Optimal Task Scheduling in Cloud Computing

IMRAN HOSSAIN
MD. PARIMUL HAQUE
MD. NAEEM-AL-IMRAN

A thesis submitted for the degree of
BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering
Dhaka University of Engineering & Technology, Gazipur

June 2023

Optimal Task Scheduling in Cloud Computing

IMRAN HOSSAIN

Student no: 174075

Reg:10376, Session:2020-2021

MD. PARIMUL HAQUE

Student no: 174077

Reg:10378, Session:2020-21

MD. NAEEM-AL-IMRAN

Student no: 174078

Reg:10379, Session:2020-21

Supervisor: Dr. Mohammad Abdur Rouf

Professor, Department of Computer Science and Engineering

A thesis submitted for the degree of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering

Dhaka University of Engineering & Technology, Gazipur

June 2023

Declaration

It is hereby declared that this paper titled ”Optimal Task Scheduling in Cloud Computing”, is our own work and it is written under the supervision of **Dr. Mohammad Abdur Rouf** and in accordance with the guidelines and regulations. This work has not previously been submitted for publication or any other academic qualification.

Signature

.....
(IMRAN HOSSAIN)

.....
(MD. PARIMUL HAQUE)

.....
(MD. NAEEM-AL-IMRAN)

.....
Dr. Mohammad Abdur Rouf
Professor
Department of Computer Science and Engineering

Acknowledgments

We want to give a special thanks to our thesis supervisor **Dr. Mohammad Abdur Rouf, Professor, Department of Computer Science and Engineering, DUET, Gazipur**, for giving us a great opportunity and motivating us to work on this thesis. Without his consistent guidance, support and help, we would not have been able to complete this thesis. He has played a crucial role in bringing this thesis to its current stage. We sincerely appreciate him. We also want to express our gratitude to Engr. Md. Sirajul Islam, System Analyst, DUET, Gazipur, for his continuous support and assistance.

Abstract

Now a days, we rely heavily on our machines and devices to perform a multitude of tasks. However, certain tasks demand significant computational power, either due to their complexity or resource requirements. In these situations, purchasing high-end machines can be prohibitively expensive or impractical. This is where cloud services come into play. Cloud service providers offer virtual machines that can efficiently handle a wide range of client tasks. These providers cater to countless clients who rely on the cloud to execute their tasks effectively. As a result, the task scheduling process becomes crucial for service providers to ensure optimal resource utilization and timely task completion.

To address this challenge, we have developed an advanced task scheduling algorithm specifically designed for cloud computing environments. Our algorithm, called Modified Particle Swarm Optimization (MPSO), leverages the principles of swarm intelligence inspired by the collective behavior of birds or fishes. PSO is a population-based optimization algorithm that simulates the movement and interaction of particles in a search space. Each particle represents a potential solution to the task scheduling problem. By exploring and adjusting their positions in the search space, particles collectively converge towards an optimal solution. In the context of cloud computing, the task scheduling problem involves allocating tasks to virtual machines (VMs) in a manner that optimizes various objectives, such as minimizing task completion time, maximizing resource utilization and achieving load balancing. The MPSO algorithm effectively addresses these objectives by dynamically assigning tasks to VMs based on their characteristics and availability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Cloud computing has become increasingly popular for researchers and industries. Efficient task scheduling, which focuses on maximizing throughput and achieving load balancing. It is crucial for optimizing resource usage and system performance. This research aims to enhance the Particle Swarm Optimization (PSO) algorithm to achieve improved throughput and load balancing in the cloud computing environment.

We customize the PSO algorithm by maximizing throughput and achieving load balancing. This involves adjusting the algorithm's rules and incorporating adaptive mechanisms. These changes help the algorithm explore and exploit effectively, considering task dependencies, resource constraints and workload characteristics.

Through extensive experimentation, we compare our enhanced PSO algorithm with standard PSO variants and other well-known scheduling approaches. We evaluate metrics such as throughput, makespan and the Average Resource Utilization Ratio (ARUR).

