ADVANCED LOAD BALANCING TECHNIQUES

Bachelor of Technology

Computer Science and Engineering Department

CSDA

Netaji Subhas University of Technology

East Campus, Geeta Colony, Delhi

8th Semester

Mamta
2020UCB6052

Nitesh Mishra
2020UCB6044

MAY 2024

Certificate

This is to certify that the project entitled Analysis of vocal track and implementation of rating system is the bonafide work of **Mamta** and **Nitesh Mishra** who carried out the project work under my supervision. It has been duly presented in the partial fulfilment of the requirements for the degree of Bachelor of Technology, Computer Science Engineering (CSDA Branch) at Netaji Subhas University of Technology, East Campus, Geeta Colony, Delhi.

Dr. Ram Shringar Raw

Associate Professor
Department of Computer Science and Engineering
Netaji Subhash University of Technology
East Campus, Geeta Colony, Delhi

Acknowledgement

We extend our heartfelt gratitude to all those who made it possible for us to complete this report. A special thanks is given to our project supervisor, **Dr. Ram Shringar Raw**, whose contribution in providing feedbacks, encouragement and suggestions, helped us in the coordination of our project, specifically in the writing of this report. Moreover, we would also like to acknowledge with much appreciation the crucial role of the staff of the Computer Science Engineering Department, who granted us the permission to use the required resources and the necessary equipment for completing the task “Advanced Load Balancing techniques”. Last but not least, we would like to thank our parents for their unflagging love and support.

Abstract

Cloud computing has swiftly become a core technology, providing businesses and individuals with unprecedented efficiency and accessibility via a pay-as-you-go basis. However, other obstacles have surfaced, including security vulnerabilities, high infrastructure failure rates, and, most importantly, load balancing issues. This paper focuses on the essential issue in load balancing techniques in the cloud computing and presents a better solution to address it. By dynamically reallocating workloads based on node availability and requirements, the proposed approach intends to reduce burden on individual nodes while optimizing overall system performance.

The study thoroughly examines existing load balancing algorithms, determining their strengths and drawbacks. This analysis proposes a synergistic algorithm capable of dynamically adapting to the demands placed on cloud infrastructure. This integrated method not only optimizes load distribution but also prevents potential overloads on certain nodes. The study provides substantial insights into advanced load balancing strategies, indicating a possible approach to improving the efficiency and performance of cloud computing systems.

The suggested approach attempts to improve the overall efficiency and reliability of cloud computing systems by dynamically addressing load distribution while taking into account node availability and workload needs. The findings provide significant contributions to the field, opening the path for more resilient and scalable cloud architecture in the future.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER-1
INTRODUCTION

1.1 Background

Cloud computing has witnessed significant growth and adoption due to its scalability, flexibility, and cost-efficiency. However, efficient load balancing is still a critical challenge in Cloudcomputing environments. Dynamic unpredictable workload fluctuation results in suboptimal resource utilization, increased response times, and higher operational costs. Traditional load balancing techniques often struggle to adapt to changing demands, resulting in performance bottlenecks and diminished system efficiency.

1.2 Research Gap

Despite advancements in Cloud computing, several gaps persist in load balancing strategies:

1. **Dynamic Workload Fluctuations:** Existing approaches struggle to effectively handle dynamic workload changes, leading to inefficient resource allocation.
2. **Operational Inefficiencies:** Suboptimal load balancing results in increased response times and operational costs due to underutilized resources or overloaded nodes.
3. **Limitations of Current Approaches:** Traditional load balancing methods lack adaptability to evolving Cloud infrastructures, impacting overall system performance.
4. **Growing Scale and Complexity:** With the increasing scale and complexity of Cloud environments, there is a need for more sophisticated and adaptive load balancing solutions.
5. **Need for Advanced Load Balancing:** To address these challenges, there is a pressing need for innovative load balancing algorithms that can optimize job scheduling, enhance system performance, and mitigate operational costs.

1.3 Problem Statement

1.3.1 Objective

In the landscape of Cloud computing, the overarching objective of this research is to pioneer an adaptive load balancing algorithm, specifically employing the Genetic Algorithm. This innovative approach aims to dynamically allocate resources in response to changing workload patterns within Cloud environments. The focal points of this research include optimizing job scheduling, enhancing overall system performance, and mitigating operational costs associated with inefficient resource utilization. A crucial aspect of the research is to evaluate the adaptability of the proposed Genetic Algorithm-based load balancing approach to accommodate dynamic workload fluctuations, ensuring consistent performance under varying operational conditions.

