An intelligent virtual machine allocation optimization model for cloud computing environments

Aryan Kumar (M23CSA510), Ayush Mishra (M22AIE251), Harsh Parashar (M22AIE210), Prateek Singhal (M22AIE215)

Dr. Sumit Kalra, CSE IIT-Jodhpur

Abstract—This project focuses on improving the Ant Colony Optimization (ACO)-based model for optimization of the number of virtual machines on physical machines on basis of used power and available RAM & CPU described in the paper "An intelligent virtual machine allocation optimization model for cloud computing environments" (Journal of Supercomputing, 2024). While the original research is designed for cloud computing platforms, our strategy utilizes arbitrarily designed machines on the cloud. The ACO algorithm is implemented in Python and enhanced with deep learning techniques for workload prediction to optimize resource allocation further. We assess the performance of the improved model against energy efficiency and resource utilization.
The link to the GitHub repository is https://github.com/Aryank47/intelligent-vm-allocation
The implementation, scripts,and results can be found in the GitHub repository mentioned above.

Keywords—*VCC, Cloud Computing, , Scaling*

1. Introduction

Cloud computing has changed the delivery of scalable and flexible computing resources, enabling users to access services on-demand. However, the rapid expansion of cloud services has led to significant challenges, particularly in energy consumption and efficient resource utilization within data centers. Fluctuating workloads, along with fixed virtual machine (VM) configurations usually cause suboptimal resource allocation, which can later lead to increased operational costs.

In response to these challenges, Swain et al. [3] introduced an intelligent VM allocation optimization model that leverages a feed-forward neural network optimized using a self-adaptive differential evolution algorithm. This approach help in predicting resource usage, which helps in more efficient and reliable VM placements.

We are using this paper as the foundation for our project, aiming to replicate and improve the VM allocation model it proposes. To keep costs low, we simulate a cloud-like environment instead of using real cloud services like GCP, as running hundreds of VMs on actual cloud platforms would be too expensive.

2. Project Overview and Technology Stack

The objective of this project is to replicate the virtual machine (VM) allocation optimization model proposed in the paper titled *An intelligent virtual machine allocation optimization model for energy-efficient and reliable cloud environment* [3]. The VM allocation algorithm, originally involving a feed-forward neural network and a self-adaptive differential evolution optimizer, has been implemented in Python.

Technology Stack

We used the following tools and technologies to build and test our project:

Programming Language

We used **Python** to write all the code. We used some built-in Python libraries like **Math**, **Random** etc. These helped us with basic tasks like math operations, random values, copying data, and working with data files.

Deep Learning

We used **PyTorch** to create and train a neural network that helps predict future workloads.

Data Processing and Visualization

To handle data and do calculations, we used **Pandas** and **NumPy**. For creating graphs and charts, we used **Matplotlib**.

Evaluation Metrics

We evaluated our approch using the following metrics:

- **CPU Usage**: Measures how much CPU resources are used during VM allocation and execution.
- **Number of Permanent Machines**: Tracks how many physical machines (PMs) remain active during the simulation.
- **Total Power Consumption**: Calculates the total energy used by all machines to evaluate energy efficiency.
- **Reliablity**: Overall reliability is the product of each server's reliability which will be calculated as given below.

2.0.1. Hazard Rate as a Function of Resource Utilization

$$\lambda_{ij} \;=\; \lambda_{\max} \times (\mathrm{RU})^{\beta}$$

- λ_{ij}: Hazard (failure) rate of the j-th service (or VM) running on server i.
- $\lambda_{\max}$: Maximum hazard rate (a chosen constant).
- RU: Resource utilization ratio (between 0 and 1).
- β: Sensitivity exponent indicating how strongly the failure rate depends on resource utilization.

2.0.2. Hazard Rate via Mean Time Between Failures (MTBF)

$$\lambda_i \;=\; \frac{1}{\mathrm{MTBF}_i}$$

- λ_i: Hazard rate for server i.
- MTBF_i: Mean Time Between Failures for server i.

