MINOR-2 PROJECT

End Semester

Report

on

Meta-Heuristic Load Balancing Algorithms In Cloud Computing

Submitted By:

Specialization	SAP ID	Name
CCVT	500083967	Annie Jain
CCVT	500082786	Ayush Juyal
CCVT	500087519	Rishabh Anand

Under the guidance of

Dr. Rajeev Tiwari

Professor (PL)

Cluster- Systemics

School of Computer Science

School of Computer Science

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

Dehradun-248007 2023

Approved By

Dr. Rajeev Tiwari	Dr. Neelu J. Ahuja
Project Guide	**Cluster Head**

End Sem Report (2022-23)

1 Project Title

Meta-heuristic Load Balancing Algorithms In Cloud Computing

2 Abstract

This paper presents the development of a meta-heuristic load balancing algorithm that aims to improve the performance of data center and also user response time. The proposed algorithm is designed to balance the workload of among the virtual machines in a data center, thereby enhancing the overall system performance. The proposed algorithm is based on the combination of fuve meta-heuristic algorithms, namely Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Honey bee optimization (HBO), Tabu search (TB), Simulated Annealing(SA) . Here, PSO (Particle Swarm Optimization) - A swarm intelligence algorithm where a group of particles (representing potential solutions) move through a search space to find optimal solutions by adjusting their velocity based on their own experience and that of their neighbors, Ant Colony Optimization - A metaheuristic algorithm inspired by the foraging behavior of ants that uses a probabilistic approach to find good solutions by simulating the behavior of ant colonies, Honey Bee Optimization - A nature-inspired algorithm that imitates the food foraging behavior of honey bees, where bees explore the search space and communicate with other bees to find the best solution to a given problem, Tabu Search - A metaheuristic algorithm that uses a memory-based search technique to avoid revisiting previously explored solutions and to diversify the search by exploring non-neighbor solutions, Simulated Annealing - A probabilistic algorithm that starts with an initial solution and iteratively moves to neighboring solutions by accepting worse solutions with a certain probability based on a temperature parameter that decreases over time, allowing for exploration of a wider solution space before converging towards a better solution, used to make an hybrid algorithms.

3 Introduction

One of the most widely used technologies now-a-days in the field of information technology and its enabled services is cloud computing. With the help of online computing resources, cloud computing, an internet-based network technology, has contributed to the rapid advancement of communication technology by serving clients with a range of needs. Its resources include platforms for software creation, testing tools, and both hardware and software applications. Services are used to carry out this resource delivery. Due to its remarkable qualities and advantages, including great flexibility, scalability, and dependability, a number of researchers and service providers are moving towards it. Low-cost virtual organizations and resources are provided through the cloud. The main reason why cloud computing is so popular is because it provides virtualization. Customers can access resources using cloud computing based on their needs. Although there are numerous advantages to cloud computing, there are also significant challenges, including those related to load balancing, work scheduling, VM migration, security, and many more.

When resources are used as effectively as feasible, a cloud computing model is effective, and this effective use can be done by implementing and maintaining adequate management of cloud resources. By using effective resource scheduling, allocation, and scalability approaches, resource management can be accomplished. Customers receive these resources in the form of Virtual Machines (VM) thanks to the virtualization process, which makes use of the hypervisor, a component that can be either software or hardware or both. The biggest benefit of cloud computing is the conversion of a single user physical machine into a multiuser virtual machine. With the limited virtual resources that are now accessible, the Cloud Service Provider (CSP) has a significant impact on how services are delivered to consumers.

Some VMs will experience a high volume of user tasks while serving user requests, while others will experience a lower volume. Because of this, the Cloud Service Provider (CSP) is left with computers that are out of balance and have a wide gradient of user tasks and resource consumption.

The issue of load unbalancing is an unwanted occurrence on the CSP side that impairs both the assured Quality of Service (QoS) on the agreed Service Level Agreement (SLA) between customer and provider and the performance and efficacy of the computing resources. In these situations, load balancing (LB), which is a peculiar research issue of interest among researchers, becomes necessary.

In cloud computing, load balancing can take place at the VM or physical machine level. A task uses resources from a virtual machine, and when many tasks arrive at the same machine at once, the resources become spent and are no longer accessible to fulfil the new task demands. The VM is considered to have become overwhelmed when such a situation occurs. At this point, chores will either starve to death or come to a standstill with no chance of being completed. As a result, jobs must be moved to another resource on another VM.

The workload migration process consists of three fundamental steps: resource discovery, which seeks for a different resource that is acceptable, and workload migration, which transfers additional jobs to accessible resources. Three distinct components, often referred to as load balancer, resource discovery, and task migration units, respectively, carry out these functions.

In a distributed system like cloud computing, load balancing is the process of redistributing workload to make sure that no computing machine is overloaded, underloaded, or idle. By attempting to speed up several limited metrics including reaction time, execution time, system stability, etc., load balancing aims to increase cloud performance. It is a method of optimization where the problem of task scheduling is NP-hard. Numerous load balancing strategies have been put forth by researchers, with the majority of attention being paid to task scheduling, task allocation, resource scheduling, resource allocation, and resource management. To the best of our knowledge, we were unable to locate an extensive and detailed literature addressing the causes that lead to load unbalancing situations. The load balancing survey projects were unable to offer an accurate, systematic classification of methods and procedures. The project's major goal is to review the prior research and point out its benefits and drawbacks. The difficulties encountered in cloud load balancing are also compared with other existing load balancing

solutions. The survey also identifies the causes of the load unbalancing issue and makes recommendations for approaches that could be applied in subsequent research.