1. Comparison with Traditional Methods: Conduct a comparative analysis between the proposed Genetic Algorithm-based approach and traditional load balancing methods, evaluating performance metrics, responsiveness, and scalability.
2. Develop an Adaptive Load Balancing Algorithm: Design and implement an adaptive load balancing algorithm, leveraging the Genetic Algorithm, to dynamically allocate resources based on changing workload patterns in Cloud computing.
3. Enhance System Performance: Improve overall system performance by fine-tuning load distribution mechanisms, reducing bottlenecks, and maximizing resource utilization through the proposed advanced load balancing approach.
4. Mitigate Operational Costs: Evaluate the impact of the developed algorithm on operational costs, aiming to minimize unnecessary expenditures associated with underutilized resources and inefficient load distribution.

1.3.2 Motivation

In the dynamic landscape of Cloud computing, the rapid evolution of technology has brought about unprecedented opportunities for organizations to leverage scalable and flexible computing resources.

1. Rapid Evolution of Cloud Computing: Cloud computing's swift evolution offers numerous opportunities but introduces challenges, emphasizing the need for innovative solutions.
2. Challenge of Efficient Load Balancing: The pivotal challenge involves ensuring efficient load balancing for optimal resource utilization, responsiveness, and cost-effectiveness in Cloud-based services.
3. Critical Role of Load Balancing: Load balancing is crucial for maintaining seamless performance, adapting to varying workloads, and sustaining overall efficiency in Cloud environments.
4. Contributing to Ongoing Improvement of Cloud Services: The research seeks to contribute to the continuous improvement of Cloud-based services, aiming to foster reliability and efficiency in the face of evolving computational demands.

CHAPTER-2

RELATED LITERATURE STUDY

In the realm of cloud computing, the efficient allocation of tasks and resources is paramount for optimizing system performance and responsiveness. This essay delves into two distinct research works that tackle this challenge through innovative task scheduling and load balancing algorithms.

2.1 Dynamic Task Scheduling:

In the study, conducted by C. K. Rath, P. Biswal, and S. S. Suar, explores "Dynamic Task Scheduling with Load Balancing using Genetic Algorithm" at the 2018 International Conference on Information Technology (ICIT) [1] addresses task mapping in heterogeneous systems by representing applications as Directed Acyclic Graphs (DAGs). Here, nodes symbolize computational tasks, and edges denote communication between tasks, each assigned weights corresponding to computation and communication costs. The survey evaluates static and dynamic task scheduling algorithms, emphasizing the effectiveness of dynamic approaches like Genetic Algorithms. Notably, the survey introduces the Clustering Based HEFT with Duplication (CBHD) algorithm, designed for dynamic scheduling in Heterogeneous Computing Systems (HC). CBHD combines the Triplet Clustering algorithm and HEFT to enhance load balancing and computation overhead management.

2.2 Dual process approach:

In another study by Shafiq et al. published in IEEE Access (2021) [2] , a novel Load Balancing (LB) algorithm is presented, specifically targeting optimized Cloud Computing environments. This unique algorithm adopts a dual-process approach, integrating both Task Scheduling and Load Balancing processes to achieve optimal cloud resource utilization. By addressing workload migration, unbalanced workloads, and task rejection, this model contributes to enhanced performance and responsiveness in cloud

applications. The framework is designed to dynamically adapt to varying workloads, ensuring efficient load balancing and CPU usage optimization. This study underscore the importance of dynamic approaches in tackling the challenges in load balancing in cloud computing. Leveraging innovative algorithms like Genetic Algorithms and a comprehensive dual-process framework, these research endeavours pave the way for enhanced efficiency and performance in cloud environments. The findings highlight the significance of adaptive resource allocation strategies to meet the evolving demands of cloud computing applications.

2.3 Hybrid Approach:

Adding to the exploration of innovative algorithms in optimization, "A Hybrid Differential Evolution Algorithm for Solving the Travelling Salesman Problem" by S. Sivakumar, S. Lakshmi, and M. Praveen Kumar (2021) [3] introduces a novel approach to tackling the classic Travelling Salesman Problem (TSP). In this paper, the authors propose a hybrid differential evolution algorithm (HDEA) that integrates the differential evolution algorithm with a local search approach to enhance both convergence speed and solution quality. The TSP is a well-known combinatorial optimization problem where the objective is to find the shortest possible route that visits each city exactly once and returns to the origin city. The HDEA introduced in this work leverages the strengths of differential evolution, a population-based optimization technique, along with local search techniques to efficiently explore the solution space and refine candidate solutions.