2.0.3. Reliability of a Single Server Over Time

$$R_{S_i}(t) \;=\; \exp\Big(-\lambda_i\, t\Big)$$

- $R_{S_i}(t)$: Probability that server i operates correctly (does not fail) over the interval $[0, t]$.
- λ_i: Server i's (constant) failure rate.
- t: Time duration of interest.

2.0.4. Reliability of a Server Running Multiple VMs

Assuming each of the n VMs on the same server has the same hazard rate λ_i (i.e., a single hardware failure affects them all):

$$R_{S_i}(t) \;=\; \big(\exp(-\lambda_i\, t)\big)^{n} \;=\; \exp\Big(-n\,\lambda_i\, t\Big)$$

- n: Number of VMs (or services) running on server i.
- Note: Adjust this formula for series or parallel reliability models if different assumptions apply.

2.0.5. Reliability of the Entire System

For a *series* configuration (where all servers must remain operational for system functionality), the overall reliability is the product of each server's reliability:

$$R_{\text{system}}(t) \;=\; \prod_{i=1}^{m} R_{S_i}(t) \;=\; \exp\Big(-t \sum_{i=1}^{m} \lambda_i\Big)$$

- m: Total number of servers in the system.
- $\sum_{i=1}^{m} \lambda_i$: Sum of the individual failure rates, assuming independence.

For systems with a *parallel* configuration (or redundancy), the calculation of $R_{\text{system}}(t)$ will be different.

3. Literature Review

Researchers have explored various methods to improve how virtual machines (VMs) are allocated in cloud computing to save energy and use resources better. These strategies aim to make data centers more efficient by reducing power usage and improving system performance.

3.1. Ant Colony Optimization (ACO) for VM Allocation

Ant Colony Optimization is inspired by how ants find the shortest paths. It has been used to allocate VMs efficiently. For example, a study applied ACO to reduce energy consumption in data centers by optimizing VM placement.

3.2. Deep Learning for Predicting Workloads

Deep Neural Networks help in predicting future workloads, allowing better VM allocation. This research used neural networks to forecast resource demands, leading to improved energy efficiency and reduced costs.

3.3. Energy-Efficient VM Allocation

Some studies focused on allocating VMs in a way that saves energy. By considering factors like CPU usage and power consumption, these methods aim to reduce the number of active physical machines without affecting performance.

3.4. Combining ACO and Neural Networks

Combining ACO with deep learning techniques has shown promising results. This hybrid approach benefits from ACO's optimization capabilities and machine learning's predictive power, leading to more efficient VM allocation.

3.5. SM-VMP using Genetic and Whale Optimization Algorithms

Recent work by Saxena et al. [2] introduced a Secure and Multi-objective Virtual Machine Placement (SM-VMP) framework. This method combines NSGA-II (a genetic algorithm) and the Whale Optimization Algorithm (WOA) to improve resource utilization and minimize power consumption. The model considers multiple system constraints, such as CPU and memory capacity, idle and maximum power, reliability, and VM-to-PM mapping, making it a comprehensive and practical solution for VM placement in cloud data centers.

4. Architecture Overview

Below sections explain three key components of the architecture i.e. the Resource Estimation Unit, the Neural Network Model, and the Self-Adaptive Differential Evolution Algorithm.

4.1. Resource Estimation Unit (Data Preprocessing)

The resource estimation unit helps predict how many resources (like CPU and memory) will be needed in the future. It uses past data from borge dataset[1] (source: Kaggle) tasks to make this prediction.

Let the past workload data be:

$$\{D_1, D_2, ..., D_n\}$$

These values are normalized using the formula:

$$\hat{D}_i = \frac{D_i - D_{\min}}{D_{\max} - D_{\min}}$$

This process adjusts the values so they all fall within the same range, making them easier to handle in the next steps. This step helps improve accuracy in resource prediction and avoids assigning too few or too many physical machines.

4.2. Neural Network Model

The neural network uses the processed data to learn patterns and make decisions about required physical machines.