This research contributes to advancing task scheduling techniques in cloud computing. The enhanced PSO algorithm offers a promising solution for achieving higher throughput and better load balancing, resulting in improved resource utilization and overall system performance.

1.2 Motivation

The motivation behind this research paper is to present an optimized version of the PSO algorithm specifically tailored for cloud task scheduling. The primary objectives of our research are to enhance the throughput of task execution and achieve load balancing across virtual machines, leading to improved system efficiency and client satisfaction.

Throughput Enhancement: The proposed optimization techniques will focus on improving the overall task execution rate in cloud environments. By dynamically assigning tasks to virtual machines with optimal resource allocation, we seek to minimize task completion time and maximize overall system throughput.

Load Balancing: Load balancing in task scheduling involves distributing tasks evenly across available resources to optimize resource utilization, ensure fair workload distribution and enhance system performance. It aims to avoid overloading resources, prioritize critical tasks, dynamically allocate resources and adapt to changing workload conditions. The goal is to achieve efficient task execution and timely completion while maximizing system throughput and minimizing response time.

Scalability: Cloud computing environments often deal with large-scale task scheduling scenarios. Our optimized PSO algorithm will consider scalability challenges, ensuring its effectiveness even when dealing with a vast number of tasks and virtual machines.

Through this research, we aim to contribute to the existing body of knowledge in cloud task scheduling by presenting an optimized PSO algorithm that addresses the specific challenges of throughput enhancement and load balancing. The outcomes of this study will be valuable for cloud service providers, researchers and practitioners, empowering them to design more efficient and reliable cloud systems while meeting the diverse needs of their clients.

1.3 Research Objectives

- ➢ Improvement of Throughput
- ➢ Load Balance

- ➢ Reduce Makespan
- ➢ Reduced total execution time

1.4 Contributions

This research paper makes the following key contributions to the field of cloud task scheduling:

i) Optimized PSO Algorithm: The paper introduces an optimized version of the Particle Swarm Optimization (PSO) algorithm specifically tailored for cloud task scheduling. By enhancing the algorithm's design and mechanisms, we improve its performance in terms of throughput and load balancing in cloud computing environments.

ii) Throughput Enhancement Techniques: We propose innovative techniques within the optimized PSO algorithm to enhance the overall task execution rate in cloud environments. By dynamically allocating tasks to virtual machines with optimal resource utilization, we minimize task completion time and improve the system's throughput. Our research provides practical insights into achieving higher efficiency and faster task execution in cloud task scheduling scenarios.

iii) Load Balancing Strategies: Achieving load balancing among virtual machines is crucial for fair resource distribution and optimal system performance. This paper presents novel load balancing strategies embedded within the optimized PSO algorithm. By intelligently assigning tasks based on resource availability, workload distribution and system dynamics, we ensure better utilization of resources and an equal distribution of tasks among virtual machines.

iv) Experimental Validation: To evaluate the effectiveness and superiority of our proposed optimization techniques, we conduct extensive experiments and provide comprehensive experimental results. Through these experiments, we demonstrate the performance enhancements achieved by the optimized PSO algorithm in terms of throughput improvement and load balancing in various cloud task scheduling scenarios. The experimental findings contribute empirical evidence supporting the practical applicability of our approach.

By combining these contributions, our research significantly advances the state-of-the-art in cloud task scheduling. The proposed optimized PSO algorithm provides cloud service providers, researchers, and practitioners with a robust and efficient solution for handling task scheduling in cloud computing environments. It is not only enhances the overall system throughput but also achieves load balancing, resulting in improved resource utilization, reduced task completion time and enhanced client satisfaction.

1.5 Organization of the Thesis

In the rest of the thesis book
Chapter 2: Literature Review - Provides literature review and background information.
Chapter 3: Proposed Method - Describes the proposed approach for task allocation.
Chapter 4: Result and Discussion - Presents results and analysis.
Chapter 5: Conclusion and Future Work - Summarizes findings and outlines future research.