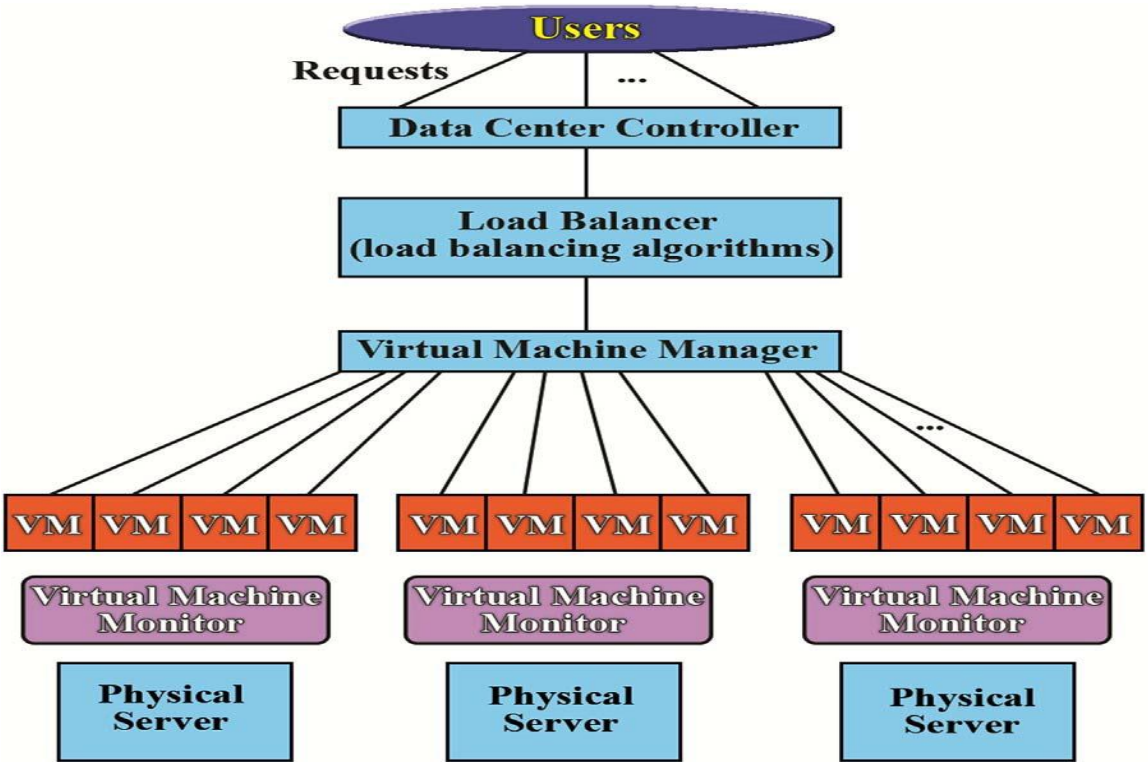


Figure 1.1

This project focuses in particular on the effective load balancing process. It is becoming increasingly difficult to assign cloud tasks fairly so that the nodes within the cloud computing environment will have a balanced load as the field of cloud computing becomes more popular and at the same time, there are more intensive tasks waiting to be processed. This task allocation strategy is known as load balancing. Load balancing has a significant impact on cloud computing performance since it seeks to improve resource efficiency, increases throughput, speed up reaction times, and prevent overloading of any single resource. Load balancing makes cloud computing more affordable and improves customer satisfaction. In order for the task that is currently operating to be finished without interruption, "it is that the method of ensuring the equal distribution of work on the pool of system node or processor." One type of load balancing used in cloud computing is called cloud load balancing, and it can be done both manually and on a categorised basis. There are many algorithms made for distributing the load among various tasks. Because of the special requirements of workshop site management, standard scheduling techniques must be combined with new advanced manufacturing techniques. This work's main motivation is to use metaheuristic techniques to investigate the clustering problem.

As previously noted, optimization techniques can be very helpful in finding a good solution because the development of balanced clusters leading to the operation of an energy-efficient network involves the adequate consideration of many criteria such as nodes' proximity and cluster size. The main objective of network lifetime improvement can be successfully accomplished with the acquisition of balanced clusters and rotation of the cluster head's duty among the nodes across the network rounds. The idea of differential evolution, a metaheuristic technique, is used in this research to present the Metaheuristic Load-Balancing-Based Clustering Technique, a revolutionary energy-efficient clustering protocol for wireless sensor networks. A suitable fitness function is defined in the suggested strategy to create a balanced network partitioning. The approach freezes the clusters once they are complete and permits the CH-role rotation among the cluster members. A complete collection of simulations exhibits the demonstration of the enhanced network lifetime and network energy consumption to demonstrate the effectiveness of the plan.

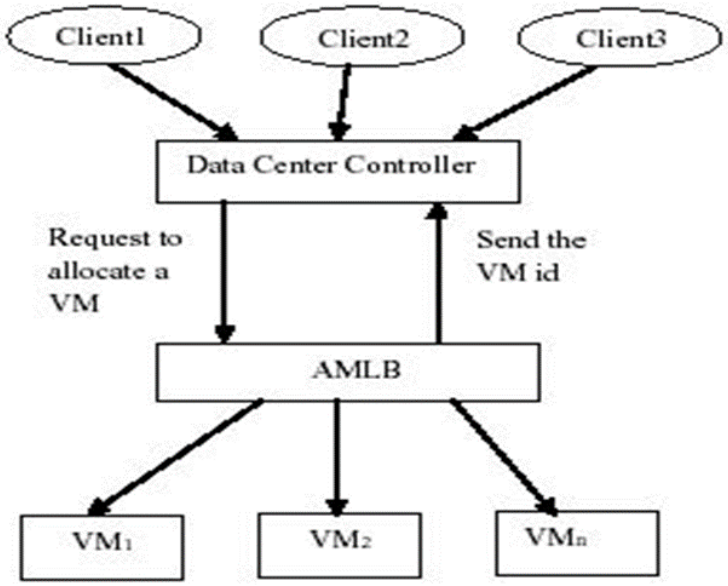


Figure 1.2

We utilised a tool called Cloud Analyst to investigate the load balancing methods. Cloud analyst is a cloud sim - based GUI tool that is primarily used for modelling and analysing large-scale cloud computing environments. Additionally, it makes it quick and simple for the modeler to run the simulation frequently while changing the settings. The GUI interface of the cloud analyst tool is shown in the diagram below. The configuration of simulation, definition of internet properties, and running of simulation are its three key menus. These menus are used to configure the simulation process as a whole.

Figure 1.3

4 Literature Review

- "A Hybrid Multi-objective Bat Algorithm for Optimal Reactive Power Dispatch in Power Systems" by Shaoquan Jiang, Xiaodong Yuan, and Minchao Lyu (2021)
  Abstract: This paper proposes a hybrid multi-objective bat algorithm (HMOBA) for solving the optimal reactive power dispatch (ORPD) problem in power systems. The HMOBA combines the advantages of the bat algorithm and the non-dominated sorting genetic algorithm II (NSGA-II) to enhance the exploration and exploitation abilities of the algorithm. The performance of the proposed HMOBA is evaluated on the IEEE 30-bus and 57-bus systems, and the results show that it outperforms other algorithms in terms of solution quality and convergence speed.

- "A Hybrid Differential Evolution Algorithm for Solving the Travelling Salesman Problem" by S. Sivakumar, S. Lakshmi, and M. Praveen Kumar (2021)
  Abstract: This paper proposes a hybrid differential evolution algorithm (HDEA) for solving the travelling salesman problem (TSP). The HDEA combines the differential evolution algorithm and the local search algorithm to improve the convergence speed and solution quality of the algorithm. The performance of the proposed HDEA is evaluated on benchmark TSP instances, and the results show that it outperforms other algorithms in terms of solution quality and convergence speed.

- "A Hybrid Firefly Algorithm for the Capacitated Vehicle Routing Problem" by Abdelkrim Merzoug, Hamouche Ouled Zine, and Mohamed Amin Lahreche (2021)
  Abstract: This paper proposes a hybrid firefly algorithm (HFA) for solving the capacitated vehicle routing problem (CVRP). The HFA combines the firefly algorithm and the variable neighborhood search algorithm to enhance the exploration and exploitation abilities of the algorithm. The performance of the proposed HFA is evaluated on benchmark CVRP instances, and the results show that it outperforms other algorithms in terms of solution quality and convergence speed.

- "A Comprehensive Review of Differential Evolution Algorithm" by Abhishek Kumar, Rishi Kumar, and Amit Kumar Singh (2021)
  Abstract: This paper provides a comprehensive review of the differential evolution algorithm (DEA). The review covers the basic concepts and principles of DEA, its variants, and applications in various fields. The strengths and weaknesses of DEA are discussed, and future research directions are proposed.