4.3. Simulated Machines Instead of Real Cloud

Instead of using actual cloud platforms, which can be costly, we used three types of simulated machines S1, S2, and S3. These machines allow us to test how different setups perform under controlled conditions and helps us understand how systems use resources and how reliable they are.

4.3.1. Machine Details

S1 Machine

- **Speed and Memory:** 5320 MIPS (processing speed) and 4 GB RAM. It offers average performance.
- **Power Usage:** Maximum power usage is 135 units, and idle power is 93.7 units.
- **Reliability:** With a reliability score of 0.80, this machine works reliably most of the time.

S2 Machine

- **Speed and Memory:** 12268 MIPS and 8 GB RAM, suitable for more demanding tasks.
- **Power Usage:** Uses 113 units at full load and only 42.3 units when idle, making it more energy-efficient.
- **Reliability:** Has a reliability score of 0.70, which balances performance with moderate stability.

S3 Machine

- **Speed and Memory:** 36804 MIPS and 16 GB RAM. It is the most powerful among the three.
- **Power Usage:** Consumes 222 units at maximum and 58.4 units when idle.
- **Reliability:** With a high reliability score of 0.90, it performs very stably under heavy loads.

4.3.2. Why We Used These Machines

By simulating machines instead of using actual cloud infrastructure, we avoid unpredictable changes and extra costs. This setup allows us to test performance, energy usage, and reliability more accurately. It gives us better control and helps produce meaningful and fair results during experiments.

Table 1. Compact Machine Specs and Efficiency Metrics

Mach.	MIPS	P_{max} (W)	P_{idle} (W)	MIPS/W	MIPS/(W_a)
S1	5320	135	93.7	39.4	5320 / (135−93.7) ≈ 128.8
S2	12268	113	42.3	108.5	12268 / (113−42.3) ≈ 173.6
S3	36804	222	58.4	165.7	36804 / (222−58.4) ≈ 224.9

5. Existing Results

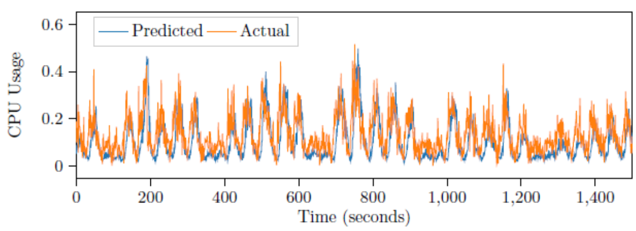

Figure 1. Reliability results as given in paper[3]

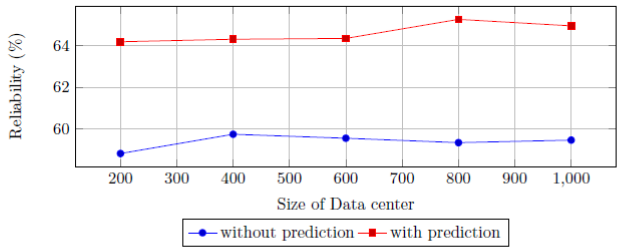

Figure 2. CPU Uses results as given in paper[3]