Chapter 2

Literature Review

2.1 Overview

The task scheduling in cloud computing has received considerable attention due to its crucial role in optimizing resource utilization, achieving load balancing and improving system performance. Various approaches have been proposed, including genetic algorithms, ant colony optimization and particle swarm optimization (PSO). In this thesis, our focus is on conducting significant research related to PSO-based task scheduling algorithms in cloud computing environments.

PSO has demonstrated its effectiveness in solving optimization problems in various domains, including task scheduling. The concept of utilizing PSO for task scheduling in cloud computing was first introduced by Kennedy and Eberhart (1995) [4], who adapted PSO for solving the job-shop scheduling problem. Since then, researchers have explored and expanded upon this idea, adapting PSO for cloud task scheduling scenarios.

Zheng et al. (2012) proposed a PSO-based task scheduling algorithm for cloud computing, taking into account task deadlines and resource constraints. Their approach aimed to minimize task completion time and maximize resource utilization. Through experimental evaluation, they demonstrated the effectiveness of the algorithm in achieving superior task scheduling results compared to conventional approaches.

In a similar manner, Yang et al. (2014) presented a PSO-based algorithm that took into account multiple objectives, including makespan, cost, and energy utilization, in cloud task scheduling. Their approach employed a multi-objective fitness function and a cooperative PSO variant to optimize task assignment to virtual machines. The experimental results

demonstrated improved performance in terms of makespan reduction and load balancing.

To address task scheduling, Li et al. (2016) proposed a PSO-based algorithm that consolidated energetic task scheduling and task balancing mechanisms. Their approach aimed to achieve superior resource utilization and fairness in task scheduling across virtual machines. The algorithm dynamically balanced the particle swarm based on real time workload data, resulting in improved load balancing and reduced task execution time.

In recent years, machine learning techniques have also been integrated with PSO for cloud task scheduling. Wang et al. (2019) proposed a hybrid algorithm that combined PSO with a deep learning-based prediction model. The algorithm utilized historical task execution data to forecast task completion time and dynamically adjusted the PSO parameters for improved scheduling decisions. The experimental results demonstrated improved task completion time and load balancing.

Whereas existing considers have contributed to the field of PSO-based task scheduling in cloud computing, there's still room for advancement in terms of throughput upgrade and load balancing. This paper builds upon the existing writing by showing an optimized PSO calculation particularly custom-made for cloud task scheduling. Our approach focus on improving throughput and accomplishing load balancing through inventive strategies inserted inside the PSO calculation. The test comes about given in this paper illustrate the prevalent execution and viability of our optimized calculation compared to existing approaches.

2.2 Background Study

2.2.1 Task

In cloud computing, a task mention to a particular computational operation performed inside the cloud environment. The task in the context of cloud computing refers to a unit of work, such as running a program application or processing data. Task scheduling involves allocating these assignments to available cloud resources, with the aim of optimizing resource utilization and minimizing task completion time.

2.2.2 Cloud computing

Cloud computing is a revolutionary innovation that enables on-demand access to shared computing resources over the Internet. This eliminates the need for users to maintain their own physical infrastructure and allows them to leverage various resources such as computing power, storage and networking.