2.4 Genetic Algorithm:

“A Review of Genetic Algorithm and its Applications” by Sanjana Mukherjee and Anish Sachdeva (2020) [4] comprehensively covers genetic algorithms (GAs), their variants, and practical applications. GAs, inspired by natural selection, use selection, crossover, and mutation to optimize solutions iteratively. The review explores different GA variants like adaptive GAs, multi-objective GAs, and hybrid GAs, highlighting their effectiveness in various fields like engineering, robots, finance, bioinformatics, and scheduling. Additionally, the paper discusses the strengths and weaknesses of GAs and proposes

future research directions. This review serves as an important resource in understanding GAs & their wide-ranging applications in finding solutions of difficult optimization problems. The integration of dynamic task scheduling & load balancing algorithms represents a promising avenue for optimizing cloud computing systems. These studies contribute valuable insights and methodologies to the field, offering practical solutions to enhance resource utilization and overall system efficiency in cloud environments.

CHAPTER-3

METHODS AND METHODOLOGY

The project develops metaheuristic load balancing algorithms using Cloud Analyst which is an extension of CloudSim, for real-time modelling and analysis of cloud environments. Our aim is to identify the most effective load balancing algorithm based on data centre response time and service processing metrics. Through simulations, we compare different metaheuristic algorithms to determine the best fit for specific scenarios.

Approach involves developing an advanced load balancing algorithm using Swarm Intelligence, leveraging key steps from initialization to simulation, with components including fitness evaluation, selection (e.g., tournament selection), crossover, and mutation. Simulation validation is conducted using CloudSim, a tool for cloud computing service modelling and experimentation. The research focuses on heuristic job scheduling algorithms in cloud computing, particularly prioritizing Online Mode scheduling algorithms like Ant Colony Optimisation and Honeybee Optimisation due to their adaptability to dynamic processor speeds in cloud environments.

3.1 Ant Colony Optimisation (ACO)

3.1.1 PRINCIPLE

ACO takes an inspiration from the foraging behaviour of ants in finding food. In this approach, artificial ants continuously deposit pheromone trails on paths, influencing their probability of choosing a path based on the amount of pheromone present..

3.1.2 PSUDOCODE

```
Initialize the amount of pheromone on all paths
Repeat until termination condition is met:
  Create ant agents and place them at random starting points
  While ant has not completed its tour:
    Choose the next path based on the amount of pheromone and information
    Update pheromone amount on the chosen path
  Update global pheromone levels based on ant trails
  Evaporate pheromone levels to prevent stagnation
  Check termination condition
```

Steps Used

1. **Initialization:** Initialize amount of pheromone level on all the paths in the network.
2. **Ant's Movement:** Each individual ant chooses the next path depending on the amount of pheromone and heuristic information (e.g., distance).
3. **Update Pheromones:** After all the ants complete their exploration, update the amount of pheromone based on the amount of pheromone deposited by the ants.
4. **Evaporation:** Evaporate again the amount of pheromone on all paths to prevent stagnation and promote exploration.
5. **Termination:** Check if termination condition (e.g., maximum iterations) is met. If not, repeat the process with new ant tours.

3.2 Honeybee Optimization

3.2.1 PRINCIPLE

Honeybee Optimization mimics the foraging behaviour of honeybees, where bees communicate through dance movements to share information about food sources. This algorithm optimizes based on exploration and exploitation of potential solutions.

3.2.2 PSUDOCODE

```
Initialize scout bees and employed bees
Repeat until termination condition is met:
    Employed bees exploit food sources by evaluating neighbouring solutions
    Onlooker bees choose food sources based on employed bees' information and perform waggle dance
    Update employed bees based on onlooker bees' choices
    Scout bees randomly search for new food sources
    Check termination condition
```

Steps Used

1. **Initialization:** Initialize scout bees and employed bees with random solutions.
2. **Exploitation:** Employed bees exploit food sources by evaluating neighbouring solutions.
3. **Communication:** Onlooker bees choose food sources based on employed bees' information and perform a waggle dance to communicate findings.
4. **Update Employed Bees:** Employed bees update their solutions based on onlooker bees' choices.
5. **Exploration:** Scout bees randomly search for new food sources to maintain diversity.
6. **Termination:** Check if termination condition (e.g., maximum iterations) is met. If not, repeat the process with updated solutions.