6. Architecture Design

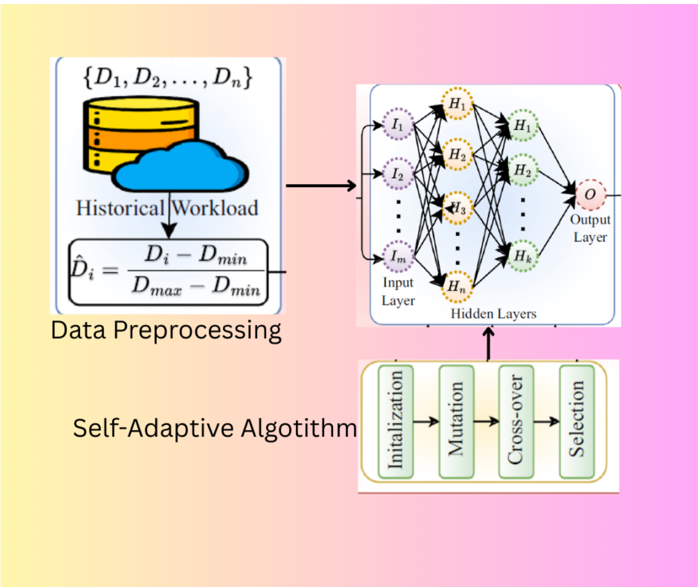

Figure 3. Architure design for Intelligent VM Allocation

7. Results

This section shows the key results from our implementation and compares them with the findings from the reference paper.

Active Physical Machines (PM):
The chart for Active PM shows how the number of running physical machines changes during the simulation. Our system turns machines on or off depending on the load, which helps save energy. This means our resource optimization works well.

Total Power Consumption:
The Total Power chart shows how much power is used by all machines.
Even during high load, our system keeps the power usage under control. This supports our method of adjusting power based on how much is being used.

CPU Usage (Predicted vs Actual):
The comparison between predicted and actual CPU usage shows that our model is accurate. The lines match closely, with only small differences during sudden changes. This proves that our prediction model works well.

Reliability Analysis:
Both our results and the paper show that reliability goes down as usage increases. This is expected, and our model follows the same pattern. It confirms that using failure rate and MTBF gives realistic reliability results.

- **Figure 5:** Shows predicted vs. actual CPU usage.

Algorithm 1 Reliable and Efficient VM Allocation

Require: ListPM, ListVM, p, q, Z, population size M
Ensure: Reliable and optimized VM allocations

/* Step 1: Random VM Allocation */
for $k = 1$ to Z **do**
  for $i = 1$ to p **do**
    for $j = 1$ to q **do**
      while ListPM is not empty **do**
        for each VM V_j in ListVM **do**
          Randomly choose PM S_i from ListPM
          for each resource $R \in \{\text{CPU, RAM, Mem}\}$ **do**
            if $V_j^R \leq S_i^R$ **then**
              Assign V_j to S_i
              $z_{ji} \leftarrow 1$
              $S_i^R \leftarrow S_i^R - V_j^R$
              Remove V_j from ListVM
            else
              $z_{ji} \leftarrow 0$
            end if
          end for
        end for
      end while
    end for
  end for
end for
Apply FFD to unallocated VMs
/* Step 2: VM Resource Prediction via Neural Network */
Initialize network with layers, weights, solutions
Evaluate initial fitness using RMSE
for each generation i **do**
  for each solution j **do**
    Generate mutation probability score mps
    if $mps \leq \alpha_1$ **then**
      Apply DE/best/1
    else if $mps \leq \alpha_1 + \alpha_2$ **then**
      Apply DE/current-to-best/1
    else if $mps \leq \alpha_1 + \alpha_2 + \alpha_3$ **then**
      Apply DE/rand/1
    else
      Apply DE/current-to-rand/1
    end if
  end for
end for
Apply uniform crossover and evaluate fitness
Perform selection
/* Step 3: Reliable Allocation using Prediction */
for each VM V_j and PM S_i **do**
  Predict resource needs of V_j using NN
  if Predicted $CPU_j \leq CPU_i$ **and** $RAM_j \leq RAM_i$ **then**
    Allocate V_j to S_i
  else
    Allocation unsuccessful
  end if
end for
Compute system reliability Re and power consumption PW

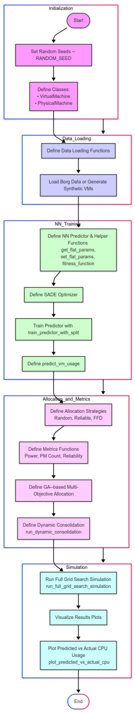

Figure 4. Code flow diagram

- **Figure 6:** Reliability increases with predicted even if when number of VMs increases but goes down in case of random after a point.
- **Figure 7:** Total power consumption rises with more PMs, emphasizing the need for efficient allocation.
- **Figure 8:** Active PMs increase with VM count, but efficient allocation helps reduce unnecessary activation as shown in case of predictive.