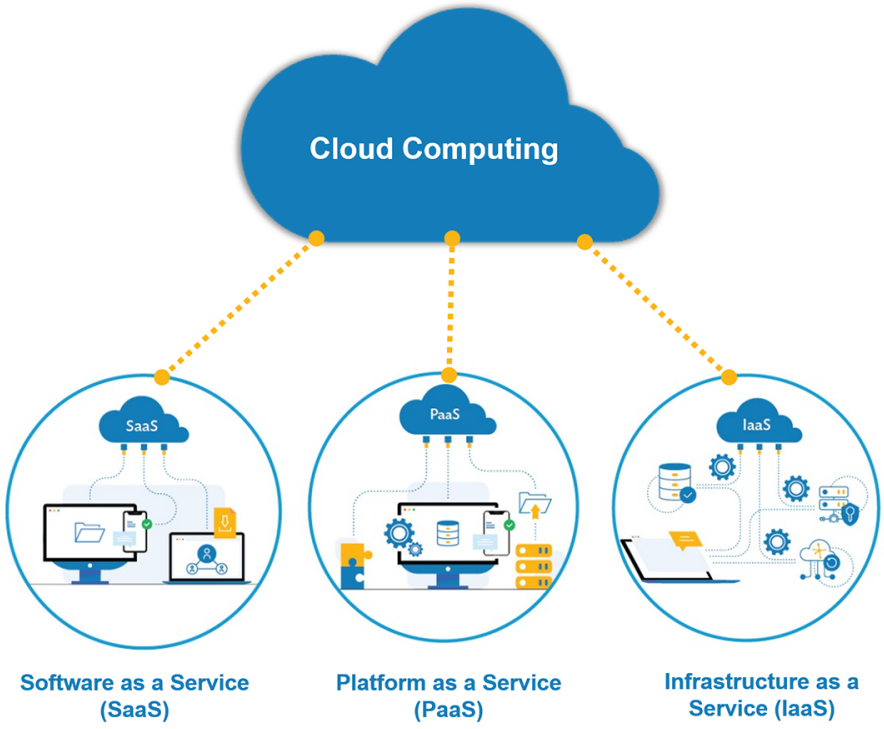

Figure 2.1: Cloud Computing [1]

Cloud computing offers different service models such as infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). Its scalability allows users to easily adjust resource allocation to meet their needs. This flexibility combined with cost efficiency and high availability enables organizations to adapt quickly and seize opportunities. Cloud computing improves collaboration and data security and supports the development of new technologies such as artificial intelligence, big data analytics, and the Internet of Things (IoT). Cloud computing has become very popular due to its many advantages and is revolutionizing how computing services are delivered and accessed.

2.2.3 Task scheduling

Task scheduling in cloud computing involves allocating and managing tasks on available resources for optimal efficiency. It aims to optimize resource utilization, minimize task completion time, and maximize system throughput. Factors such as task requirements, resource availability, dependencies, and deadlines are considered. Strategies like load balancing and dynamic scheduling are used to evenly distribute tasks, prioritize critical ones, and adapt to changing conditions. Efficient task scheduling improves resource utilization, system performance, and user satisfaction in cloud computing environments.