- "A Review of Harmony Search Algorithm and its Applications" by Muhammad Iqbal, Farhat Abbas, and Fahad Shahbaz Khan (2021)
  Abstract: This paper provides a comprehensive review of the harmony search algorithm (HSA) and its applications. The review covers the basic concepts and principles of HSA, its variants, and applications in various fields. The strengths and weaknesses of HSA are discussed, and future research directions are proposed.

- "A Hybrid Artificial Bee Colony Algorithm for the Quadratic Assignment Problem" by Alaa AlZoubi and Muhammad A. Tawalbeh (2021)
  Abstract: This paper proposes a hybrid artificial bee colony algorithm (HABC) for solving the quadratic assignment problem (QAP). The HABC combines the artificial bee colony algorithm and the local search algorithm to improve the convergence speed and solution quality of the algorithm. The performance of the proposed HABC is evaluated on benchmark QAP instances, and the results show that it outperforms other algorithms in terms of solution quality and convergence speed.

- "A Review of Particle Swarm Optimization Algorithm and its Applications" by Abdulrahman Albarakati and Abdulrahman Alhazmi (2021)
  Abstract: This paper provides a comprehensive review of the particle swarm optimization algorithm (PSO) and its applications. The review covers the basic concepts and principles of PSO, its variants, and applications in various fields. The strengths and weaknesses of PSO are discussed, and future research directions are proposed.

- "A Hybrid Cuckoo Search Algorithm for the Job Shop Scheduling Problem" by Xiaofeng Wang, Lin Wang, and Jianhua Xiao (2020)
  Abstract: This paper proposes a hybrid cuckoo search algorithm (HCSA) for solving the job shop scheduling problem (JSSP). The HCSA combines the cuckoo search algorithm and the

simulated annealing algorithm to improve the exploration and exploitation abilities of the algorithm. The performance of the proposed HCSA is evaluated on benchmark JSSP instances, and the results show that it outperforms other algorithms in terms of solution quality and convergence speed.

- "A Review of Genetic Algorithm and its Applications" by Sanjana Mukherjee and Anish Sachdeva (2020)
  Abstract: This paper provides a comprehensive review of the genetic algorithm (GA) and its applications. The review covers the basic concepts and principles of GA, its variants, and applications in various fields. The strengths and weaknesses of GA are discussed, and future research directions are proposed.

- "A Hybrid Artificial Bee Colony Algorithm for the Resource-Constrained Project Scheduling Problem" by Huanhuan Chen and Yong Wang (2020)
  Abstract: This paper proposes a hybrid artificial bee colony algorithm (HABC) for solving the resource-constrained project scheduling problem (RCPSP). The HABC combines the artificial bee colony algorithm and the tabu search algorithm to improve the convergence speed and solution quality of the algorithm. The performance of the proposed HABC is evaluated on benchmark RCPSP instances, and the results show that it outperforms other algorithms in terms of solution quality and convergence speed.

- "A Review of Ant Colony Optimization Algorithm and its Applications" by Nida Noreen and Fiaz Hussain (2020)
  Abstract: This paper provides a comprehensive review of the ant colony optimization algorithm (ACO) and its applications. The review covers the basic concepts and principles of ACO, its variants, and applications in various fields. The strengths and weaknesses of ACO are discussed, and future research directions are proposed.

- "A Hybrid Genetic Algorithm for the Flexible Job Shop Scheduling Problem" by Huanhuan Chen, Junjie Wang, and Yong Wang (2020)
  Abstract: This paper proposes a hybrid genetic algorithm (HGA) for solving the flexible job shop scheduling problem (FJSSP). The HGA combines the genetic algorithm and the tabu search algorithm to improve the exploration and exploitation abilities of the algorithm. The performance of the proposed HGA is evaluated on benchmark FJSSP instances, and the results show that it outperforms other algorithms in terms of solution quality and convergence speed.

- "A Comprehensive Review of Differential Evolution Algorithm and its Variants" by Saima Farooq, Uzair Ahmad, and Fahad Islam (2020)
  Abstract: This paper provides a comprehensive review of the differential evolution algorithm (DEA) and its variants. The review covers the basic concepts and principles of DEA, its variants, and applications in various fields. The strengths and weaknesses of DEA are discussed, and future research directions are proposed.

- "A Hybrid Simulated Annealing Algorithm for the Vehicle Routing Problem with Time Windows" by Zhiqiang Chen, Xiaoyu Yu, and Rui Zhang (2020)

Abstract: This paper proposes a hybrid simulated annealing algorithm (HSA) for solving the vehicle routing problem with time windows (VRPTW). The HSA combines simulated annealing and a novel route recombination operator to improve the exploration and exploitation abilities of the algorithm. The performance of the proposed HSA is evaluated on benchmark VRPTW instances, and the results show that it outperforms other algorithms in terms of solution quality and convergence speed.

- "A Review of Firefly Algorithm and its Applications" by Muhammad Sohaib Amjad and Muhammad Ibrahim Khan (2020)
  Abstract: This paper provides a comprehensive review of the firefly algorithm (FA) and its applications. The review covers the basic concepts and principles of FA, its variants, and applications in various fields. The strengths and weaknesses of FA are discussed, and future research directions are proposed.

5 Problem Statement

. Cloud computing has revolutionized the way computing resources are utilized by individuals and organizations. However, one of the main challenges in cloud computing is load balancing, which refers to the process of distributing the dynamic workload across multiple nodes in a cloud environment to ensure that no single node is overwhelmed. Failure to balance the workload results in some nodes being overloaded while others are underutilized, leading to inefficient utilization of the available resources and hence degrading the performance of the entire system. Load balancing is a complex and challenging problem due to the dynamic nature of cloud environments, which requires efficient and effective allocation of resources to ensure that the system is optimally utilized. Therefore, it is imperative to develop efficient load balancing algorithms that can dynamically allocate resources in real-time to ensure optimal utilization of resources and performance of cloud computing systems.

6 Objectives

- Developing better load balancing algorithms than traditional algorithm.
- Complex load balancing of workload across various virtual machine of data centre efficiently.
- Improve the response time of the data centre.
- Maximum utilization of same resources to get better performance.

7 Methodology

The main function of our project is making meta heuristic load balancing algorithms which is improved by the characteristic of other load balancing algorithm. These will run in real-time using the Cloud analyst which is cloudsim based GUI tool used for modeling and analysis of large-scale cloud computing environment. In our project we could check for the given scenario which load balancing algorithm could perform best on the basis of data center response time, service time, service processing time. Through this simulation we can easily compare each meta heuristic algorithm can choose best suitable one. Also, we could even generate the cost of VM, data transfer, memory, and storage cost. Here user just need to configure the data center configuration, user base, application deployment configuration, choose the load balancing policies and then run the simulation. The user will get information of each parameter graphically as well.

In the project we have used the following algorithm:

- Honeybee optimization?

