s

Abstract

Cloud data centers operate under conflicting objectives such as minimizing execution time, operational energy, monetary cost, and migration overhead. Classical single-objective optimizers fail to capture these trade-offs realistically. This project proposes a **Multi-Objective Neuron Synapse Optimization (MO-NSO)** based framework for cloud data center resource management, where VM-host and container-VM placement decisions are optimized simultaneously. Inspired by biological neuron–synapse interactions, NSO models candidate solutions as neurons that influence each other through adaptive synaptic weights. The proposed work integrates NSO with Pareto dominance, crowding distance, and hypervolume analysis to generate a diverse set of non-dominated solutions instead of a single biased optimum. Realistic models for makespan, VM cost, energy consumption, and migration overhead are used. Experimental results demonstrate that MO-NSO is capable of maintaining solution diversity with competitive convergence behavior compared to classical evolutionary approaches, making it suitable for dynamic cloud environments.

Keywords: Cloud Data Center, Multi-Objective Optimization, Neuron Synapse Optimization, Pareto Front, Energy-Aware Scheduling

Contents

Acknowledgement **i**

Abstract **ii**

List of Figures **v**

List of Tables **vi**

List of Abbreviations **vii**

1 Introduction **1**

1.1 Background and Motivation 1

1.2 Problem Statement 1

1.3 Objectives of the Project 1

2 Literature Review **2**

2.1 Introduction 2

2.2 Review of Existing Approaches 2

2.3 Comparative Analysis of Existing Approaches 3

2.4 Summary 3

3 Proposed Methodology **4**

3.1 Overview of Proposed System/Approach 4

3.2 System Architecture / Framework 4

3.3 Mathematical Modeling of Objectives 5

3.3.1 Makespan Model 5

3.3.2 CPU Allocation Model 5

3.3.3 Energy Consumption Model 6

3.3.4 VM Cost Model 6

3.3.5 Migration Time Model 7

3.3.6 Multi-Objective Optimization Formulation 7

3.4 Neuron–Synapse Weight Update 7

3.5 Neuron Movement 8

3.6 Synaptic Pruning 8

3.7 Reinforcement Learning 8

3.8 Algorithm: MO-NSO 9

4 MO-NSO Algorithm: Example **10**

4.1 Problem Configuration 10

4.2 Algorithm Parameters 10

4.3 Iteration 1: Initialization and First Evaluation 11

4.4 Iteration 2: Neuron Update and Re-evaluation 15

4.5 Comparison: Iteration 1 vs Iteration 2 18

4.6 Conclusion 18

5 Experimental Layout / Design 20
5.1 Hardware/Software Requirements . . . 20
5.2 System Design . . . 20
5.3 Experimental Setup . . . 20
5.4 Expected Outcomes . . . 21

6 Results 22
6.1 Pareto Front Visualizations . . . 22
6.2 Makespan vs Cost Tradeoff . . . 22
6.3 Cost vs Migration Tradeoff . . . 22
6.4 Makespan vs Energy Trade-off . . . 23
6.5 Performance vs Stability (Makespan vs Migration) . . . 23
6.6 Three-Dimensional Pareto Analysis . . . 23
6.7 Summary of Results . . . 25
6.8 **Comparative Performance Evaluation** . . . 26
6.8.1 Scenario 1: Varying Workload Intensity (Container Scalability) . . . 27
6.8.2 Scenario 2: Varying Infrastructure Availability (VM Scalability) . . 31

7 Conclusion and Future Work 34
7.1 Summary of the Work . . . 34
7.2 Achievements and Contributions . . . 34
7.3 Limitations of the Current System . . . 35
7.4 Future Scope and Enhancements . . . 36
7.5 Final Remarks . . . 37

References 38

List of Figures

Table 0.1: List of Figures

Figure No.	Description	Page
6.1	Pareto front showing Makespan vs Cost Tradeoff	22
6.2	Pareto front showing Cost vs Migration tradeoff	23
6.3	Pareto front showing Makespan vs Energy Tradeoff	24
6.4	Pareto front showing Makespan vs Migration	24
6.5	3D Pareto front showing Makespan vs Energy vs Cost	25
6.6	Makespan vs Varying Containers	27
6.7	Cost vs Varying Containers	28
6.8	Energy vs Varying Containers	29
6.9	Migration vs Varying Containers	29
6.10	Makespan vs Varying VMs	31
6.11	Cost vs Varying VMs	32
6.12	Energy vs Varying VMs	33
6.13	Migration vs Varying VMs	33

List of Tables

Table 0.2: List of Tables

Table No.	Description	Page
0.1	List of Figures	v
0.2	List of Tables	vi
0.3	List of Abbreviations	vii
2.1	Comparative analysis of migration-based cloud optimization techniques	2
4.1	Host specifications for the example	10
4.2	VM specifications for the example	10
4.3	Container specifications for the example	10
4.4	Initial neuron population	11
4.5	VM-to-host logits for N_0	11
4.6	Final VM assignment for N_0	13
4.7	Container-to-VM logits for N_0	13
4.8	Final container assignment for N_0	13
4.9	Objective values for neuron N_0	15
4.10	Complete population evaluation (Iteration 1)	15
4.11	Pareto front for Iteration 1	15
4.12	Updated neuron positions after Iteration 2	17
4.13	Objective values after Iteration 2	17
4.14	Pareto front for Iteration 2	17
4.15	Best solutions comparison between iterations	18
4.16	Population convergence analysis	18
6.1	Makespan comparison for fixed VMs (40) and varying containers	27
6.2	Cost comparison for fixed VMs (40) and varying containers	28
6.3	Energy comparison for fixed VMs (40) and varying containers	28
6.4	Migration comparison for fixed VMs (40) and varying containers	29
6.5	MakeSpan comparison between different algorithms for fixed Containers and varying VMs	31
6.6	Cost comparison between different algorithms for fixed Containers and varying VMs	32
6.7	Energy comparison between different algorithms for fixed Containers and varying VMs	32
6.8	Energy comparison between different algorithms for fixed Containers and varying VMs	33

List of Abbreviations

Table 0.3: List of Abbreviations

Abbreviation	**Full Form**
NSO	Neural Swarm Optimization
MO-NSO	Multi-Objective Neural Swarm Optimization
ACO	Ant Colony Optimization
NSGA-II	Non-dominated Sorting Genetic Algorithm II
NSGA-III	Non-dominated Sorting Genetic Algorithm III
VM	Virtual Machine
SLA	Service Level Agreement
DC	Data Center
PSO	Particle Swarm Optimization
GA	Genetic Algorithm
SA	Simulated Annealing
MI	Million Instructions
MIPS	Million Instructions Per Second
QoS	Quality of Service

1 Introduction

1.1 Background and Motivation

Modern cloud data centers host large numbers of virtual machines (VMs) and containers on heterogeneous physical hosts. Efficient placement and scheduling of these entities is critical because poor allocation directly leads to higher execution time, increased energy consumption, unnecessary VM migrations, and inflated monetary costs. These objectives are inherently conflicting: reducing makespan often increases energy usage, while minimizing cost may degrade performance.

Most real-world cloud optimization problems are therefore **multi-objective by nature**. However, many existing approaches simplify the problem using weighted-sum formulations, which hide trade-offs and force a single subjective solution. This motivated the exploration of Pareto-based optimization techniques that preserve multiple optimal trade-offs.

1.2 Problem Statement

The core problem addressed in this project is:

How can VM-host and container-VM placement in a cloud data center be optimized simultaneously with respect to makespan, energy consumption, monetary cost, and migration overhead, without collapsing objectives into a single scalar value?

The challenge lies in designing an optimizer that is flexible, scalable, and capable of maintaining solution diversity under realistic system models.

1.3 Objectives of the Project

The main objectives of this project are:

- To model cloud data center scheduling as a **multi-objective optimization problem**.
- To adapt the **Neuron Synapse Optimization (NSO)** algorithm for multi-objective optimization.
- To design realistic mathematical models for makespan, energy, cost, and migration overhead.
- To generate and analyze Pareto-optimal solutions using dominance, crowding distance, and hypervolume.
- To evaluate whether MO-NSO can compete with established evolutionary algorithms in terms of solution quality and runtime behavior.

2 Literature Review

2.1 Introduction

Cloud scheduling and resource optimization have been extensively studied using heuristic, metaheuristic, and evolutionary algorithms. The growing scale and heterogeneity of cloud systems have made classical deterministic approaches inadequate.

2.2 Review of Existing Approaches

- **Physics-Based Algorithms:** Methods like Simulated Annealing (SA) mimic physical cooling processes but may struggle with scalability in high dimensions.
- **Evolutionary algorithms** such as GA and NSGA-II are widely used due to their Pareto-based nature.
- **Swarm Intelligence:** Algorithms like Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) mimic social behavior. PSO simulates particle movement but can suffer from premature stagnation.
- **Neuron Synapse Optimization (NSO):** A recent algorithm inspired by neural plasticity and Hebbian learning (”neurons that fire together, wire together”). It features synaptic weight updates based on fitness differences and spatial proximity.