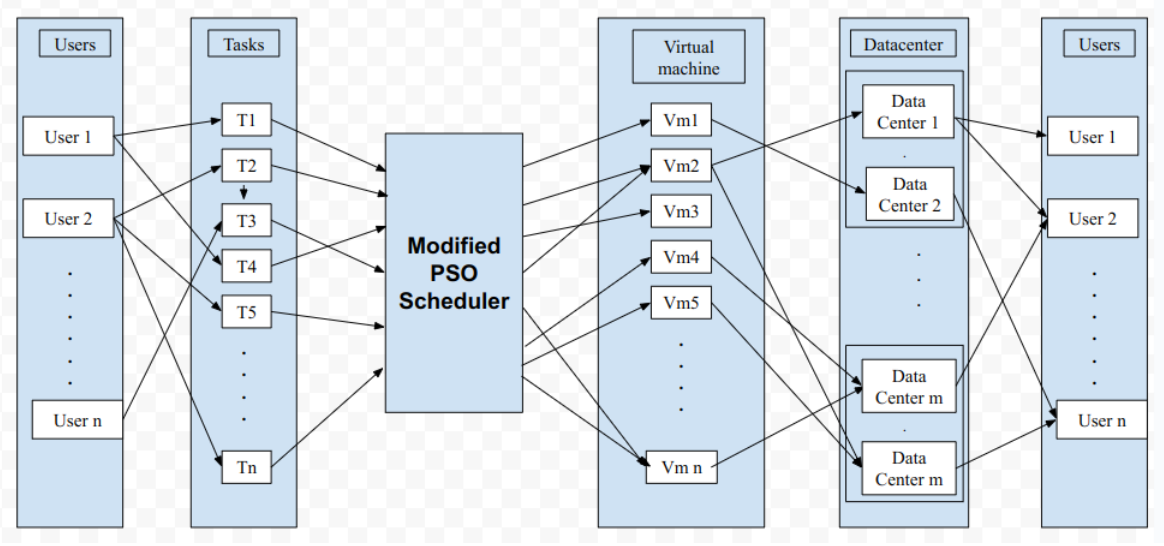

Figure 2.2: Task Scheduling [2]

2.2.4 Virtual Machine (VM)

A virtual machine (VM) is a computer system created using software on a physical computer to emulate the functionality of a separate physical computer. VMs allow multiple operating systems and applications to run simultaneously on a single machine, improving resource utilization and flexibility. They provide isolation, enable better hardware utilization and offer features like snapshotting and live migration. VMs are widely used in testing, development and cloud computing environments.

2.2.5 Makespan

Makespan in cloud computing refers to the total time required to complete a specific set of tasks. It represents the duration from the start of the first task to the completion of the last task in the set. Makespan is a critical performance metric in task scheduling and resource allocation algorithms as it measures the overall efficiency and timeliness of task execution. Minimizing makespan is often a key objective in optimizing task scheduling to ensure prompt task completion and maximize system throughput.

2.2.6 Throughput

In cloud computing, throughput describes how fast a system can process activities and data. It indicates how fast the system can efficiently complete tasks and process data. Task completion rate, or the amount of data processed in a period of time, is a common way to assess throughput.

High throughput means that the system can process many orders and data quickly and effectively. This is an important performance parameter for cloud computing as it directly affects overall system effectiveness and user satisfaction.

2.2.7 Position

The allocation of tasks to virtual machines (VMs) is referred to as ”position” in PSO. It shows how tasks are split among VMs and adjusted to enhance performance while minimizing resource utilization.

2.2.8 Velocity

In the context of task scheduling in cloud computing using particle swarm optimization (PSO), the velocity component is really important. It governs how particles navigate the solution space and arrive at improved task scheduling solutions. The velocity guides the particles' movement, balancing exploration and exploitation. By updating the velocity based on historical information and the best solutions, PSO improves the particles' ability to explore and converge in the context of cloud task scheduling.

2.2.9 Personal Best (pBest)

In PSO, a particle's personal best is the best solution it has found in its search history. It acts as a reference point for the particle's travel and aids in directing the particle's search to promising places in the search space.

2.2.10 Global Best (gBest)

The global best (gBest) in PSO indicates the best solution found by any particle in the whole swarm. It acts like a global reference point, allowing all particles to compare their own solutions to the best solution discovered thus far. The gBest leads the swarm as a whole, supporting exploration and convergence on optimum solutions.

2.2.11 Response Time

The time it takes for a system to react to a request or finish an activity is referred to as response time. It is a key information for assessing system efficiency and user experience. Response times that are shorter suggest higher performance and faster execution of services. Identifying bottlenecks and enhancing system components are every component of optimizing response time.

2.2.12 Load Balancing

The process of equally spreading workloads across many resources in order to enhance performance and utilize resources is known as load balancing. It reduces overloading and increases the scalability and stability of the system.

2.3 Related Work

2.3.1 Genetic Algorithm

In cloud computing, genetic algorithms (GA) [5] are used as an optimization technique to address various challenges related to resource allocation, task scheduling, and load balancing. GA in cloud computing aims to find optimal or near-optimal solutions by mimicking the principles of natural selection and genetics.