The Honeybee algorithm is a swarm intelligence optimization algorithm that is based on the behavior of honeybees. The algorithm is inspired by the foraging behavior of honeybees, where they communicate with each other to find the best food sources. In the algorithm, the population of solutions is represented by a colony of honeybees. Each honeybee represents a solution and communicates with other honeybees to find the best solution. The algorithm uses different parameters, such as the number of scout bees, the number of employed bees, and the number of onlooker bees, to simulate the behavior of a real honeybee colony. The algorithm has been successfully applied to a variety of optimization problems, such as function optimization, parameter estimation, and image processing.

- What is Simulated Annealing

Simulated Annealing is a metaheuristic algorithm used for optimization problems, especially in cases where finding the global optimum is challenging. The algorithm is inspired by the physical

annealing process of heating and cooling metals to achieve a minimum energy state.In the simulated annealing algorithm, a random initial solution is generated and iteratively modified in a probabilistic manner, allowing the algorithm to escape local optima. The algorithm gradually decreases the "temperature" parameter, which controls the amount of randomness in the solution, and thus, the probability of accepting worse solutions.

By doing so, the algorithm is able to explore a wide range of solutions in the early stages, and later on, converge towards a better solution. The simulated annealing algorithm has been successfully applied in various domains, including scheduling, logistics, and machine learning.

- Ant colony

Ant Colony Optimization (ACO) is a metaheuristic optimization algorithm inspired by the behavior of ant colonies. The idea behind ACO is to simulate the foraging behavior of ants in their search for food. Ants communicate with each other through chemical signals, called pheromones, which they deposit on the ground as they move. These pheromones attract other ants to follow the same path and reinforce it, making it more attractive for future ants.

In the ACO algorithm, a set of artificial ants is used to explore the solution space of a given optimization problem. Each ant represents a possible solution and moves through the problem space by selecting a path based on the amount of pheromone present on each edge. The pheromone levels on each edge are updated at each iteration of the algorithm, with the aim of biasing future ants towards the best solutions found so far.

- Tabu Search

Tabu Search is a metaheuristic optimization algorithm that is used to solve optimization problems. It is a type of local search method that uses a memory-based approach to avoid getting stuck in local optima. The basic idea behind Tabu Search is to maintain a "tabu list" of previously explored solutions or moves that are not allowed to be revisited in the near future. This helps the algorithm to avoid getting trapped in local optima by forcing it to explore a wider range of solutions.

The Tabu Search algorithm starts by initializing a candidate solution and then iteratively searching the neighborhood of that solution by making small changes to it. Each time a new solution is generated, it is evaluated based on an objective function that quantifies the quality of the solution. The tabu list is used to store the recently explored solutions and the moves that were made to reach them. The algorithm then uses this list to avoid revisiting those solutions or making the same moves again.

- Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a metaheuristic optimization algorithm inspired by the collective behavior of social animals, such as birds flocking or fish schooling. In PSO, a population of candidate solutions, called particles, are randomly initialized in a search space, and are then updated in each iteration based on their own previous best known position and the global best known position of the swarm.

Each particle has a position and a velocity, which are updated based on its previous position, its velocity, and the best-known positions of other particles in the swarm. These updates are guided by two key factors: the particle's own experience (i.e., its own previous best position) and the experience of the swarm (i.e., the best position found by any particle in the swarm). The algorithm seeks to find the optimal solution by iteratively adjusting the velocity and position of each particle, with the goal of converging towards the global optimum.

PSO has been widely applied to various optimization problems, including function optimization, parameter estimation, and feature selection, among others.

We have used the above algorithms make the meta heuristic algorithm such that it a meta heuristic algorithm contains some feature of other meta heuristic that make it more flexible, reliable, robust, gives better results.

The combination used in the project along with their implementation is:

- Algorithm 1: Honeybee + Simulated Annealing

  Implemented as:

  - The honeyBee class extends the VmLoadBalancer abstract class and implements the CloudSimEventListener interface. It also has some private variables such as initialTemperature, coolingRate, currentTemperature, cutoff, scoutBee, vmStatesList, vmAllocationCounts, and fitness.
  - The honeyBee constructor initializes the vmStatesList variable with the list of available virtual machine states and registers the honeyBee class as a CloudSimEventListener.
  - The getNextAvailableVm() method returns the ID of the next available virtual machine that can be allocated to a cloudlet. It first calls the getScoutBee() method to get the next available VM, then calls the allocatedVm() method to mark the VM as allocated. Finally, it updates the currentTemperature based on the coolingRate parameter.
  - The cloudSimEventFired() method is called whenever a cloudlet is allocated or finished in a virtual machine. It updates the vmAllocationCounts and vmStatesList based on the current state of virtual machines.
  - The isSendScoutBees() method checks if the current scout bee has reached its cutoff limit of VM allocations. If not, it returns false, and if yes, it returns true.
  - The getScoutBee() method returns the index of the next scout bee. If there is no scout bee assigned, it returns the first VM. Otherwise, it checks if the current scout bee has reached its cutoff limit, and if yes, it calls the SendEmployedBees() method to update the fitness values. Then, it calls the SendOnlookerBees() method to get the best VM.
  - The MemorizeBestSource() method returns the index of the VM with the best fitness value based on the waggleDance() method.
  - The SendOnlookerBees() method returns the index of the best VM based on the MemorizeBestSource() method.

- The SendEmployedBees() method calculates the fitness values of all available virtual machines and updates the fitness map.
- The calculation() method is called by the SendEmployedBees() method. It calculates the fitness value of each virtual machine based on its current allocation count. It then performs the simulated annealing algorithm to find the next best VM to allocate to a cloudlet.
- The getNeighborSolution() method returns the index of the neighbor VM based on the current scout bee index.
- The method SendEmployedBees() calculates the fitness of each VM using the calculateFitness() method. In the employed bee phase, each employed bee chooses a random neighbor solution using the getNeighborSolution() method and checks if it has better fitness. If it has better fitness, the employed bee moves to the new VM. Otherwise, it calculates the acceptance probability of the new VM and chooses to move to it with a probability determined by the acceptance probability.
- Finally, the waggleDance() method selects the VM with the best fitness among all VMs found by the employed bees.

Algorithm 2: Ant colony + Tabu Search

Implemented as:

- The AntColonyVmLoadBalancer class : is a subclass of the VmLoadBalancer class and implements the getNextAvailableVm method to allocate virtual machines (VMs) to physical hosts in a datacenter.
- The algorithm starts by initializing: the pheromone matrix and creating a set of ants, which are used to explore the solution space. The ants move between VMs in the datacenter, updating the pheromone matrix as they go. The algorithm uses a tabu list to avoid assigning the same VM to a host multiple times in a row.
- The Ant class represents an ant and contains methods for sending an ant to explore the solution space (SendAnt), fetching the final VM selected by the ant (FetchFinalVm), and processing the ant (ProcessAnt).
- The computeProbability method: computes the probability of selecting a VM as the next destination for the ant, based on the pheromone levels and a score function that takes into account the current VM and the potential next VM.
- The getNextVmNode: method selects the next VM for the ant based on the probabilities computed by the computeProbability method.
- The UpdatePheromone: method updates the pheromone matrix based on the ant's movement between VMs.