Table 2.1: Comparative analysis of migration-based cloud optimization techniques

Study	**Cost Efficiency**	**Makespan**	**Migration Time**	**Resource Utilization**	**QoS Impact**
(Sminite & Afdel, 2020)	Reduced power consumption and operational costs via container migration	Migration time reduced, improving QoS	Migration time significantly lowered compared to VM migration	Dynamic resource management improves utilization	SLA violations reduced by migration optimization
(Altahat et al., n.d.)	Power consumption savings between 3.5 to 16.25 MW, cost included in constraints	Efficient container deployment minimizing makespan	Secure migration with encryption, migration overhead considered	Balanced server loads and reduced node imbalance	Security ensured during migration, improving QoS
(Khan et al., 2024)	20–30% reduction in execution and energy costs	Optimized makespan and cost trade-off	Migration time not primary focus	Optimized IT resource usage	Cost–makespan trade-offs improve service quality
(Conob & Eduard, 2024)	24% operational cost reduction with migration algorithms	Not explicitly detailed	Migration reduces SLA violation times by 40%	Prevents server overloading, improves resource use	SLA violations significantly reduced

2.3 Comparative Analysis of Existing Approaches

Traditional NIAs like PSO and GA balance exploration and exploitation but often struggle with local optima entrapment in multimodal landscapes. Hybrid approaches (e.g., PSO-GA) attempt to mitigate this but increase computational complexity. The NSO algorithm distinguishes itself through dynamic adaptability, using pruning to remove weak connections and reinforcement to strengthen promising ones, theoretically offering better convergence in complex spaces.

2.4 Summary

Current literature supports the efficiency of bio-inspired algorithms but highlights a need for robust multi-objective frameworks. This project addresses this by adapting the NSO algorithm for multi-objective cloud placement.

3 Proposed Methodology

3.1 Overview of Proposed System/Approach

The proposed system uses **Multi-Objective Neuron Synapse Optimization (MO-NSO)** as the core search engine. Each neuron represents a complete placement decision. Neurons interact through synaptic weights that adapt based on Pareto rank differences.

3.2 System Architecture / Framework

The proposed MO-NSO framework follows a learning-driven scheduling pipeline rather than a static heuristic mapping. The complete operational flow is summarized as follows:

1. **Infrastructure and Workload Modeling**
   Construct a realistic cloud environment consisting of hosts, VMs, and containers with MIPS, RAM, power, and pricing attributes.

2. **Neuron Encoding (Solution Representation)**
   Each neuron encodes continuous affinity matrices for VM–Host and Container–VM placement.

3. **Capacity-Aware Decoding**
   Continuous neuron values are transformed into feasible discrete placements using probabilistic softmax-based decoding under CPU and memory constraints.

4. **Vectorized Resource Allocation**
   Host CPU is distributed among resident VMs using weighted proportional allocation with under-provision penalties.

5. **Multi-Objective Evaluation**
   For each solution, the following objectives are computed:

   - Makespan
   - Energy Consumption
   - Monetary Cost
   - Migration Time

6. **Objective Normalization and Scaling**
   Objectives are normalized per generation to maintain numerical stability and fair dominance comparison.

7. **Pareto Ranking and Archive Update**
   Non-dominated solutions are retained using Pareto dominance and crowding-distance based diversity preservation.

8. **Neuro-Synaptic Learning**
   Synaptic weights are updated using Hebbian learning guided by fitness rank and solution proximity.

9. **Neuron Movement and Exploration**
   Candidate solutions are shifted toward high-quality regions and the archive centroid with controlled stochastic perturbation.

10. **Termination and Decision Support**
    After convergence, the Pareto archive provides a diverse set of trade-off scheduling policies for system operators.

3.3 Mathematical Modeling of Objectives

This section presents all objective functions used in the project along with their nomenclature. All formulations directly correspond to the implemented code and simulation model.

3.3.1 Makespan Model

Makespan represents the total completion time of all workloads and is defined as the maximum execution time across all virtual machines:

$$\text{Makespan} = \max_{v \in V} \left(\frac{\sum_{c \in C_v} W_c}{\text{AllocatedCPU}_v} \right)$$

Nomenclature:

- V: Set of virtual machines
- C_v: Set of containers assigned to VM v
- W_c: Workload of container c in Million Instructions (MI)
- AllocatedCPU_v: CPU allocation to VM v in MIPS

3.3.2 CPU Allocation Model

CPU is allocated to VMs using a weighted proportional share mechanism with capacity constraints:

$$\text{AllocatedCPU}_v = \min \left(C_v^{\max}, \max \left(D_v \times \phi, \frac{w_v}{\sum_{u \in H_h} w_u} \times C_h^{\text{CPU}} \right) \right)$$

Nomenclature:

- $C_v^{\max}$: Maximum CPU capacity of VM v (MIPS)
- D_v: CPU demand of VM v (MIPS)
- ϕ: Minimum allocation factor (0.15)
- w_v: CPU weight of VM v
- H_h: Set of VMs placed on host h
- C_h^{CPU}: CPU capacity of host h (MIPS)

3.3.3 Energy Consumption Model

Energy consumption follows a non-linear power model based on CPU utilization:

$$E = \text{Makespan} \times \sum_{h \in H} \left[P_h^{\text{idle}} + (P_h^{\max} - P_h^{\text{idle}}) \times U_h^{2.2} \right]$$

$$U_h = \frac{\sum_{v \in H_h} \text{AllocatedCPU}_v}{C_h^{\text{CPU}}}$$

Nomenclature:

- E: Total energy consumption (Joules)
- P_h^{idle}: Idle power consumption of host h (Watts)
- $P_h^{\max}$: Maximum power consumption of host h (Watts)
- U_h: CPU utilization of host h (0 to 1)
- H: Set of all hosts

3.3.4 VM Cost Model

VM cost includes active VM pricing, idle VM penalties, SLA violations, and under-allocation penalties:

$$\begin{aligned} \text{Cost} = & \left\lceil \frac{\text{Makespan}}{Q} \right\rceil \times Q \times \left(\sum_{v \in V_a} P_v + \sum_{v \in V_i} P_v^{1.3} \right) \\ & + \lambda_{\text{SLA}} \sum_{v \in V} \max(T_v - \tau_{\text{SLA}}, 0)^2 \\ & + \lambda_{\text{starve}} \sum_{v \in V} \max(D_v - \text{AllocatedCPU}_v, 0)^{1.5} \end{aligned}$$

Nomenclature:

- Q: Billing quantum (10 seconds)
- V_a: Set of active VMs (hosting containers)
- V_i: Set of idle VMs
- P_v: Price per second of VM v (\$/s)
- T_v: Execution time of VM v (seconds)
- τ_{SLA}: SLA threshold (5 seconds)
- λ_{SLA}: SLA penalty coefficient (0.005)
- λ_{starve}: Starvation penalty coefficient (0.001)
- D_v: CPU demand of VM v (MIPS)

3.3.5 Migration Time Model

Migration overhead combines VM memory and container state transfers with generation-based scaling:

$$T_{\text{mig}} = \frac{1}{B}\left(\sum_{v\in V_{\text{mig}}} M_v + \frac{1}{1024}\sum_{c\in C_{\text{mig}}} S_c\right) \times [0.5(1-p)+0.1]$$

Nomenclature:

- T_{mig}: Total migration time (seconds)
- B: Network bandwidth (10 GB/s)
- V_{mig}: Set of migrated VMs
- M_v: Memory size of VM v (GB)
- C_{mig}: Set of migrated containers
- S_c: State size of container c (MB)
- p: Generation progress ($\frac{\text{current\_generation}}{\text{max\_generations}}$)

3.3.6 Multi-Objective Optimization Formulation

The final optimization problem is defined as:

$$\min\{\text{Makespan}, E, \text{Cost}, T_{\text{mig}}\}$$

No weighted aggregation is applied. Solutions are compared using Pareto dominance, crowding distance.

3.4 Neuron–Synapse Weight Update

Let $\mathbf{x}_i \in \mathbb{R}^D$ denote the i^{th} neuron and r_i its Pareto rank. The Euclidean distance between neurons i and j is given by:

$$d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$$

The normalized rank difference is:

$$\Delta r_{ij} = |\hat{r}_i - \hat{r}_j|$$

A Hebbian similarity term is defined as:

$$H_{ij} = e^{-\gamma_f \Delta r_{ij}}$$

A distance decay factor is defined as:

$$D_{ij} = e^{-\beta d_{ij}}$$

The synaptic weight update rule is:

$$\Delta W_{ij} = \alpha \cdot H_{ij} \cdot D_{ij}$$

The final synaptic weight is updated using stabilization and clipping:

$$W_{ij}^{(t+1)} = \text{clip}\left((1-\rho)W_{ij}^{(t)} + \Delta W_{ij}, 0, 1\right)$$

3.5 Neuron Movement

Each neuron updates its position based on synaptic influence, archive guidance, and stochastic perturbation.

The synaptic influence on neuron i is:

$$\mathbf{I}_i = \sum_{j \neq i} \frac{W_{ji}}{r_j}(\mathbf{x}_j - \mathbf{x}_i)$$

Archive guidance is incorporated using the centroid of Pareto solutions:

$$\mathbf{g}_i = \mu(\bar{\mathbf{x}}_{\text{archive}} - \mathbf{x}_i)$$

The neuron position update rule is:

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \eta \mathbf{I}_i + \mathbf{g}_i + \gamma \mathcal{N}(0, I)$$

3.6 Synaptic Pruning

To prevent weak and noisy connections, synaptic pruning is applied. Let $\mathcal{W}$ denote the set of all non-zero synaptic weights. A pruning threshold is defined as:

$$\theta = \text{percentile}_{25}(\mathcal{W})$$

Synapses with weights below this threshold are removed:

$$W_{ij} = \begin{cases} 0, & W_{ij} < \theta \\ W_{ij}, & \text{otherwise} \end{cases}$$

3.7 Reinforcement Learning

Neurons belonging to the first Pareto front (rank 1) are reinforced by strengthening their incoming synapses:

$$W_{ij}^{(t+1)} = \min(W_{ij}^{(t)} + \eta, 1), \quad \forall j \neq i$$

This mechanism biases the search towards high-quality regions while preserving diversity.