Genetic algorithms in cloud computing are typically based on representing potential solutions as chromosomes or individuals within a population. Each chromosome encodes a set of parameters or decisions that define the assignment of tasks to resources within the cloud environment.

The GA process involves iterative generations in which individuals undergo selection, recombination (crossover), and mutation manipulations. Selection selects individuals with higher levels of fitness. This is usually determined by evaluating performance against a specific fitness function or objective measure.

Recombination, or crossover, combines genetic material from two or more individuals to produce new offspring. This allows you to consider different combinations of task and resource assignments. Mutations introduce small random changes in the genetic material, creating diversity and avoiding falling into local optimization.

2.3.2 Teaching Learning Based Optimization (TLBO)

Teaching Learning Based Optimization (TLBO) [6] is a hybrid load balancing approach that combines the strengths of Gray Wolves Optimization (GWO) and TLBO algorithms. The goal of this approach is to achieve efficient load balancing, both in terms of time required and cost.

TLBO is an optimization algorithm inspired by the process of teaching and learning in the classroom. It involves two important phases: teaching and learning. In the teaching phase, better people and better solutions guide others to perform better. During the learning phase, individuals learn from information shared by others and adapt their solutions accordingly.

GWO is another optimization algorithm based on the social hierarchy and hunting behavior of gray wolves. This mimics the leader hierarchy within a wolf pack, with alpha, beta and delta wolves representing the primary individuals leading the search process.

A hybrid approach combines the advantages of TLBO and her GWO to meet the load balancing challenges. By integrating these two algorithms, the hybrid approach aims to optimize the load balancing process in terms of time and associated costs. Effective load bal-

ancing distributes workload evenly across available resources, minimizing response time and maximizing resource utilization. By considering time and cost factors, the hybrid TLBO-GWO approach can achieve efficient load balancing, resulting in improved system performance and reduced operating costs.

This hybrid approach leverages the strengths of TLBO and GWO to create a more robust and effective load balancing strategy. Combining each optimization technique and harnessing the power of social interaction and learning provides a comprehensive solution for optimal load balancing in complex systems.

2.3.3 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) [7] is a population-based optimization algorithm that draws inspiration from the social behavior of particles in a swarm. In PSO, a group of particles iteratively explores the search space by adjusting their positions based on personal experience and the collective knowledge of the swarm.

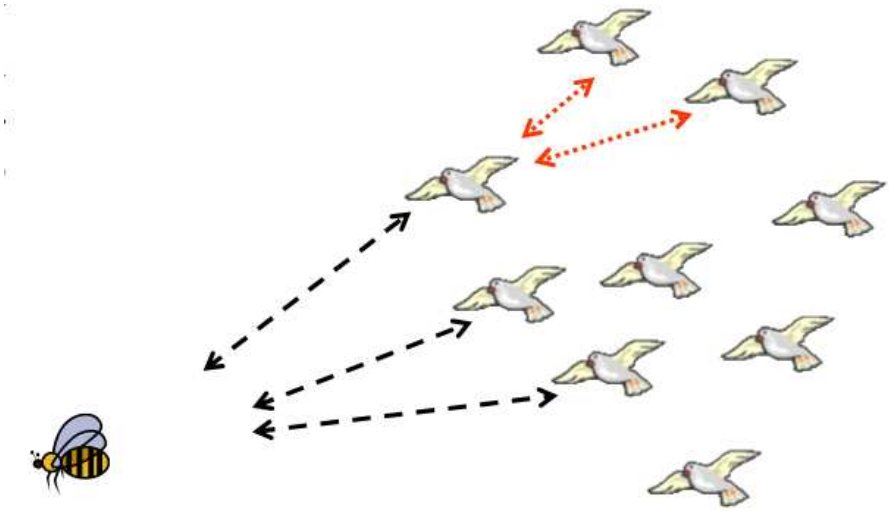

Figure 2.3: Particle Swarm Optimization [3]

Each particle in the swarm represents a potential solution to the optimization problem. The particles move through the search space, and their positions are updated based on their own best-known solution (personal best) and the best-known solution of the entire swarm (global best).