3.3 Round Robin Algorithm

3.3.1 PRINCIPLE

Round Robin is a simple scheduling algorithm where tasks are assigned to resources in a circular order. Each resource gets an equal share of time to execute tasks, ensuring fairness and minimal waiting times.

3.3.1 PSUDOCODE

```
Initialize a list of resources
```

```
Initialize a queue of tasks
Repeat until all tasks are completed:
   For each resource in the list:
      If resource is available:
         Assign the next task from the queue to the resource
         Execute the task for a fixed time quantum
         Move the task to the end of the queue if not completed
      Move to the next resource in the list
```

Steps Used

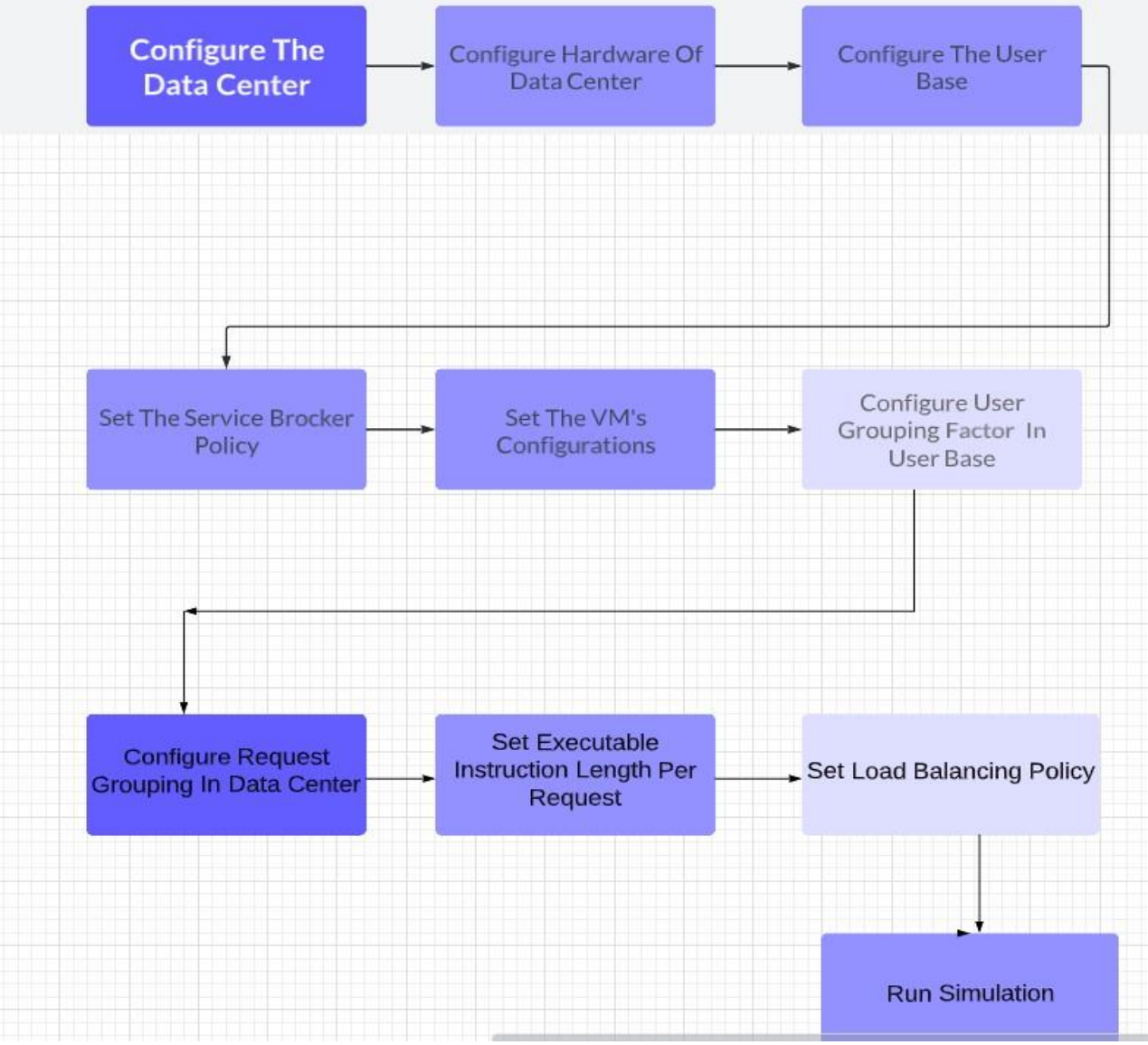
1. **Initialization:** Prepare a list of resources and a queue of tasks.
2. **Task Assignment:** Iterate over each resource in a circular manner.
3. **Task Execution:** Assign the next task from the queue to the resource and execute it for a fixed time quantum.
4. **Task Management:** If the task is not completed within the time quantum, move it to the end of the queue.
5. **Resource Rotation:** Move to the next resource in the list and repeat the task assignment process.
6. **Completion:** Continue the process until all tasks are completed.