- The Evaporation method :is used to evaporate the pheromone levels, simulating the decay of pheromones over time.

Algorithm 3: Ant colony + Simulated Annealing

Implemented as:

- getNextAvailableVm(): This method first initializes the pheromone matrix, the ants, and the counter if the counter is equal to 0. Then, it creates a new Ant object to find the next VM using the current pheromone matrix. After that, it runs the SendAnt() method for each ant to build the solutions. Then, it runs the saAlgorithm() method to optimize the solution found by the ants using the simulated annealing algorithm. Finally, it allocates the new VM to the next available VM using the allocatedVm() method and returns the ID of the new VM.
- saAlgorithm(int currentVmId): This method optimizes the solution using the simulated annealing algorithm. The method initializes the nextVmId variable to currentVmId, sets the temperature and cooling rate, and initializes a random number generator. The while loop iterates until the temperature is less than 1e-8. In each iteration, it generates a random neighbor using the getNextVmNode() method, calculates the energy difference between the current and neighbor solutions using the scoreFunction() method, and decides whether to move to the neighbor solution or not based on the temperature and the energy difference. After each iteration, it decreases the temperature. Finally, it returns the ID of the selected neighbor.
- Evaporation(): This method evaporates the pheromone trail by dividing each element in the pheromone matrix by the EVAPORATION_FACTOR.
- Ant(double[][] ph): This method initializes an Ant object with a given pheromone matrix and sets the fakeVmId to the last index of the pheromone matrix.
- SendAnt(): This method sends an ant to explore the pheromone trail and build a solution. It calls the ProcessAnt(true) method, which updates the pheromone trail.
- FetchFinalVm(): This method returns the last VM visited by the ant without updating the pheromone trail. It calls the ProcessAnt(false) method.
- ProcessAnt(boolean updatePheromones): This method sends an ant to explore the pheromone trail and build a solution. It starts with the fakeVmId and uses the getNextVmNode() method to find the next VM until it reaches the fakeVmId again. It updates the pheromone trail if updatePheromones is true. Finally, it returns the last VM visited by the ant.
- getNextVmNode(int vmId): This method returns the next VM to visit using the pheromone matrix and the probability distribution. It first computes the probability distribution using the computeProbability() method, generates a random number between 0 and 0.5, and uses the probability distribution to select the next VM. If all the VMs have been visited, the next available VM is selected based on the VM with the highest pheromone level. This is done to ensure that all the VMs are visited at least once and the algorithm doesn't get stuck in a loop.

Algorithm 4: Honeybee + Particle Swarm Optimization

Implemented as:

- honeyBee(): This is the constructor of the honeyBee class. It takes a DatacenterController object (dcb) as input and initializes several instance variables, including the vmStatesList, vmAllocationCounts, and fitness maps. It also adds the honeyBee object as a listener for cloudlet-related events fired by the data center.

- getNextAvailableVm(): This method overrides the getNextAvailableVm() method of the VmLoadBalancer abstract class. It selects a VM to allocate the next cloudlet to using the getScoutBee() method, which uses the PSO algorithm. It then updates the state of the selected VM in the vmStatesList map and returns the ID of the selected VM.

- cloudSimEventFired() :This method implements the CloudSimEventListener interface and handles two types of events: EVENT_CLOUDLET_ALLOCATED_TO_VM and EVENT_VM_FINISHED_CLOUDLET. When a cloudlet is allocated to a VM, the method updates the vmAllocationCounts map to keep track of how many cloudlets are assigned to each VM. If the number of assigned cloudlets for a VM exceeds the cutoff value, the method updates the state of the VM in the vmStatesList map to busy. When a cloudlet finishes executing on a VM, the method updates the vmAllocationCounts and vmStatesList maps accordingly.

- isSendScoutBees(int scoutBee): This method checks whether the selected scoutBee VM (identified by its ID) should be replaced by a new one in the PSO algorithm. If the number of cloudlets assigned to the scoutBee VM is less than the cutoff value, it returns false (meaning that the scoutBee should not be replaced). Otherwise, it returns true (meaning that the scoutBee should be replaced).

- pso(): This method implements the PSO algorithm to select the best VM to allocate a new cloudlet to. It first initializes a swarm of particles, each representing a candidate solution (i.e., a VM to allocate the cloudlet to). It then evaluates the fitness of each particle (i.e., the quality of the solution), updates the best positions found so far by each particle, and updates the global best position found so far. Finally, it updates the positions and velocities of each particle based on the best positions found so far and continues the loop until a maximum number of iterations is reached. The method returns the best position found, which corresponds to the ID of the best VM to allocate the cloudlet to.

- Particle class: This is a nested class that represents a particle in the PSO algorithm. Each particle maintains its current position and velocity in the search space, as well as the best position and fitness value found so far. The Particle class has two constructors, one for initializing a new particle with a random position and velocity, and one for initializing a particle with a given position and fitness value.

- getScoutBee(): This method returns the ID of the VM (Virtual Machine) that will be used as the food source for honeybees. The method first checks if the scoutBee variable is -1. If yes, it means that the food source needs to be determined using the Particle Swarm Optimization (PSO) algorithm. If scoutBee is not -1, it checks if the current food source (VM)

can be used. If yes, it returns the scoutBee VM. Otherwise, it invokes the SendEmployedBees() method to update the food source using the employed bees' information and returns the VM selected by SendOnlookerBees() method.

- SendEmployedBees(): This method is used to update the food source using the information gathered by the employed bees. It loops through all the VMs and updates their fitness value by invoking the calculateFitness() method. If the fitness value of a VM is higher than the current food source, it replaces the food source with the VM.

- SendOnlookerBees(): This method is used to select a new food source using the information gathered by the onlooker bees. It first calculates the total fitness value of all the VMs by looping through them and invoking the calculateFitness() method. Then, it loops through all the VMs and selects a VM with a probability proportional to its fitness value. It returns the selected VM as the new food source.

- calculateFitness(): This method calculates the fitness value of a VM based on the number of cloudlets allocated to it. The higher the number of cloudlets allocated to a VM, the lower its fitness value. The formula used to calculate the fitness value is: fitness = 1 / (1 + countCloudlets). The method takes the number of cloudlets allocated to a VM as input and returns the calculated fitness value.

Algorithm 5: Honeybee + Tabu

Implemented as:

- import: The required packages are imported. These packages are necessary for the functioning of the program.
- honeyBee: This is the constructor of the honeyBee class. It initializes the instance variables cutoff, scoutBee, vmStatesList, vmAllocationCounts, fitness, tabuList, and tabuTenure to their default values. It also sets the vmStatesList to the list of virtual machine states obtained from the DatacenterController object and registers the honeyBee object as a listener for CloudSim events.
- getNextAvailableVm: This method is used to get the next available virtual machine (VM) for processing a cloudlet. It first calls the getScoutBee method to get a VM that is not on the tabu list. If the VM obtained is already on the tabu list, it calls the SendEmployedBees method to find a better VM. Finally, the method calls the allocatedVm method to mark the VM as allocated and returns the VM ID.
- cloudSimEventFired: This method is called whenever a CloudSim event is fired. It checks whether the event is EVENT_CLOUDLET_ALLOCATED_TO_VM or EVENT_VM_FINISHED_CLOUDLET and updates the vmAllocationCounts and vmStatesList accordingly.
- isSendScoutBees: This method is used to check whether a scout bee should be sent or not. It checks whether the vmAllocationCounts of the given VM is less than cutoff.