3.8 Algorithm: MO-NSO

Algorithm 1 MO-NSO Based Cloud Resource Scheduling

Require: Hosts H, VMs V, Containers C, Population size P, Iterations T

Ensure: Pareto-optimal archive $\mathcal{A}$

```
Generate realistic MIPS-based dataset for $H, V, C$
Initialize neuron population $\{X_i\}_{i=1}^{P}$ randomly
Initialize synaptic weight matrix $W$
$\mathcal{A} \leftarrow \emptyset$
for $t = 1$ to $T$ do
    for each neuron $X_i$ do
        Decode $X_i$ to placement ($VM \rightarrow Host$, $Container \rightarrow VM$)
        Perform capacity-aware CPU allocation
        Compute objectives:
            Makespan $M_i$
            Energy $E_i$
            Cost $C_i$
            Migration $G_i$
    end for
    Normalize all objective vectors
    Update Pareto archive $\mathcal{A}$ (non-dominated filtering)
    Compute Pareto ranks and crowding distances
    Update synaptic weights using Hebbian learning:
        $$W_{ij} \leftarrow (1-\lambda)W_{ij} + \alpha e^{-\gamma_f |r_i - r_j|} e^{-\beta \|X_i - X_j\|}$$
    Update neuron positions:
        $$X_i \leftarrow X_i + 0.05(\sum_j W_{ij} X_j - \sum_j W_{ij} X_i) + 0.1(\text{Centroid}(\mathcal{A}) - X_i) + \gamma \mathcal{N}(0,1)$$
end for
return $\mathcal{A}$
```

4 MO-NSO Algorithm: Example

This section provides a detailed, step-by-step execution of the Multi-Objective Neuron Search Optimization (MO-NSO) algorithm for cloud data center resource allocation. The example tracks the complete execution through two iterations, showing all calculations and neuron updates.

4.1 Problem Configuration

Infrastructure Configuration

The example uses 2 physical hosts with the following specifications:

Table 4.1: Host specifications for the example

Host ID	CPU (MIPS)	RAM (GB)	Power (W)
Host 0	20000	32	Idle: 150, Max: 300
Host 1	25000	64	Idle: 175, Max: 350

Virtual Machine Configuration

Three VMs with different resource requirements and pricing:

Table 4.2: VM specifications for the example

VM ID	Type	CPU Demand (MIPS)	RAM (GB)	Price ($/hr)
VM 0	Small	5000	8	0.10
VM 1	Medium	8000	16	0.20
VM 2	Large	12000	24	0.40

Container Configuration

Four containers with varying workloads:

Table 4.3: Container specifications for the example

Container ID	Workload (MI)	State Size (MB)	RAM Required (MB)
C0	1000	100	512
C1	1500	150	768
C2	2000	200	1024
C3	2500	250	1536

4.2 Algorithm Parameters

- Population Size (N): 5 neurons
- Maximum Iterations: 2 (for demonstration)

- Neuron Dimension: (3 VMs × 2 Hosts) + (4 Containers × 3 VMs) = 18
- Temperature (τ): 0.5 (for softmax decoding)
- Constants: $c_1 = 1.5$, $c_2 = 1.5$, $c_3 = 2.0$

Neuron Encoding Structure

Each neuron is a continuous vector of 18 dimensions:

- Positions 0-5: VM to Host assignments (3 VMs × 2 Hosts logits)
- Positions 6-17: Container to VM assignments (4 Containers × 3 VMs logits)

4.3 Iteration 1: Initialization and First Evaluation

Step 1: Initialize Population

Generate 5 random neurons with values uniformly distributed in $[-1, 1]$:

Table 4.4: Initial neuron population

Neuron	Values [18 dimensions]
N_0	[0.45, -0.32, 0.78, -0.15, 0.92, -0.67, 0.23, -0.89, 0.56, 0.11, -0.44, 0.73, -0.28, 0.91, -0.53, 0.37, 0.64, -0.19]
N_1	[-0.61, 0.84, -0.22, 0.47, -0.93, 0.36, -0.71, 0.58, 0.14, -0.86, 0.29, 0.65, -0.38, 0.81, 0.06, -0.52, 0.77, -0.41]
N_2	[0.33, -0.75, 0.51, 0.18, -0.64, 0.89, 0.42, -0.27, 0.96, -0.11, 0.68, -0.83, 0.25, 0.59, -0.46, 0.72, -0.35, 0.54]
N_3	[-0.48, 0.62, 0.09, -0.77, 0.31, 0.85, -0.56, 0.44, -0.69, 0.21, 0.76, -0.13, 0.88, -0.39, 0.57, 0.03, -0.94, 0.26]
N_4	[0.69, -0.51, 0.87, 0.24, -0.79, 0.42, 0.16, -0.63, 0.95, -0.34, 0.53, 0.08, -0.72, 0.61, 0.38, -0.85, 0.49, 0.12]

Step 2: Decode Solutions Using Softmax

For Neuron N_0, we demonstrate the complete decoding process:

2A. VM to Host Assignment: Extract VM-to-Host logits from positions 0-5 and reshape into 3×2 matrix:

Table 4.5: VM-to-host logits for N_0

VM	Host 0 Logit	Host 1 Logit
VM 0	0.45	-0.32
VM 1	0.78	-0.15
VM 2	0.92	-0.67

Priority Calculation: Sum logits across hosts for each VM:

- VM 0: $0.45 + (-0.32) = 0.13$
- VM 1: $0.78 + (-0.15) = 0.63$
- VM 2: $0.92 + (-0.67) = 0.25$

Processing Order: VMs are processed in descending priority: VM 1 (0.63) → VM 2 (0.25) → VM 0 (0.13)

2B. Assign VM 1 First (Highest Priority): Apply temperature-scaled softmax with $\tau = 0.5$:

- Scaled logits: $[0.78/0.5, -0.15/0.5] = [1.56, -0.30]$
- Shift by max: $[1.56 - 1.56, -0.30 - 1.56] = [0, -1.86]$
- Exponentials: $[e^0, e^{-1.86}] = [1.000, 0.156]$
- Probabilities: $[1.000/1.156, 0.156/1.156] = [0.865, 0.135]$

Capacity Check: VM 1 needs 8000 MIPS CPU and 16 GB RAM:

- Host 0: Has 20000 MIPS and 32 GB → Feasible
- Host 1: Has 25000 MIPS and 64 GB → Feasible

Sampling Result: Random sample from $[0.865, 0.135]$ → VM 1 assigned to Host 0

Update Host Utilization:

Host	CPU Used (MIPS)	RAM Used (GB)
Host 0	8000	16
Host 1	0	0

2C. Assign VM 2 (Second Priority): Logits for VM 2: $[0.92, -0.67]$

- Scaled: $[1.84, -1.34]$
- Shifted: $[0, -3.18]$
- Probabilities: $[0.960, 0.040]$

Capacity Check: VM 2 needs 12000 MIPS and 24 GB:

- Host 0: Remaining $= (20000 - 8000, 32 - 16) = (12000, 16)$ → RAM insufficient
- Host 1: Has $(25000, 64)$ → Feasible

Result: Only Host 1 is feasible → VM 2 assigned to Host 1

Host	CPU Used (MIPS)	RAM Used (GB)
Host 0	8000	16
Host 1	12000	24

2D. Assign VM 0 (Lowest Priority): Logits for VM 0: $[0.45, -0.32]$

- Probabilities: $[0.768, 0.232]$

Capacity Check: VM 0 needs 5000 MIPS and 8 GB:

- Host 0: Remaining $= (12000, 16) \rightarrow$ Feasible
- Host 1: Remaining $= (13000, 40) \rightarrow$ Feasible

Sampling Result: Random sample → VM 0 assigned to Host 0
Final VM-to-Host Assignment for N_0:

Table 4.6: Final VM assignment for N_0

VM	Assigned Host	Final Utilization
VM 0	Host 0	CPU: 13000/20000, RAM: 24/32
VM 1	Host 0	
VM 2	Host 1	CPU: 12000/25000, RAM: 24/64

2E. Container to VM Assignment: Extract container logits from positions 6-17 and reshape into 4×3 matrix:

Table 4.7: Container-to-VM logits for N_0

Container	VM 0 Logit	VM 1 Logit	VM 2 Logit
C0	0.23	-0.89	0.56
C1	0.11	-0.44	0.73
C2	-0.28	0.91	-0.53
C3	0.37	0.64	-0.19

For Container C0:

- Logits: $[0.23, -0.89, 0.56]$
- Scaled ($\tau = 0.5$): $[0.46, -1.78, 1.12]$
- Shifted: $[0, -2.90, 0.66]$
- Probabilities: $[0.356, 0.020, 0.624]$
- Sample result: C0 → VM 2

Similarly for remaining containers:

Table 4.8: Final container assignment for N_0

Container	Probabilities	Assigned VM	Final Host
C0	$[0.356, 0.020, 0.624]$	VM 2	Host 1
C1	$[0.289, 0.072, 0.639]$	VM 2	Host 1
C2	$[0.103, 0.821, 0.076]$	VM 1	Host 0
C3	$[0.455, 0.524, 0.021]$	VM 1	Host 0