CHAPTER-4
SIMULATION

4.1 Steps Involved in Simulation:

1. **Launch CloudAnalyst:** Start by launching the CloudAnalyst application on your system. Ensure that CloudSim is properly configured and integrated with CloudAnalyst.
2. **Configure Data Centre Parameters:** Define the characteristics of the cloud data centre, such as the number of hosts (physical machines), virtual machines (VMs), data centre architecture, and resource allocation policies. This step involves specifying details like processing capacity, memory capacity, storage capacity, and bandwidth of each host and VM.
3. **Define User Base:** Specify the characteristics of the user base accessing the cloud services. This includes setting the number of users, their arrival patterns (e.g., uniform, exponential), and workload requirements (e.g., amount of CPU and memory needed for tasks).
4. **Configure Application Deployment:** Define the applications to be deployed on the cloud infrastructure. Specify the number and type of tasks associated with each application, their execution requirements, and dependencies between tasks.
5. **Select Load Balancing Policies:** Choose the load balancing policies or algorithms to be evaluated during the simulation. CloudAnalyst supports various load balancing strategies such as Round Robin, Ant Colony Optimization (ACO), Honeybee Optimisation , and more.
6. **Run the Simulation:** Start the simulation within the CloudAnalyst environment. The software will simulate the interaction between users, applications, and the cloud infrastructure based on the specified parameters and load balancing policies.

7. **Monitor Simulation Progress:** During the simulation, CloudAnalyst provides real-time monitoring and visualization of key performance metrics. This includes data on response times, resource utilization, task scheduling efficiency, and overall system performance.
8. **Analyse Simulation Results:** After the simulation completes, analyse the results generated by CloudAnalyst. Evaluate the impact of different load balancing strategies on system performance, scalability, and resource allocation efficiency.
9. **Optimize Parameters and Re-run Simulations (Optional):** Based on the analysis, refine simulation parameters, adjust load balancing policies, or experiment with different configurations. Re-run the simulations to compare outcomes and identify optimal settings for cloud infrastructure management.
10. **Generate Reports and Insights:** Use CloudAnalyst to generate comprehensive reports and insights based on simulation results. Communicate findings, trends, and recommendations for improving cloud resource management and workload distribution.



4.2 Key Performance Metrics

The simulation captured the following key performance metrics:

1. **Response Time:** Average time taken to respond to user requests.
2. **Resource Utilization:** Percentage of CPU and memory resources utilized.
3. **Task Scheduling Efficiency:** Efficiency of task allocation and execution across VMs.
4. **System Throughput:** Number of tasks processed per unit time.

CHAPTER-5

Results and Discussion

5.1 Ant Colony Optimisation

Response Time by Region

Userbase	Avg (ms)	Min (ms)	Max (ms)
UB1	50.19	37.61	60.86
UB2	50.01	38.64	60.39
UB3	49.98	38.89	61.07
UB4	50.27	39.13	62.38
UB5	299.99	225.14	360.14
UB6	300.10	232.64	363.14

Overall Response Time Summary

	Average (ms)	Minimum (ms)	Maximum (ms)
Overall Response Time:	134.42	37.61	363.14
Data Center Processing Time:	0.46	0.02	1.26

5.2 Honeybee Optimisation

Response Time by Region

Userbase	Avg (ms)	Min (ms)	Max (ms)
UB1	50.18	37.61	60.36
UB2	50.00	38.64	60.39
UB3	49.98	38.89	62.41
UB4	50.24	39.13	60.88
UB5	300.26	225.14	363.14
UB6	299.93	232.64	364.64

Overall Response Time Summary

	Average (ms)	Minimum (ms)	Maximum (ms)
Overall Response Time:	134.42	37.61	364.64
Data Center Processing Time:	0.45	0.02	0.90