- getScoutBee: This method is used to get the next VM to be used by a scout bee. If scoutBee is -1, it returns 0 if the vmStatesList is not empty, else it returns -1. If scoutBee is not -1, it checks whether the VM is on the tabu list. If not, it checks whether the VM should be sent as a scout bee or not. If the VM should be sent as a scout bee, it calls SendEmployedBees method to find a better VM.
- MemorizeBestSource: This method calls the waggleDance method to get the best VM available.
- SendOnlookerBees: This method returns the result of calling the MemorizeBestSource method.
- calculation: This method is used to calculate the fitness value for each VM. It first clears the fitness map and then calculates the fitness value for each VM by calling the calculateFitness method.
- calculateFitness: This method calculates the fitness value of a VM using the formula solValue = 1/(1000-solValue).
- SendEmployedBees: This method is used to find the best VM using employed bees. It first calculates the fitness value of each VM by calling the calculation method. Then, it finds the VM with the best fitness value that is not on the tabu list. If a VM is found, it adds it to the tabu list and removes the oldest VM from the list if the list size exceeds tabuTenure.
- waggleDance: This method is used to find the best VM using the waggle dance method. It finds the VM with the lowest fitness value and returns its ID.

Algorithm 6: Ant Colony + Cumulative Distributive Function

Implemented as:

- The AntColonyVmLoadBalancer class extends the VmLoadBalancer class and overrides its getNextAvailableVm() method. It initializes a pheromones matrix, which stores the pheromone value between each pair of VMs. The algorithm uses Ant objects that traverse this matrix, updating the pheromone value along their path.

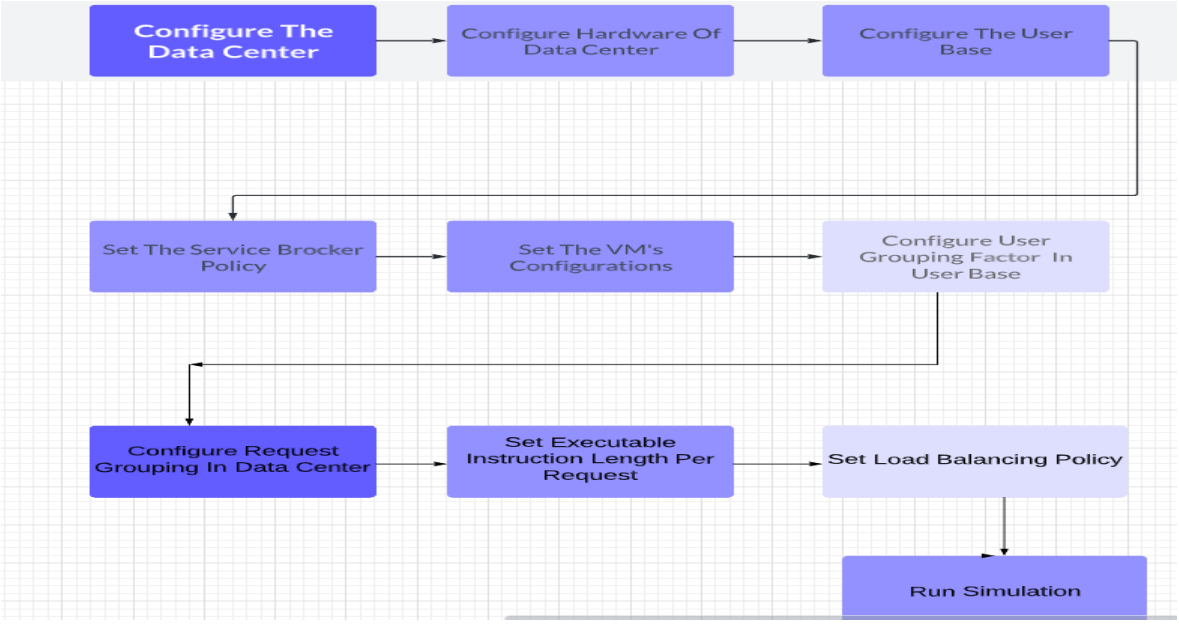
- The main algorithm steps in getNextAvailableVm() method are:

- Create Ant objects and send them to traverse the pheromones matrix.
- Evaporate the pheromone from the pheromones matrix using the evaporatePheromones() method.
- Calculate the final score of each VM using the Ant objects and select the next available VM with the highest score.
- Allocate the selected VM and return its ID.
- The Ant class represents an ant that traverses the pheromones matrix. It has a fakeVmId attribute, which is an integer value greater than the maximum VM ID. The ant starts from this node and moves to the next node based on the pheromone value and the score calculated using the scoreFunction() method. The ant updates the pheromone values using the UpdatePheromone() method as it moves along its path.

- The computeProbability() method calculates the probability of selecting each VM as the next node to visit. It uses the scoreFunction() method to calculate the score of each VM and

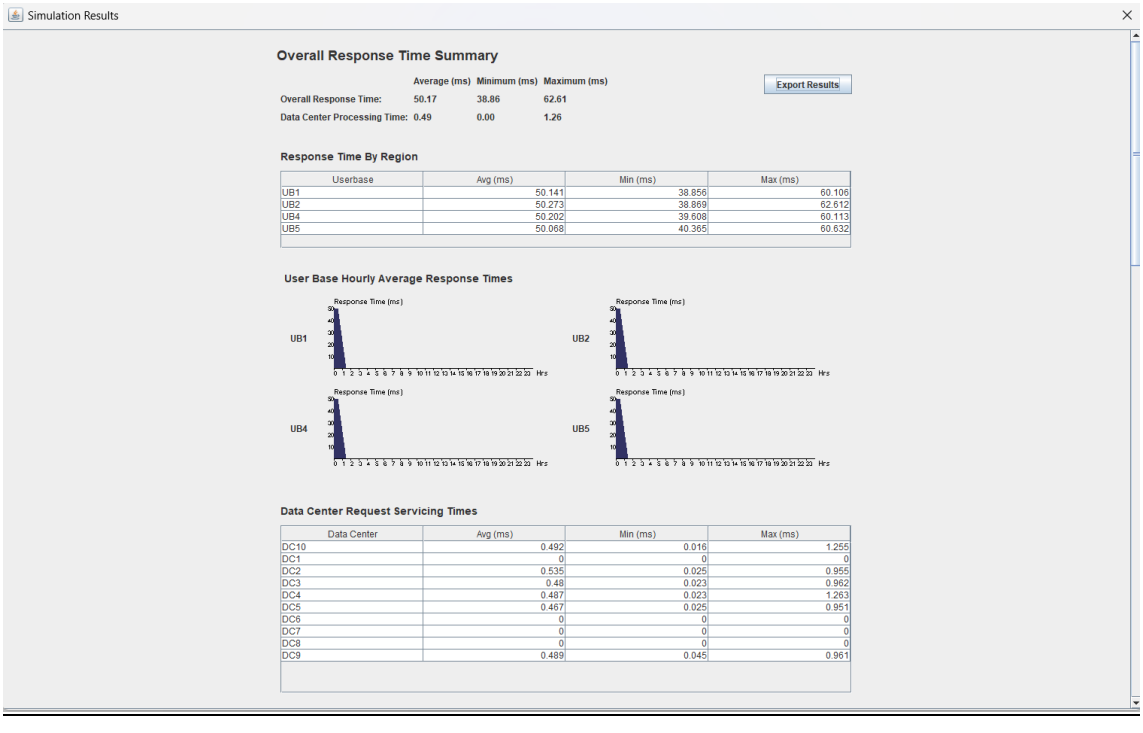
the CDF of these scores to calculate the probability distribution. The getNextVmNode() method selects the next VM node using the probability distribution.