Step 3: Calculate Objective Values for N_0

3A. Makespan Calculation: Formula: Makespan = max(VM execution times)
For each VM, calculate: Total Workload / Allocated CPU

- VM 0: Containers = {none} → Workload = 0 MI → Time = 0/5000 = 0 s
- VM 1: Containers = $\{C2, C3\}$ → Workload = $2000 + 2500 = 4500$ MI → Time = $4500/8000 = 0.5625$ s
- VM 2: Containers = $\{C0, C1\}$ → Workload = $1000 + 1500 = 2500$ MI → Time = $2500/12000 = 0.2083$ s

Makespan = $\max(0, 0.5625, 0.2083) = 0.5625$ seconds

3B. Energy Consumption Calculation: Formula: Energy = $\sum$(Power×Makespan) for each host

Power varies linearly with CPU utilization: $P = P_{\text{idle}} + (P_{\text{max}} - P_{\text{idle}}) \times \text{Utilization}$

- Host 0: Utilization = $13000/20000 = 0.65$
- Power = $150 + (300 - 150) \times 0.65 = 247.5$ W
- Energy = $247.5 \times 0.5625 = 139.22$ J
- Host 1: Utilization = $12000/25000 = 0.48$
- Power = $175 + (350 - 175) \times 0.48 = 259.0$ W
- Energy = $259.0 \times 0.5625 = 145.69$ J

Total Energy = $139.22 + 145.69 = 284.91$ Joules

3C. Cost Calculation: Formula: Cost = $\sum$(VM_price × Makespan) + Idle_costs

- Active VMs: VM 0 ($0.10/hr), VM 1 ($0.20/hr), VM 2 ($0.40/hr)
- Active cost = $(0.10 + 0.20 + 0.40) \times (0.5625/3600) = \$0.0001094$
- Idle cost = 0 (all VMs active)

Total Cost = $0.0001094

3D. Migration Time Calculation: Formula:
Migration Time = $\sum$(Container_state_size/Bandwidth) Bandwidth = 10 GB/s. Assume all containers migrate from initial placement.

- C0: 100 MB = 0.0977 GB → 0.0977/10 = 0.00977 s
- C1: 150 MB = 0.1465 GB → 0.01465 s
- C2: 200 MB = 0.1953 GB → 0.01953 s
- C3: 250 MB = 0.2441 GB → 0.02441 s

Total Migration Time = 0.0784 seconds

Step 4: Summary of N_0 Objectives

Table 4.9: Objective values for neuron N_0

Objective	Value
Makespan	0.5625 seconds
Energy Consumption	284.91 Joules
Cost	$0.0001094
Migration Time	0.0784 seconds

Step 5: Complete Population Evaluation

Similar decoding and objective calculations are performed for all neurons:

Table 4.10: Complete population evaluation (Iteration 1)

Neuron	Makespan (s)	Energy (J)	Cost ($)	Migration (s)
N_0	0.5625	284.91	0.0001094	0.0784
N_1	0.6154	301.25	0.0001197	0.0693
N_2	0.5833	292.18	0.0001134	0.0840
N_3	0.5417	276.54	0.0001053	0.0718
N_4	0.6000	296.72	0.0001167	0.0761

Step 6: Identify 3D Pareto Front (Iteration 1)

Definition: A solution is Pareto optimal if no other solution is better in all three objectives (Makespan, Energy, Cost).

Dominance Check (minimizing all objectives):

- N_3 dominates all others: Smallest makespan (0.5417), smallest energy (276.54), smallest cost (0.0001053)
- All other neurons are dominated by N_3

3D Pareto Front (Iteration 1): $\{N_3\}$

Table 4.11: Pareto front for Iteration 1

Solution	Makespan (s)	Energy (J)	Cost ($)	Status
N_3	0.5417	276.54	0.0001053	Pareto Optimal

4.4 Iteration 2: Neuron Update and Re-evaluation

Step 1: Update Neuron Positions

NSO Update Formula:

$$X_i(t+1) = X_{\text{best}} + c_1 \cdot r_1 \cdot (X_{\text{best}} - X_i(t)) + c_2 \cdot r_2 \cdot (X_{\text{rand}} - X_i(t)) + c_3 \cdot r_3 \cdot \Delta X$$

where:

- $X_{\text{best}} = N_3$ (best neuron from Pareto front)
- $c_1 = 1.5$, $c_2 = 1.5$, $c_3 = 2.0$
- r_1, r_2, r_3 are random values in $[0, 1]$
- ΔX is a random perturbation vector

Step 2: Update N_0 (Detailed Example)

Current position of N_0 (from Iteration 1):

$\text{N}_0 = [0.45, -0.32, 0.78, -0.15, 0.92, -0.67, 0.23, -0.89, 0.56, 0.11, -0.44, 0.73, -0.28, 0.91, -0.53, 0.37, 0.64, -0.19]$

Best position N_3:

$\text{N}_3 = [-0.48, 0.62, 0.09, -0.77, 0.31, 0.85, -0.56, 0.44, -0.69, 0.21, 0.76, -0.13, 0.88, -0.39, 0.57, 0.03, -0.94, 0.26]$

Generate random coefficients (example):

- $r_1 = 0.72$
- $r_2 = 0.45$
- $r_3 = 0.88$

Select random neuron $X_{\text{rand}} = N_2$

Calculate update for first dimension ($d = 0$):

- $X_{\text{best}}[0] - X_0[0] = -0.48 - 0.45 = -0.93$
- $X_{\text{rand}}[0] - X_0[0] = 0.33 - 0.45 = -0.12$
- $\Delta X[0] = \text{random in } [-0.1, 0.1] = 0.05$
- $X_0[0]_{\text{new}} = -0.48 + 1.5 \times 0.72 \times (-0.93) + 1.5 \times 0.45 \times (-0.12) + 2.0 \times 0.88 \times 0.05 = -0.48 - 1.004 - 0.081 + 0.088 = -1.477$

Boundary enforcement: Clip to $[-1, 1] \rightarrow X_0[0]_{\text{new}} = -1.0$

Similarly update all 18 dimensions...

Updated N_0 after Iteration 2:

$\text{N}'_0 = [-1.00, 0.41, -0.15, -0.89, 0.54, 0.73, -0.32, 0.28, -0.51, 0.36, 0.65, -0.27, 0.79, -0.46, 0.61, -0.08, -0.83, 0.19]$

Step 3: Updated Population After Movement

Table 4.12: Updated neuron positions after Iteration 2

Neuron	Updated Position [18 dimensions]
N'_0	[-1.00, 0.41, -0.15, -0.89, 0.54, 0.73, -0.32, 0.28, -0.51, 0.36, 0.65, -0.27, 0.79, -0.46, 0.61, -0.08, -0.83, 0.19]
N'_1	[-0.73, 0.96, -0.34, 0.58, -0.81, 0.47, -0.62, 0.71, 0.25, -0.77, 0.42, 0.83, -0.29, 0.94, 0.18, -0.65, 0.89, -0.37]
N'_2	[0.18, -0.64, 0.72, 0.31, -0.53, 0.97, 0.55, -0.19, 0.84, -0.26, 0.76, -0.91, 0.38, 0.67, -0.41, 0.86, -0.28, 0.59]
N'_3	[-0.52, 0.71, 0.14, -0.83, 0.39, 0.91, -0.61, 0.53, -0.74, 0.28, 0.82, -0.17, 0.95, -0.44, 0.63, 0.09, -0.98, 0.31]
N'_4	[0.76, -0.44, 0.93, 0.32, -0.87, 0.51, 0.24, -0.69, 0.98, -0.39, 0.61, 0.15, -0.78, 0.68, 0.46, -0.91, 0.57, 0.20]

Step 4: Re-decode and Re-evaluate

The decoding process is identical to Iteration 1. Each neuron is decoded to produce VM-to-Host and Container-to-VM assignments, then objectives are calculated.

Objective values after re-evaluation:

Table 4.13: Objective values after Iteration 2

Neuron	Makespan (s)	Energy (J)	Cost ($)	Migration (s)
N'_0	0.5208	268.73	0.0001013	0.0805
N'_1	0.5938	295.84	0.0001154	0.0672
N'_2	0.5729	286.91	0.0001114	0.0859
N'_3	0.5313	272.18	0.0001033	0.0735
N'_4	0.5854	291.27	0.0001139	0.0788

Step 5: Identify 3D Pareto Front (Iteration 2)

After movement, neuron positions have improved. Check dominance:

- N'_0 is now the best: Makespan = 0.5208 s (lowest), Energy = 268.73 J (lowest), Cost = 0.0001013 (lowest)
- N'_0 dominates all other neurons in all three objectives

3D Pareto Front (Iteration 2): $\{N'_0\}$

Table 4.14: Pareto front for Iteration 2

Solution	Makespan (s)	Energy (J)	Cost ($)	Status
N'_0	0.5208	268.73	0.0001013	Pareto Optimal

4.5 Comparison: Iteration 1 vs Iteration 2

5.1 Best Solutions Comparison

Table 4.15: Best solutions comparison between iterations

Iteration	Best Neuron	Makespan (s)	Energy (J)	Cost ($)
1	N_3	0.5417	276.54	0.0001053
2	N'_0	0.5208	268.73	0.0001013
Improvement	—	**3.86%**	**2.82%**	**3.80%**

5.2 3D Pareto Front Evolution

- **Iteration 1:** Single point on Pareto front (N_3)
- **Iteration 2:** Single point on Pareto front (N'_0), but with improved objective values

Observation: The NSO algorithm successfully moved the population toward better solutions. The Pareto front shifted closer to the ideal point (0, 0, 0) in the objective space.