Round Robin Algorithm

Response Time by Region

Userbase	Avg (ms)	Min (ms)	Max (ms)
UB1	50.20	37.61	60.96
UB2	50.05	38.64	60.39
UB3	50.06	38.89	61.66
UB4	50.26	39.13	60.38
UB5	299.91	225.14	363.15
UB6	409.49	232.64	31372.26

Overall Response Time Summary

	Average (ms)	Minimum (ms)	Maximum (ms)
Overall Response Time:	152.62	37.61	31372.26
Data Center Processing Time:	18.66	0.02	31072.75

CHAPTER-6

CONCLUSION AND FUTURE STUDY

6.1 Conclusion

6.1.1 Accomplishments:

The cloud computing simulation utilizing Honeybee Optimization, Ant Colony Optimization (ACO), and Round Robin algorithms using CloudAnalyst and CloudSim has yielded several significant accomplishments:

- **Comparison of Algorithms:** Comparative analysis of algorithm performance based on key features like time in the response, resource utilization, task scheduling efficiency.
- **Identification of Optimal Strategy:** Identification of the Genetic Algorithm (GA) as the most effective load balancing strategy, showcasing its adaptability and efficiency in dynamic workload scenarios.

6.1.2 Contributions:

This study contributes valuable insights to the field of cloud computing load balancing by:

- **Highlighting Effective Techniques:** Demonstrating the effectiveness of Swarm Intelligence-based approaches (Honeybee Optimization and ACO) alongside traditional Round Robin scheduling in optimizing cloud resource utilization.
- **Informing Decision-Making:** Providing decision-makers with actionable data on the performance of different load balancing algos to enhance the cloud infrastructure management practices.

6.1.3 Impact:

The impact of this research extends to improving system efficiency and responsiveness through the adoption of advanced load balancing techniques. It has the potential to reduce operational costs associated with cloud resource provisioning and utilization.

6.2 Future Study

6.2.1 Exploration of Hybrid Load Balancing Approaches

One key area of future work involves exploring hybrid load balancing approaches that combine Swarm Intelligence techniques (such as ACO and Honeybee) with machine learning algos. By leveraging strengths in different methodologies, hybrid approaches can enhance adaptability, scalability, and performance optimization in dynamic cloud environments. Research efforts could focus on developing novel hybrid algorithms that intelligently adapt to varying workload patterns, leveraging historical data and real-time feedback to optimize resource allocation.

6.2.2 Real-World Deployment and Validation

Validating optimized load balancing strategies in real-world cloud environments is essential to assess scalability, reliability, and cost-effectiveness in diverse operational settings. Future research should prioritize real-world deployment of advanced load balancing algorithms, collaborating with industry partners to implement and evaluate these strategies in production cloud infrastructures. Through rigorous testing and validation, researchers can gain insights into the practical implications and performance characteristics of optimized load balancing techniques, facilitating wider adoption and implementation across cloud service providers.

6.2.3 Integration of Security Mechanisms

Integrating security mechanisms into load balancing algorithms represents a critical area of future research. Cloud computing environments are inherently vulnerable to security threats, and load balancing strategies must incorporate robust security measures to protect sensitive data and mitigate potential risks. Future studies should explore the integration of encryption techniques, access controls, and anomaly detection algorithms within load balancing frameworks to enhance data protection and ensure the integrity and confidentiality of cloud-based services.

REFERENCES

[1] "*Dynamic Task Scheduling with Load Balancing using Genetic Algorithm*," 2018 International Conference on Information Technology (ICIT), Bhubaneswar, India, 2018, pp. 91-95 by C. K. Rath, P. Biswal and S. S. Suar,

[2] "*A Load Balancing Algorithm for the Data Centres to Optimize Cloud Computing Applications*." IEEE Access, 9, 3065308 by Shafiq, D. A., Jhanjhi, N. Z., Abdullah, A., & Alzain, M. A. (2021).

[3] "*A Hybrid Differential Evolution Algorithm for Solving the Travelling Salesman Problem*" by S. Sivakumar, S. Lakshmi, and M. Praveen Kumar (2021)

[4] "*A Review of Genetic Algorithm and its Applications*" by Sanjana Mukherjee and Anish Sachdeva (2023)

[5] CloudSim for simulation of Cloud computing related works (http://www.cloudbus.org/cloudsim/)

[6] Cloud Analyst Software (http://www.cloudbus.org/cloudsim/CloudAnalyst.zip)