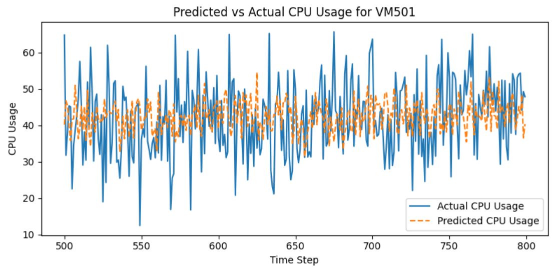

Figure 5. Predicted vs Actual CPU uses based on simulations

8. Conclusion and Comparison with Existing Work

We compared our results with the models given in the research paper. The Predicted vs Actual CPU Usage chart shows that our model works well. The predicted values are close to the actual values. Small differences appear during peak usage, which may be due to random changes in workload.

The reliability results also match the patterns shown in the paper. The paper's model uses failure rates to calculate reliability, and our results follow a similar trend. As more resources are used, the system becomes slightly less reliable, which is expected.

Overall, our simulation gives results that are similar to the paper. This means our model is working correctly. Futher, we can try to make the model even more accurate and test it under different types of workloads.

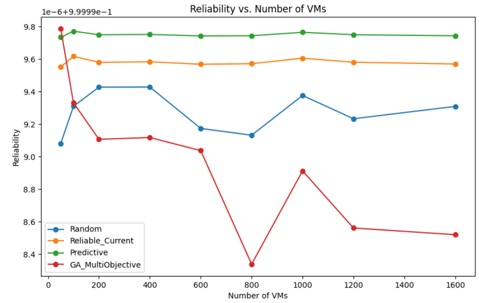

Figure 6. Reliability variations againts no. of PMs

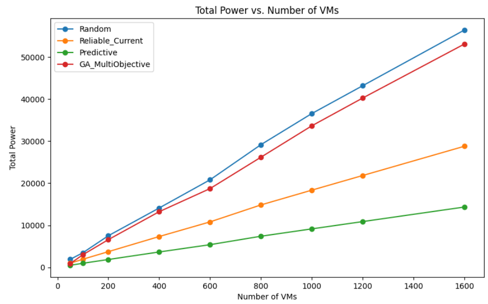

Figure 7. Total Power againts no. of PMs

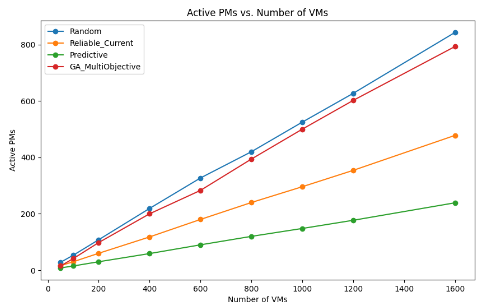

Figure 8. Active PM againts no. of VMs

9. Future Scope

To improve the model, we can add a dynamic workload classifier that automatically detects the type of workload (e.g., CPU-intensive, memory-heavy) and adjusts VM placement strategies in real-time. This can help the system respond better to different situations and improve both reliability and resource usage.

References

[1] D. Mwiti, *Borg traces dataset google clusters*, https://www.kaggle.com/datasets/derrickmwiti/google-2019-cluster-sample, 2019.

[2] A. Saxena *et al.*, "A secure and multi-objective virtual machine placement using nsga-ii and whale optimization algorithm", *Journal of Cloud Computing*, 2023.

[3] S. R. Swain, A. Parashar, A. K. Singh, and C. N. Lee, "An intelligent virtual machine allocation optimization model for energy-efficient and reliable cloud environment", *The Journal of Supercomputing*, vol. 81, no. 237, 2024. DOI: 10.1007/s11227-024-06734-1. [Online]. Available: https://link.springer.com/article/10.1007/s11227-024-06734-1.