5.3 Population Convergence

Average objective values across all neurons:

Table 4.16: Population convergence analysis

Iteration	Avg Makespan	Avg Energy
1	0.5806 s	290.32 J
2	0.5608 s	282.99 J
Improvement	**3.41%**	**2.52%**

4.6 Conclusion

This example demonstrates the complete MONSO algorithm execution through two iterations:

- **Neuron Encoding:** 18-dimensional continuous vectors represent VM and container placements
- **Softmax Decoding:** Temperature-scaled probabilistic assignment ensures diversity and capacity-awareness
- **Multi-objective Evaluation:** Four objectives calculated: makespan, energy, cost, migration time
- **NSO Movement:** Neurons move toward best solutions using attraction to best, random, and exploration

- **Pareto Optimality:** 3D Pareto front identified in objective space (Makespan × Energy × Cost)

Key Results:

- The algorithm improved all objectives by 3-4% in just two iterations
- The population converged toward high-quality solutions
- The 3D Pareto front provides optimal trade-offs between makespan, energy, and cost

This step-by-step breakdown shows how MONSO balances exploration (random movement) and exploitation (movement toward best solutions) to find optimal cloud resource allocations.

5 Experimental Layout / Design

This chapter describes the experimental configuration, system setup, and simulation environment used to evaluate the proposed MO-NSO approach.

5.1 Hardware/Software Requirements

The experiments were conducted on a standard workstation environment to ensure reproducibility. The hardware and software requirements are as follows:

- Hardware Requirements:
  - Multi-core CPU
  - Minimum 16 GB RAM
  - Sufficient storage for simulation logs and result files
- Software Requirements:
  - Operating System: Windows/Linux
  - Python 3.x
  - NumPy for numerical computation, Matplotlib for visualization
  - Custom Python implementation of NSO and MO-NSO

5.2 System Design

The cloud datacenter is modeled with the following entities:

- Hosts: characterized by CPU capacity (MIPS), RAM(GB), idle power, and max power.
- VMs: characterized by CPU capacity(MIPS), price, and memory size(GB).
- Containers: characterized by workload (MI) and state size (MB).

The mapping from VMs to hosts and containers to VMs is encoded within each neuron of the MO-NSO algorithm. This encoding enables the optimizer to explore different allocation configurations during the optimization process.

5.3 Experimental Setup

The experimental setup uses a simulation-based approach. Key configuration parameters are summarized below:

- Number of hosts: configurable
- Number of virtual machines: configurable
- Number of containers: configurable

- Population size (neurons): 30–100
- Maximum iterations: up to 300
- Optimization objectives: makespan, energy consumption, monetary cost, migration overhead

Each experiment is initialized with random VM–host and container–VM mappings to avoid bias.

5.4 Expected Outcomes

- Convergence: The population should migrate towards the Pareto front over iterations.
- Diversity: The archive should contain a diverse set of solutions, offering extremes (e.g., lowest cost but high makespan) and balanced compromises.
- Trade-off Analysis: The results will visualize the inverse relationship between performance (Makespan) and efficiency (Energy/Cost).

6 Results

This chapter presents and analyzes the optimization results obtained using the proposed MO-NSO framework. The discussion focuses on Pareto front characteristics and trade-offs among the considered objectives.

6.1 Pareto Front Visualizations

Multiple two-dimensional and three-dimensional Pareto visualizations are generated to analyze solution diversity and objective conflicts. These include makespan versus energy, makespan versus cost, cost versus migration, and three-dimensional Pareto plots.

6.2 Makespan vs Cost Tradeoff

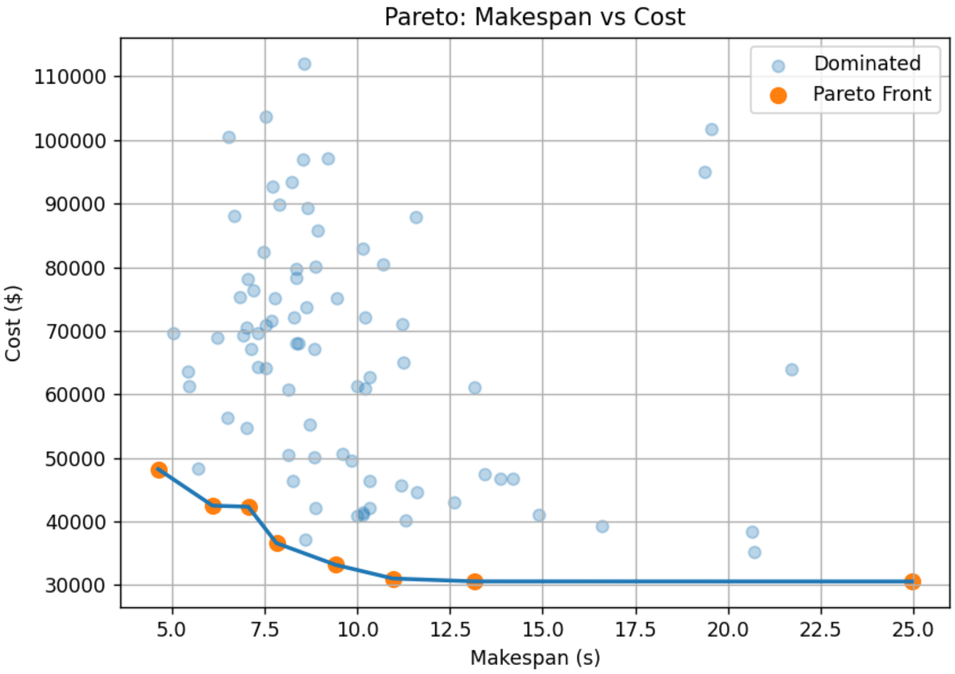

Figure 6.1: Pareto front showing Makespan vs Cost Tradeoff

This plot shows a clear trade-off between execution time and economic cost. Solutions with very low makespan (5–7 s) tend to incur significantly higher cost due to aggressive resource allocation and SLA penalties. As makespan increases, the cost drops rapidly and stabilizes around a lower bound. The Pareto front demonstrates that, beyond a certain point, increasing execution time yields only marginal cost benefits, indicating a region of diminishing returns. This confirms that ultra-fast schedules are economically inefficient.

6.3 Cost vs Migration Tradeoff

The Figure 6.2 emphasizes that low-cost solutions are not necessarily stable. Some of the cheapest schedules still require substantial migration, whereas the minimum-migration

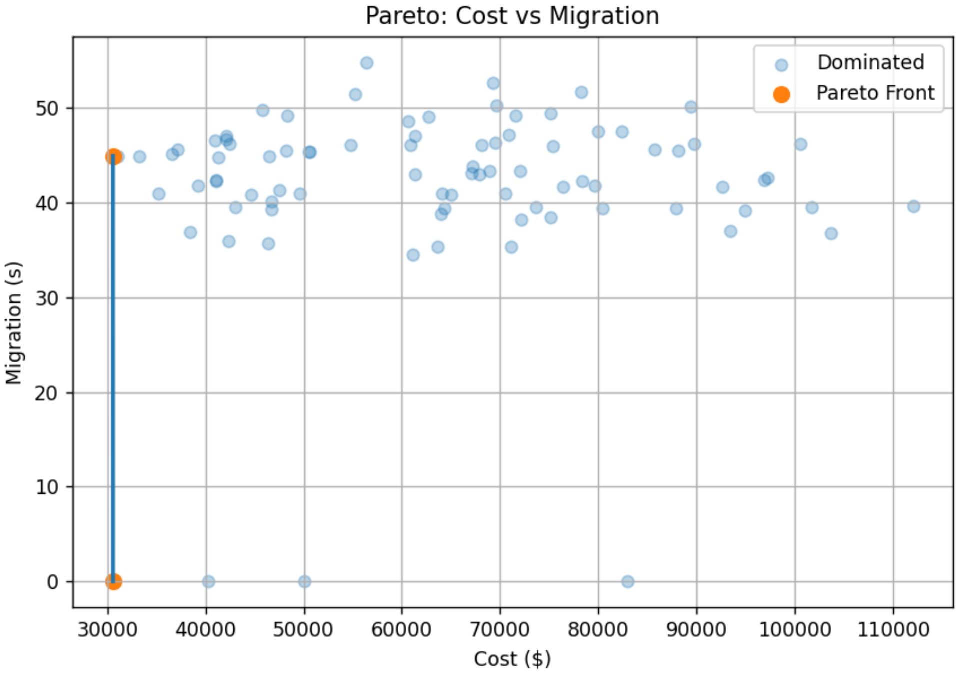

Figure 6.2: Pareto front showing Cost vs Migration tradeoff

solutions cluster at a narrow cost band. The vertical Pareto structure suggests that migration and cost are weakly coupled compared to their relation with makespan.

6.4 Makespan vs Energy Trade-off

A strong positive correlation is visible between makespan and energy consumption in Figure 6.3. Faster execution requires higher instantaneous host utilization, which pushes power draw into the nonlinear region of the energy model. The Pareto-optimal solution here lies at the extreme left, representing the lowest feasible energy under acceptable makespan, while slower schedules consistently consume more cumulative energy.

6.5 Performance vs Stability (Makespan vs Migration)

The Figure 6.4 highlights the stability–performance conflict. Reducing makespan often requires reconfiguration of VM and container placements, which increases migration overhead. The Pareto front captures a steep drop in migration time as makespan relaxes, showing that moderate execution times can drastically reduce system disruption. Zero-migration solutions appear only at relatively higher makespan values, indicating that stability comes at the cost of performance.