During each iteration, particles adjust their velocities based on a combination of their historical movement and the attractive force of the personal and global best solutions. These velocities guide the particles towards promising regions in the search space, facilitating the exploration and exploitation of potential solutions.

The iterative process continues until a termination condition is met, such as reaching a maximum number of iterations or achieving a desired level of convergence. At the end of the optimization process, the particle with the best-known solution represents the optimal solution or an approximation of it.

PSO has been successfully applied to a wide range of optimization problems, including function optimization, parameter tuning, data clustering and task scheduling in cloud computing. Its simplicity, efficiency and ability to handle both continuous and discrete optimization problems make it a popular choice in various domains.

By leveraging the collective intelligence and adaptive behavior of particles, PSO offers a metaheuristic approach for finding optimal or near-optimal solutions in complex optimization problems. Its ability to balance exploration and exploitation makes it a valuable tool for solving real-world optimization challenges.

Chapter 3

Proposed Approach

This section provides a methodology overview of the task scheduling algorithm. This is consists of two main components. Proposed inertia weight strategy and task scheduler. We also provide an overview of swarm-based particle swarm optimization (PSO) algorithms as background for the proposed methodology.

3.1 Modified PSO Model

The modified PSO algorithm is a self-adaptive optimization approach that uses global search. This is a population-based planning method that exploits the social behavior of particles. Inspired by the behavior of schools of fish and flocks of birds [8], this swarm intelligence-based approach leverages generations and particles. Generations represent the total number of iterations required to obtain an optimized solution, while each particle within a generation represents a single solution. The number of generations and particles can vary depending on the specific case, and they are adjusted to achieve a more optimal solution. Each particle possesses attributes such as position, velocity, local/personal best (pBest), global best (gBest), and load best (lBest). The personal best represents the most optimal solution found by an individual particle, while the global best represents the best solution among all particles. The load best represents the optimal load balance within the global best solution. At each iteration, the velocities and positions of all particles are updated based on various factors such as inertial weight, pBest, gBest and lBest. These updates lead the particles to more promising solutions as the algorithm progresses.

Suppose D represents the dimension of the solution space, where X_i is a vector representing the particle's position in the search space. i.e.,

$$X_i = (x_{i1}, x_{i2}, x_{i3}, ..., x_{id}) \tag{1}$$

With each iteration, the particle constantly changes its position, looking for more positions a perfect solution.

$$Pb_i = (pb_{i1}, pb_{i2}, pb_{i3}, ..., pb_{id}) \tag{2}$$

Pb_i represents the optimal solution for each particle (as shown in equation (2)). V_i denotes the velocity of the particle (as shown in equation (3)).

$$V_i = (v_{i1}, v_{i2}, v_{i3}, ..., v_{id} \tag{3}$$

Equation (4) is used by PSO to find the updated velocity of each particle.

$$v_{id}^{k+1} = w * v_{id}^{k} + c_1 r_1 (pb_{id}^{k} - x_{id}^{k}) + c_2 r_2 (Gb_{d}^{k} - x_{id}^{k}) \tag{4}$$

where $i = 1, 2, 3, ..., n$ (denoting the number of particles), $k = 1, 2, 3, ..., itrmax$ (maximum number of iterations, i.e. 200 in this article) iteration), D indicates the number of dimensions or number of tasks that should be allocated to the VM in an optimized way. $X_i^K d$ is the current position and v_{id}^{k} is the current velocity of the i^{th} particle of k^{th} iterations in d-dimensional space. The parameter w is an inertia weight that balances global and local searches for particles, and c_2 and c_1 are constant acceleration factors. r_1 and r_2 are random values between 0 and 1. Equation (5) gives the updated position of the particle[8].

$$x_{id}^{k+1} = (v_{id}^{k+1} + x_{id}^{k}) \tag{5}$$

Calculate the best value (personal best) for each element based on your fitness Functions based on maximization or minimization problems. The best fit factor for all individuals is called load balancing according to the global best.