A. Sample Code:

Ant Colony Optimisation:

```
package cloudsim.ext.datacenter; //[5][6]
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import cloudsim.VirtualMachine;
import cloudsim.VirtualMachineList;

public class AntColonyVmLoadBalancer extends VmLoadBalancer
{
   // Holds the values of pheromones
   private double[][] pheromones;
   static final double alpha = 1;
   static final double beta = 1;
   static final double ONE_UNIT_PHEROMONE = 1;
   static final double EVAPORATION_FACTOR = 2;
   private final int NUM_ANTS = 10;

   Ant[] ants;
   DatacenterController dcbLocal;

   public AntColonyVmLoadBalancer(DatacenterController dcb)
   {
      super();
      dcbLocal = dcb;
      // ((VirtualMachine)vmList.get(0)).getMemory()
   }

   @Override
   public int getNextAvailableVm()
   {
      pheromones = new double[dcbLocal.vmlist.size() + 1][dcbLocal.vmlist.size() + 1];
      ants = new Ant[NUM_ANTS];

      for (int i = 0; i < ants.length; i++)
```

```
    {
        ants[i] = new Ant(pheromones);
    }

    for (int ant = 0; ant < ants.length; ant++)
    {
        ants[ant].SendAnt();
        Evaporation();
    }

    Ant queryAnt = new Ant(pheromones);
    return queryAnt.FetchFinalVm();
}

public void Evaporation()
{
    for (int i = 0; i < pheromones.length; i++)
    {
        for (int j = 0; j < pheromones.length; j++)
        {
            pheromones[i][j] /= EVAPORATION_FACTOR;
        }
    }
}

public class Ant
{
    private double[][] pheromones;
    private boolean[] isVisited;
    public List<Integer> path;
    private int fakeVmId;

    public Ant(double[][] ph)
    {
        pheromones = ph;
        isVisited = new boolean[pheromones.length];
        fakeVmId = isVisited.length - 1;
        path = new ArrayList<Integer>();
    }
```

```
public int SendAnt()
{
   return ProcessAnt(true /* updatePheromones */);
}

public int FetchFinalVm()
{
      return ((VirtualMachine) dcbLocal.vmlist.get(ProcessAnt(false /* updatePheromones
*/))).getVmId();
}

public int ProcessAnt(boolean updatePheromones)
{
   int CurrentVmId = fakeVmId;
   int nextVmId = getNextVmNode(CurrentVmId);

   if (updatePheromones)
   {
      UpdatePheromone(CurrentVmId, nextVmId);
   }
   while (nextVmId != CurrentVmId)
   {
      path.add(nextVmId);
      CurrentVmId = nextVmId;
      nextVmId = getNextVmNode(CurrentVmId);
      if (updatePheromones)
      {
         UpdatePheromone(CurrentVmId, nextVmId);
      }
   }

   if (updatePheromones)
   {
      UpdateGlobalPheromones();
   }

   return CurrentVmId;
}
public int getNextVmNode(int vmId)
{
```

```
    double[] probability = computeProbability(vmId);
    Random rand = new Random();
    double randomization = rand.nextDouble();
    for (int i = 0; i < probability.length; i++)
    {
        randomization = randomization - probability[i];
        if (randomization <= 0)
        {
            return i;
        }
    }
    for (int i = 0; i < probability.length; i++)
    {
        System.out.println("Debug " + probability[i]);
    }
    return -1;
}

// Assumes there is at least one node that has note been visited
public double[] computeProbability(int vmId)
{
    double[] probability = new double[pheromones.length - 1];
    double sum = 0.0;
    for (int i = 0; i < probability.length; i++)
    {
        if (isVisited[i])
        {
            probability[i] = 0;
            continue;
        }
        probability[i] = scoreFunction(vmId, i);
        sum += probability[i];
    }

    // Normalize
    for (int i = 0; i < probability.length; i++)
    {
        probability[i] = probability[i] / sum;
    }
    return probability;
```

```
        }

        public void UpdatePheromone(int prevId, int newId)
        {
            pheromones[prevId][newId] += ONE_UNIT_PHEROMONE;
        }

        // TODO make more optimum.
        public double scoreFunction(int prevVmId, int newVmId)
        {
            // todo cost factor
            double maxBw = ((VirtualMachine) dcbLocal.vmlist.get(newVmId))
                    .getCharacteristics().getBw();
            double currentBw = ((VirtualMachine) dcbLocal.vmlist.get(newVmId))
                    .getBw();
            // double requestedBw = cloudlet.getUtilizationOfBw(0);
            return Math.pow(pheromones[prevVmId][newVmId], alpha) + 1.0
                    + (maxBw - currentBw / maxBw);
        }
        public void UpdateGlobalPheromones()
        { }
    }
}
```

