Royyana Muslim Ijtihadie has phd degree from kumamoto university in 2013. He is also a lecturer and a researcher in Department of Informatics, ITS surabaya, Indonesia. He has experience in various case of distributed systems and IT infrastructure.

Tohari Ahmad received the bachelor's degree in computer science from Institut Teknologi Sepuluh Nopember (ITS), Indonesia, the master's degree in information technology from Monash University, Australia, and the Ph.D. degree in computer science from RMIT University, Australia. He was a consultant for some international companies. In 2003, he moved to ITS, where he is currently a professor. His current research interests include network security, information security, data hiding, and computer networks. His awards and honours include the Hitachi Research Fellowship and JICA Research Program to conduct research in Japan. He is a reviewer of several journals.