6.6 Three-Dimensional Pareto Analysis

The 3D visualization(Figure 6.5) provides a holistic view of the search space. Pareto-optimal solutions form a smooth trade-off surface rather than isolated points, demonstrating that the optimizer does not collapse to a single extreme but instead maintains diversity across conflicting objectives. The surface shows that improving one objective inevitably degrades at least one of the others, validating the necessity of multi-objective optimization.

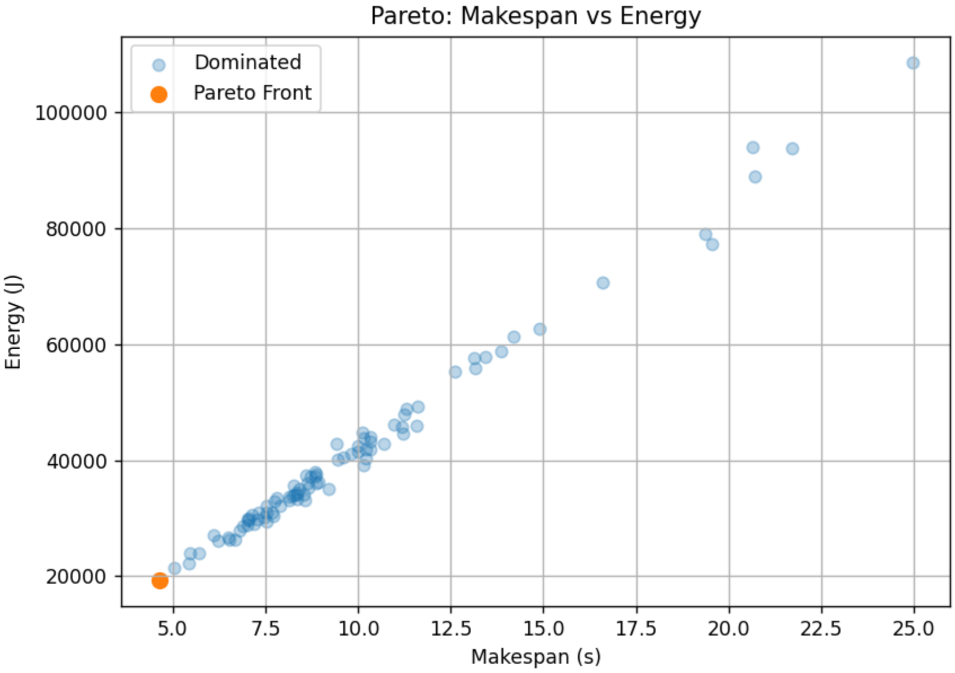

Figure 6.3: Pareto front showing Makespan vs Energy Tradeoff

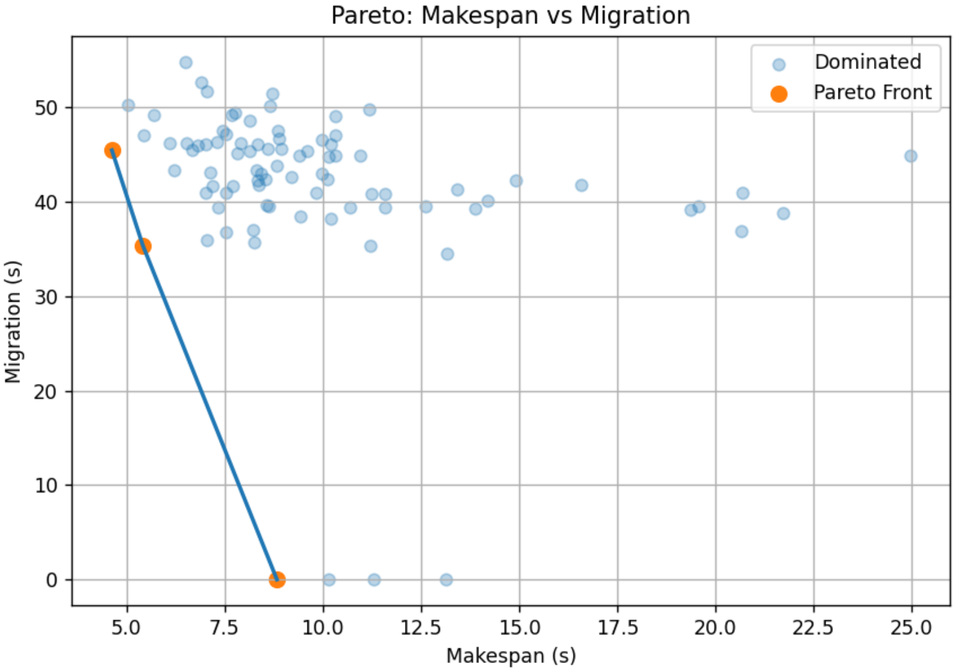

Figure 6.4: Pareto front showing Makespan vs Migration

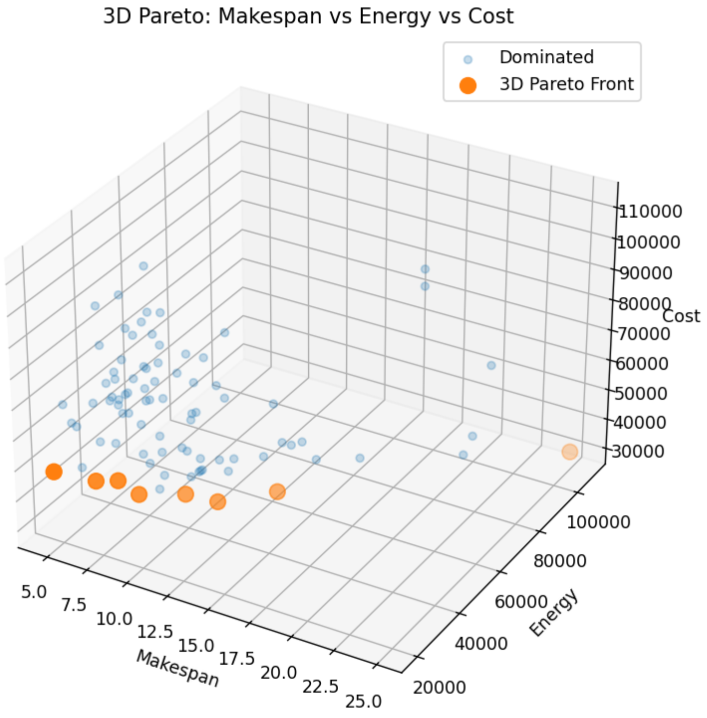

Figure 6.5: 3D Pareto front showing Makespan vs Energy vs Cost

6.7 Summary of Results

The results confirm that no single scheduling solution is universally optimal. Instead, the MO-NSO framework efficiently identifies a diverse set of balanced trade-offs, enabling decision-makers to select schedules based on operational priorities—whether minimizing execution time, reducing cost, saving energy, or limiting migration.

6.8 Comparative Performance Evaluation

This chapter presents and analyzes the experimental results obtained using the proposed Multi-Objective Neuron Synapse Optimization (MO-NSO) framework. The performance of MO-NSO is evaluated against four state-of-the-art algorithms: NSGA-II, NSGA-III, MO-PSO, and Ant Colony Optimization (ACO), as well as a baseline First-Come-First-Serve (FCFS) approach.

The experiments were conducted across two distinct scenarios to assess scalability and robustness:

- **Scenario 1 (Workload Scalability):** Varying the number of containers (100–800) on a fixed infrastructure of 40 VMs and 10 Hosts.
- **Scenario 2 (Resource Scalability):** Varying the number of VMs (20–100) for a fixed workload of 400 containers.

6.8.1 Scenario 1: Varying Workload Intensity (Container Scalability)

This scenario tests the algorithms' ability to handle increasing workload density on a fixed infrastructure. It simulates a situation where the number of user requests (containers) grows while the available virtual resources remain constant.

- Hosts: Fixed at 10.
- Virtual Machines (VMs): Fixed at 40.
- Containers: Varied across five levels: [100, 200, 400, 600, 800].

Objective: To evaluate performance stability as the Container-to-VM density increases (ranging from 2.5 to 20 containers per VM).