Honeybee Optimisation:

```
package cloudsim.ext.datacenter;
import java.util.*;

import cloudsim.ext.Constants;
import cloudsim.ext.event.CloudSimEvent;
import cloudsim.ext.event.CloudSimEventListener;
import cloudsim.ext.event.CloudSimEvents;

public class HoneyBee extends VmLoadBalancer implements CloudSimEventListener
{
    private int cutoff = 8;
    private int scoutBee = -1;
    private Map<Integer, VirtualMachineState> vmStatesList;
    Map<Integer, Integer> vmAllocationCounts = new HashMap<Integer, Integer> ();
```

```
Map<Integer, Integer> fitness = new HashMap<Integer, Integer> ();


public HoneyBee(DatacenterController dcb)
{
    this.vmStatesList = dcb.getVmStatesList();
    dcb.addCloudSimEventListener(this);
}

/* This will return the VM Id on which CLOUDLET should be submitted.. To get VM, we are using Honey Bee */
@Override
public int getNextAvailableVm()
{
    int vmId = -1;
    vmId = getScoutBee();
    scoutBee = vmId;
    allocatedVm(vmId);
    return vmId;
}

public void cloudSimEventFired(CloudSimEvent e)
{
    if (e.getId() == CloudSimEvents.EVENT_CLOUDLET_ALLOCATED_TO_VM)
    {
        int vmId = (Integer) e.getParameter(Constants.PARAM_VM_ID);
        int countGunjanCloudlets;
        if(vmAllocationCounts.get(vmId)==null)
            countGunjanCloudlets = 0;
        else
            countGunjanCloudlets = vmAllocationCounts.get(vmId);
        vmAllocationCounts.put(vmId,countGunjanCloudlets+1);

        if(vmAllocationCounts.get(vmId)>cutoff)
            vmStatesList.put(vmId, VirtualMachineState.BUSY);
    }
    else if (e.getId() == CloudSimEvents.EVENT_VM_FINISHED_CLOUDLET)
    {
        int vmId = (Integer) e.getParameter(Constants.PARAM_VM_ID);
        int countGunjanCloudlets = vmAllocationCounts.get(vmId);
```

```
        vmAllocationCounts.put(vmId,countGunjanCloudlets-1);

        if(vmAllocationCounts.get(vmId)<cutoff)
            vmStatesList.put(vmId, VirtualMachineState.AVAILABLE);
    }
}
private boolean isSendScoutBees(int scoutBee)
{
    if((vmAllocationCounts.get(scoutBee)==null)||(vmAllocationCounts.get(scoutBee) < cutoff))
        return false;
    else
        return true;
}

/* This will return food source */
int getScoutBee()
{
    if(scoutBee==-1)
    {
        if(vmStatesList.size()>0)
            return 0;
        else
            return -1;
    }
    else
    {
        if(isSendScoutBees(scoutBee)==false)
            return scoutBee;
        else
        {
            SendEmployedBees();
            return SendOnlookerBees();
        }
    }
}
int MemorizeBestSource()
{
    return waggleDance();
}
int SendOnlookerBees()
```

```
    {
        return MemorizeBestSource();
    }
    void calculation()
    {
         int i;
         for (i=0;i<vmStatesList.size();i++)
         {
             if(vmAllocationCounts.get(i)==null)
                 fitness.put(i, 0);
             else
                 fitness.put(i, calculateFitness(vmAllocationCounts.get(i)));
         }
    }
    int calculateFitness(int solValue)
    {
        return solValue;
    }

    void SendEmployedBees()
    {
      fitness.clear();
      calculation();
    }
    private int waggleDance()
    {
        int Min, i=0;
        Min = 0;
        int global = fitness.get(0);
        for(i=1;i<vmStatesList.size();i++)
        {
                if(fitness.get(i)< global)
                {
                    global = fitness.get(i);
                    Min = i;
                }
        }
        return Min;
    }
}
```

B. PLAGIARISM REPORT

ORIGINALITY REPORT

5%	2%	3%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	"Nature-Inspired Computing and Optimization", Springer Nature, 2017 Publication	1%
2	iajit.org Internet Source	1%
3	Coastal Systems and Continental Margins, 1995. Publication	1%
4	Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, Rajkumar Buyya. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms", Software: Practice and Experience, 2011 Publication	1%
5	9pdf.org Internet Source	1%
6	www.diva-portal.org Internet Source	<1%