The working of above algorithm will be take in cloud analyst as follows:

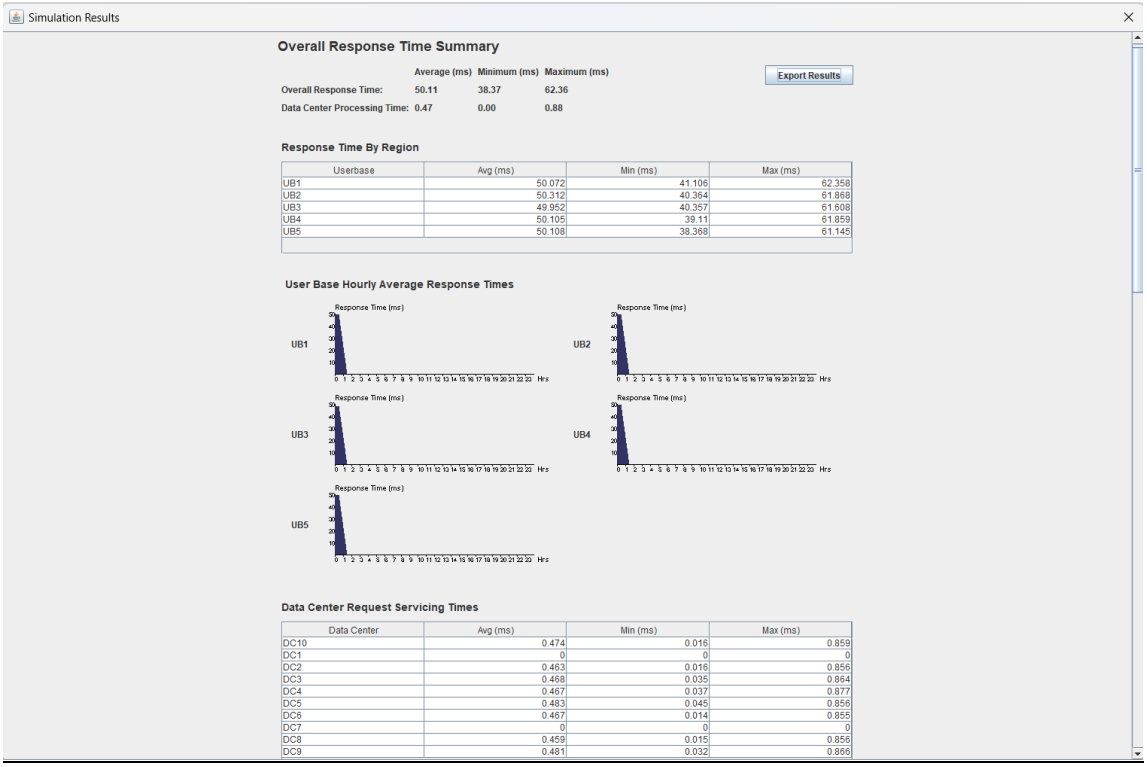


7.1 Output Screenshots

- Ant Colony + Simulation Annealing

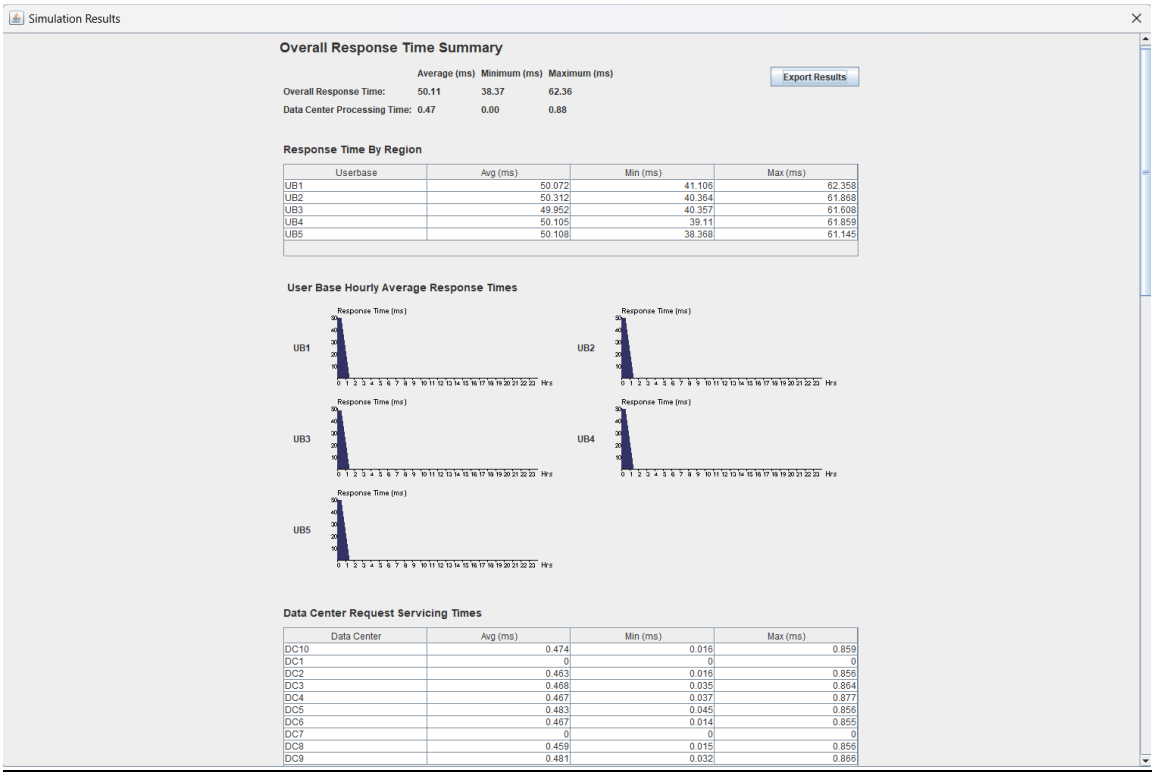


- Honeybee + Simulation Annealing

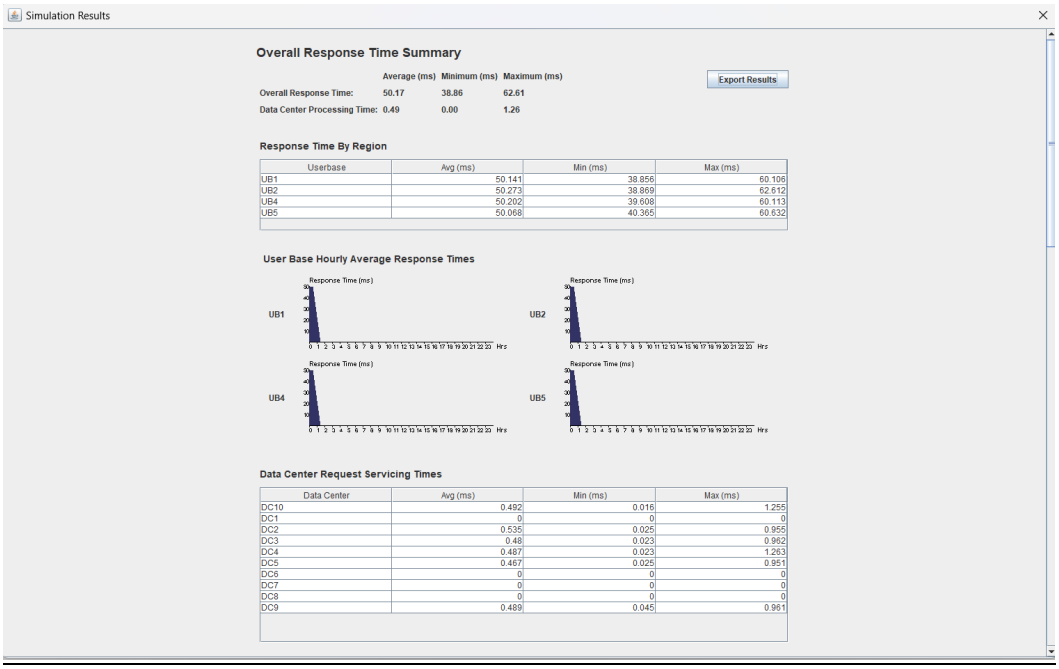


- Honeybee + Tabu

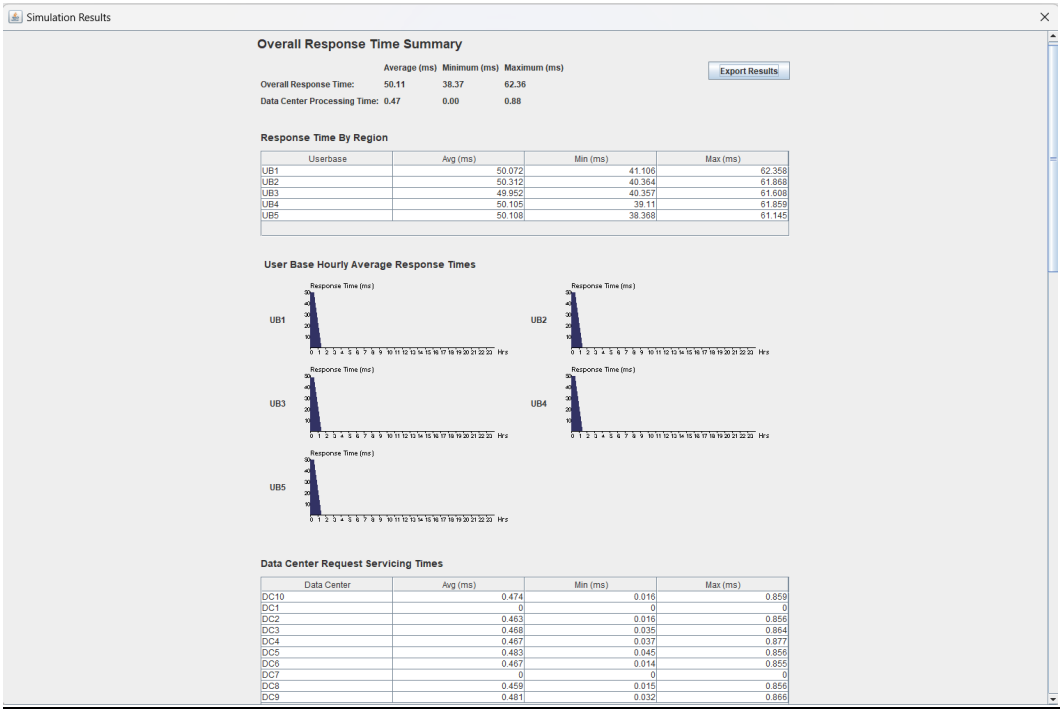
- Honeybee + Particle Swarm Optimization



- Ant Colony + Tabu Search



- Ant Colony + Computational



References

[1] Ajay Jangra, Neeraj Mangla, “An efficient load balancing framework for deploying resource schedulingin cloud based communication in healthcare, Measurement: Sensors”, Volume 25, 2023,

100584, ISSN 2665-9174
https://doi.org/10.1016/j.measen.2022.100584
https://www.sciencedirect.com/science/article/pii/S2665917422002185
[2] Afzal, S., Kavitha, G. Load balancing in cloud computing – A hierarchical taxonomical classification. J Cloud Comp 8, 22 (2019). https://doi.org/10.1186/s13677-019-0146-7
[3] Sara Tabaghchi Milan, Lila Rajabion, Hamideh Ranjbar, Nima Jafari Navimipour,"Nature inspired meta-heuristic algorithms for solving the load-balancing problem in cloud environments", Computers & Operations Research, Volume 110, 2019, Pages 159-187, ISSN 0305-0548 https://doi.org/10.1016/j.cor.2019.05.022