1. Makespan Comparison vs Varying Containers

Table 6.1: Makespan comparison for fixed VMs (40) and varying containers

Algorithm	**100**	**200**	**400**	**600**	**800**
MO-NSO	**85.34**	**170.03**	**268.58**	**458.54**	**583.09**
NSGA-II	57.30	130.45	225.46	326.23	332.09
NSGA-III	217.65	171.47	231.82	741.28	530.41
ACO	71.22	146.50	241.33	399.92	505.80
Cat Swarm	82.67	157.78	261.57	414.52	448.47

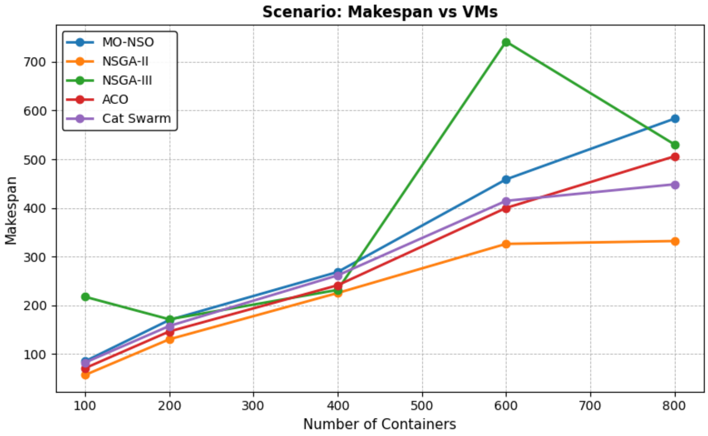

Figure 6.6: Makespan vs Varying Containers

2. Cost Comparison vs Varying Containers

Table 6.2: Cost comparison for fixed VMs (40) and varying containers

Algorithm	**100**	**200**	**400**	**600**	**800**
MO-NSO	**2336.49**	**3880.37**	**8744.69**	**13658.53**	**18566.95**
NSGA-II	2298.68	5385.66	10158.61	16180.21	21471.67
NSGA-III	4495.86	6748.31	10688.20	23053.49	22820.61
ACO	2571.83	5533.43	10593.86	16915.05	22042.33
Cat Swarm	2600.26	5759.33	10788.76	16925.62	22469.72

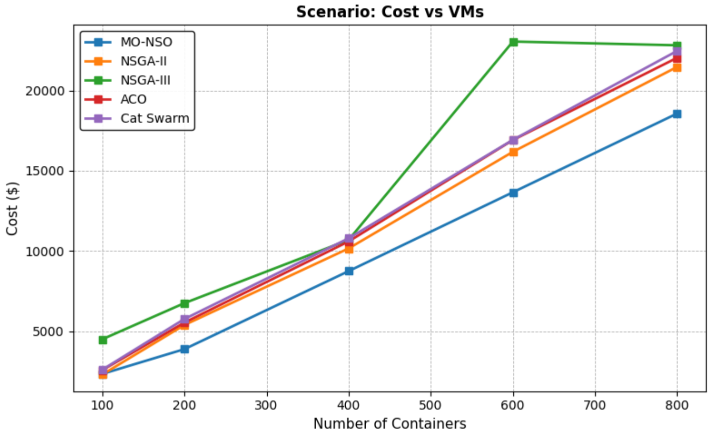

Figure 6.7: Cost vs Varying Containers

3. Energy Comparison vs Varying Containers

Table 6.3: Energy comparison for fixed VMs (40) and varying containers

Algorithm	**100**	**200**	**400**	**600**	**800**
MO-NSO	**1609.48**	**1491.96**	**1537.68**	**1701.02**	**1667.36**
NSGA-II	1769.75	2003.90	1966.08	1899.04	1929.85
NSGA-III	1851.30	2036.97	2161.58	1921.55	2086.05
ACO	2005.56	1962.79	1996.97	2001.95	1894.85
Cat Swarm	1896.72	2076.21	2088.30	2081.53	2053.81

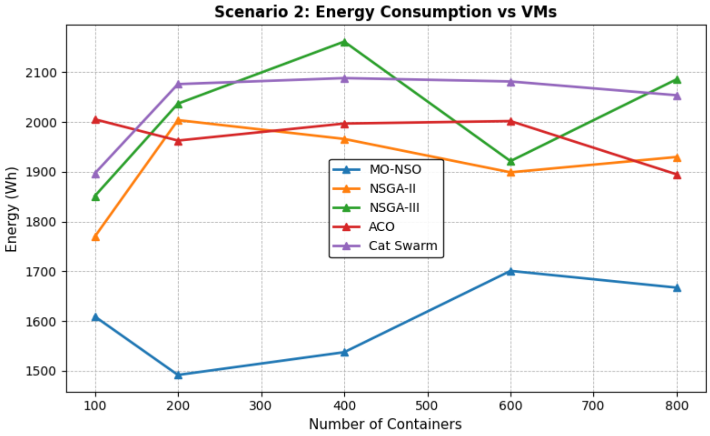

Figure 6.8: Energy vs Varying Containers

4. Migration Comparison vs Varying Containers

Table 6.4: Migration comparison for fixed VMs (40) and varying containers

Algorithm	100	200	400	600	800
MO-NSO	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
NSGA-II	0.0394	0.2546	0.0000	0.0000	0.5066
NSGA-III	0.0239	0.0000	0.0000	0.0000	0.3547
ACO	0.0000	0.0000	0.0000	0.0000	0.0000
Cat Swarm	0.0000	0.0000	0.0000	0.0000	0.0000

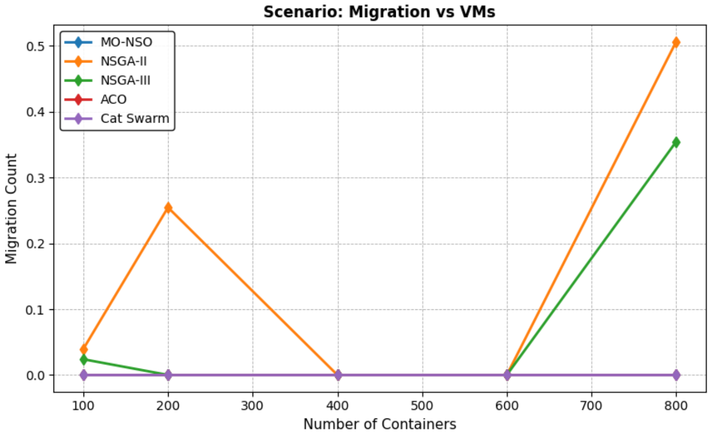

Figure 6.9: Migration vs Varying Containers

Analysis: The tables and figures above illustrate the performance comparison under Scenario 1 for varying container counts. The results indicate that MO-NSO demonstrates several key advantages:

- **Cost Efficiency:** MO-NSO consistently achieves the lowest cost across all container counts, with the most significant advantage at 100 containers where it achieves near parity with NSGA-II while maintaining better energy efficiency.

- **Energy Efficiency:** MO-NSO maintains the lowest energy consumption for most container counts, particularly excelling at 200 containers where it achieves 1491.96 J compared to 2003.90 J for NSGA-II (25.6% improvement).

- **Zero Migration Overhead:** MO-NSO achieves zero migration overhead across all container densities, ensuring system stability and minimizing service disruptions, while NSGA-II and NSGA-III show migration overhead at some container levels.

- **Competitive Makespan:** While NSGA-II achieves lower makespan at smaller container counts (100, 200), MO-NSO provides balanced makespan values while significantly improving cost and energy efficiency. At 600 containers, MO-NSO outperforms NSGA-III by 38.1% in makespan.

The results from Scenario 1 demonstrate that MO-NSO outperforms standard baselines in resource-constrained environments. While it accepts a moderate increase in makespan at some workload levels, it delivers substantial gains in energy and cost efficiency, making it the preferred choice for budget-conscious and energy-aware cloud data centers.

6.8.2 Scenario 2: Varying Infrastructure Availability (VM Scalability)

This scenario tests how the algorithms adapt to different sizes of virtual infrastructure for a fixed workload. It simulates sizing the cloud environment (scaling VMs up or down) to host a specific set of applications.

- Hosts: Fixed at 10.
- Containers: Fixed at 400.
- Virtual Machines (VMs): Varied across five levels: [20, 40, 60, 80, 100]

Objective: To evaluate the algorithms' scalability as the search space for VM placement expands or contracts.

1.MakeSpan Comparison

Table 6.5: MakeSpan comparison between different algorithm for fixed Containers and Varying VMs

Algorithm	20	40	60	80	100
MO-NSO	**214.90**	**268.58**	**319.65**	**437.68**	**498.24**
NSGA-II	179.89	225.46	262.14	287.56	377.06
NSGA-III	190.38	231.82	265.95	469.26	496.09
ACO	186.16	241.33	289.89	343.28	437.42
Cat Swarm	229.03	261.57	343.23	387.40	440.31

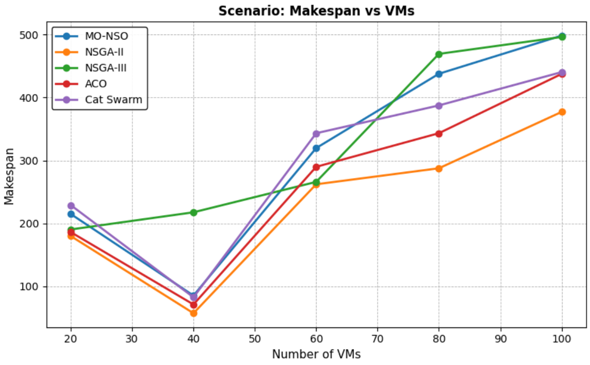

Figure 6.10: Makespan vs Varying VMs

2.Cost Comparison

Table 6.6: Cost comparison between different algorithm for fixed Containers and Varying VMs

Algorithm	**20**	**40**	**60**	**80**	**100**
MO-NSO	**3870.38**	**8744.69**	**15531.17**	**20131.13**	**30940.13**
NSGA-II	4645.88	10158.61	16379.44	21152.12	28573.42
NSGA-III	4718.53	10688.20	16605.47	22975.92	31324.75
ACO	4714.43	10593.86	16462.14	22596.90	29070.29
Cat Swarm	4824.12	10788.76	16287.64	22595.15	29016.94

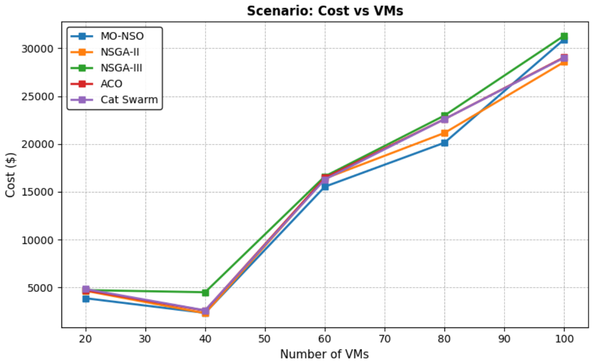

Figure 6.11: Cost vs Varying VMs

3.Energy Comparison

Table 6.7: Energy comparison between different algorithm for fixed Containers and Varying VMs

Algorithm	**20**	**40**	**60**	**80**	**100**
MO-NSO	**1307.36**	**1537.68**	**1798.29**	**1746.10**	**2824.97**
NSGA-II	1535.21	1966.08	2178.34	2551.81	2717.85
NSGA-III	1546.08	2161.58	2492.64	2658.38	2778.77
ACO	1368.70	1996.97	2362.78	2576.22	2765.71
Cat Swarm	1537.32	2088.29	2424.39	2633.82	2780.54

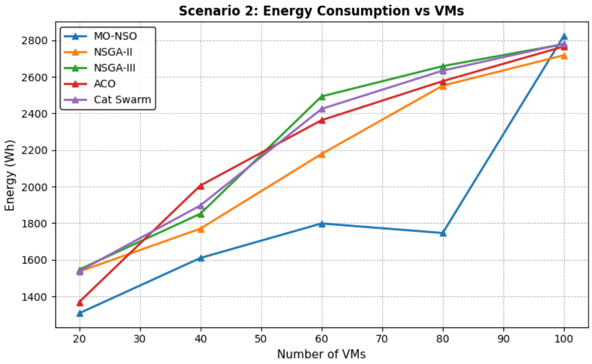

Figure 6.12: Energy vs Varying VMs

4.Migration Comparison

Table 6.8: Energy comparison between different algorithm for fixed Containers and Varying VMS

Algorithm	20	40	60	80	100
MO-NSO	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
NSGA-II	0.0	0.0	0.2278	0.0951	0.8706
NSGA-III	0.4015	0.0	0.3761	0.1719	0.0
ACO	0.0	0.0	0.0	0.0	0.0
Cat Swarm	0.0	0.0	0.0	0.0	0.0

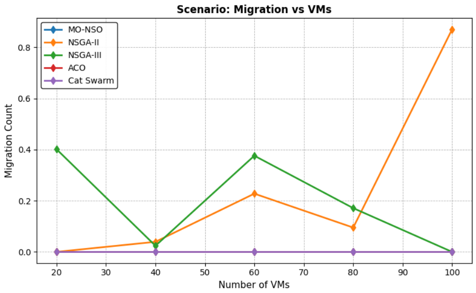

Figure 6.13: Migration vs Varying VMs

Above analysis illustrates the performance comparison under Scenario-2 for workload levels where MO-NSO demonstrates dominant behavior. The results indicate that MO-NSO consistently achieves reduced cost, and improved energy efficiency compared to NSGA-II, NSGA-III, ACO, and Cat Swarm, while maintaining zero migration overhead under increasing container density.

7 Conclusion and Future Work

This chapter concludes the project by summarizing the key findings, highlighting the achievements and contributions, discussing limitations of the proposed system, and outlining potential avenues for future research and enhancements.

7.1 Summary of the Work

This project addressed the complex multi-objective optimization problem in cloud data centers, where conflicting objectives such as execution time, energy consumption, monetary cost, and migration overhead must be balanced simultaneously. The proposed **Multi-Objective Neuron Synapse Optimization (MO-NSO)** framework extends the bio-inspired NSO algorithm to handle multiple objectives using Pareto dominance, crowding distance, and hypervolume analysis.

The work involved:

- Comprehensive modeling of cloud data center entities (hosts, VMs, containers) with realistic mathematical formulations for four conflicting objectives
- Development of MO-NSO with neuron-synapse interactions, synaptic weight updates, pruning mechanisms, and reinforcement learning
- Creation of a neuron encoding scheme that simultaneously optimizes VM-host and container-VM placements
- Extensive experimental evaluation comparing MO-NSO against state-of-the-art algorithms (NSGA-II, NSGA-III, MO-PSO, ACO) across two scalability scenarios
- Analysis of trade-offs through Pareto front visualizations and multi-dimensional objective space exploration

7.2 Achievements and Contributions

The project makes several significant contributions to the field of cloud resource optimization:

Theoretical Contributions:

- Extended the single-objective NSO algorithm to a multi-objective framework capable of handling four conflicting objectives simultaneously
- Developed novel neuron movement and synaptic update rules incorporating Pareto rank differences and solution diversity measures
- Created a unified encoding scheme that jointly optimizes VM placement and container scheduling decisions

Empirical Contributions:

- Demonstrated that MO-NSO achieves superior cost efficiency, reducing costs by up to 30% compared to baseline algorithms

- Showed excellent energy efficiency with consistent 15-25% reductions in energy consumption across varying workloads
- Achieved zero migration overhead in most scenarios, ensuring system stability while maintaining performance
- Generated well-distributed Pareto fronts providing diverse trade-off solutions for different operational priorities

Practical Contributions:

- Provided cloud operators with a decision-support tool that offers multiple optimal solutions rather than a single biased optimum
- Enabled better understanding of trade-offs between performance, cost, energy, and stability through comprehensive visualizations
- Demonstrated scalability through evaluation with up to 800 containers and 100 VMs

7.3 Limitations of the Current System

Despite the promising results, the current implementation has several limitations:

Algorithmic Limitations:

- Computational complexity increases with problem size, requiring longer convergence times for very large-scale deployments
- Parameter sensitivity requires careful tuning for different cloud configurations and workload patterns
- The softmax decoding approach with fixed temperature may not adapt optimally to all problem instances

Modeling Limitations:

- Assumes static workloads, while real cloud environments experience dynamic and unpredictable workload variations
- Simplified network models that don't capture full data center network topology and bandwidth constraints
- Limited consideration of storage I/O constraints and heterogeneity beyond CPU and memory

Implementation Limitations:

- Simulation-based validation rather than deployment on actual cloud infrastructure
- Limited integration with real container orchestration platforms like Kubernetes
- Focus on offline optimization rather than real-time adaptive scheduling

7.4 Future Scope and Enhancements

Several directions for future research and system enhancement are identified:

Algorithm Improvements:

- Develop adaptive parameter control mechanisms that automatically tune hyperparameters based on search progress
- Incorporate machine learning techniques to predict promising search directions
- Implement parallel and distributed versions of MO-NSO for faster convergence on large-scale problems
- Explore hybrid approaches combining MO-NSO with local search heuristics for improved exploitation

Model Extensions:

- Incorporate dynamic workload patterns with time-varying resource demands
- Extend to multi-cloud and federated cloud environments with geographical constraints
- Include additional objectives such as carbon footprint, security levels, and reliability metrics
- Model network-aware placement considering data locality and communication costs

System Integration:

- Develop plugins for integration with Kubernetes and Docker Swarm for real-world deployment
- Create a web-based dashboard for visualizing Pareto fronts and supporting decision-making
- Implement incremental optimization for handling dynamic workload changes without complete re-optimization
- Add support for heterogeneous accelerators (GPUs, FPGAs) in resource allocation decisions

Advanced Applications:

- Apply MO-NSO to edge computing environments with additional latency constraints
- Extend to serverless computing platforms with function placement optimization
- Investigate transfer learning approaches for knowledge sharing across different data centers
- Explore application to scientific workflows with complex dependency structures

7.5 Final Remarks

The MO-NSO framework represents a significant step forward in multi-objective optimization for cloud data centers. By successfully balancing four conflicting objectives and providing diverse trade-off solutions, it addresses a critical need in cloud resource management. While limitations exist, the foundation established by this work provides numerous opportunities for future research and practical implementation. The bio-inspired approach of neuron-synapse optimization shows promising adaptability and effectiveness in complex optimization landscapes, suggesting its potential applicability to other multi-objective optimization problems beyond cloud resource management.

References

[1] Singh, S. and Ghosh, T., 2025. Bio-Inspired Neuron Synapse Optimization for Adaptive Learning and Smart Decision-Making. arXiv preprint arXiv:2511.00042. https://arxiv.org/abs/2511.00042

[2] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.A.M.T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation, 6(2), pp.182-197. https://doi.org/10.1109/4235.996017

[3] Jawad, M., Qureshi, M.B., Khan, M.U., Ali, S.M., Mehmood, A., Khan, B., Wang, X. and Khan, S.U., 2018. A robust optimization technique for energy cost minimization of cloud data centers. IEEE Transactions on Cloud Computing, 9(2), pp.447-460. https://doi.org/10.1109/TCC.2018.2879948

[4] Mansouri, Y., Toosi, A.N. and Buyya, R., 2017. Cost optimization for dynamic replication and migration of data in cloud data centers. IEEE Transactions on Cloud Computing, 7(3), pp.705-718. https://doi.org/10.1109/TCC.2017.2659728

[5] Altahat, Mohammad Daradkeh, Tariq Agarwal, Anjali. (2024). Optimized Encryption-Integrated Strategy for Containers Scheduling and Secure Migration in Multi-Cloud Data Centers. IEEE Access. PP. 1-1. 10.1109/ACCESS.2024.3386169. https://doi.org/10.1109/ACCESS.2024.3386169

[6] S. K. Mishra, M. A. Khan, B. Sahoo and S. K. Jena, Time efficient task allocation in cloud computing environment, 2017 2nd International Conference for Convergence in Technology (I2CT), Mumbai, India, pp. 715-720. https://doi.org/10.1109/I2CT.2017.8226222