https://www.sciencedirect.com/science/article/pii/S0305054819301352
[4] Suman Sansanwal, Nitin Jain, "An Improved Approach for Load Balancing among Virtual Machines in Cloud Environment", Procedia Computer Science, Volume 215,2022, Pages 556-566, ISSN 1877-0509
https://doi.org/10.1016/j.procs.2022.12.058
https://www.sciencedirect.com/science/article/pii/S1877050922021299
[5] Agrawal, Priyanka & Gupta, Subhash & Choudhury, Tanupriya. (2021). Load Balancing Issues in Cloud Computing. 10.1007/978-981-16-4149-7_10.
[6] Nuaimi, Klaithem & Mohamed, Nader & Nuaimi, M. & Al-Jaroodi, Jameela. (2012). A survey of load balancing in cloud computing: Challenges and algorithms. Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on. 137-142.
[7] Sidhu, Amandeep & Kinger, Supriya. (2005). Analysis of Load Balancing Techniques in Cloud Computing. INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY. 4. 737-741. 10.24297/ijct.v4i2C2.4194.
[8] Nuaimi, Klaithem & Mohamed, Nader & Nuaimi, M. & Al-Jaroodi, Jameela. (2012). A survey of load balancing in cloud computing: Challenges and algorithms. Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on. 137-142.
[9] Ren, Haozheng & Lan, Yihua & Yin, Chao. (2012). The load balancing algorithm in cloud computing environment. 925-928. 10.1109/ICCSNT.2012.6526078.