$$Fitness Function = Minimization\ of\ Execution\ Time\ \&\ Maximization\ of\ lBest \tag{6}$$

If the new value is better than the current value, each particle's pBest value is updated at each iteration to account for load balancing. This update allows each particle to maintain its own optimal solution for load balancing problems. The PSO-based heuristic computes and records by the best values of gBest and lBest.

lBest is the maximum expected computation time among the minimum expected computation times of all virtual machines. It helps in load balancing by considering the max-

imum workload among the available virtual machines.(i.e. gBest shown in the Equation (7))

$$gBest = max(pBest_1, pBest_2, pBest_3, ..., pBest_n) \tag{7}$$

To solve the task scheduling problem using PSO-based heuristics and input schedules as search solutions, we identify a good map between PSO particles and problem solutions[9]. Each PSO particle represents a possible solution for the VM mapping task. Tables 3.1–3.3 are used to illustrate the relationship between PSO particles and problem solving. Table 3.3 shows a VM mapping task using PSO-based swarm intelligence. Five VMs with different computing power in millions of instructions per second (MIPS) are used for this mapping. The processing power of VM1 is 120 MIPS, 150, 50, 200, and 400 MIPS, corresponding to the processing power of VM2, VM3, VM4, and VM5 respectively (see Table 3.1).

Table 3.1: Computation power of VMs.

VMs	**VM1**	**VM2**	**VM3**	**VM4**	**VM5**
VMs Computation power(in MIPs)	120	150	50	70	100

Table 3.2 shows 10 problems with millions of different computational requirements. Instructions (MI) mapped to 5 VMs (Table 3.1).

Table 3.2: Computation requirements of Tasks.

Task	**Tsk1**	**Tsk2**	**Tsk3**	**Tsk4**	**Tsk5**	**Tsk6**	**Tsk7**	**Tsk8**	**Tsk9**	**Tsk10**
MIs	50	200	300	130	600	250	350	850	450	160

Table 3.3 shows the assignment of 10 tasks to 5 VMs. P11 (Particle 1, Iteration 1) represents the first possible solution containing Tsk8 associated with VM1. Tsk5, Tsk9 to VM2. Tsk4, Tsk10 to VM3. Tsk3, Tsk6 to VM4 and VM5 is assigned Tsk1, Tsk2, Tsk7 respectively.

Table 3.3: Tasks to VM mapping.

Task	**Tsk1**	**Tsk2**	**Tsk3**	**Tsk4**	**Tsk5**	**Tsk6**	**Tsk7**	**Tsk8**	**Tsk9**	**Tsk10**
P11	VM5	VM5	VM4	VM3	VM2	VM4	VM5	VM1	VM2	VM3
P12	VM4	VM5	VM1	VM4	VM3	VM1	VM5	VM2	VM4	VM5
P13	VM2	VM1	VM4	VM3	VM2	VM1	VM2	VM5	VM5	VM4
P14	VM5	VM1	VM2	VM1	VM4	VM5	VM2	VM4	VM2	VM1
-	-	-	-	-	-	-	-	-	-	-
P44	VM2	VM3	VM1	VM5	VM1	VM4	VM2	VM5	VM3	VM1

3.2 System Model

The goal of any task scheduling scheme is to choose an optimized task assignment. Strategies for cloud resources that can reduce and improve task execution time ARUR and cloud throughput. Consider a user application consisting of a set of D independent computationally intensive tasks. i.e. Tsks = Tsk1, Tsk2, Tsk3, ... , Tskid that should be scheduled on a set of M VMs, i.e. H. Ts = T1, T2, T3, ..., Td. It should be scheduled on a set of M VMs. That is, i.e. VMs = V1, V2, V3, ..., Vm, D >>M. For example, Table 3.1 shows the computing power (in MIPS) of a VM (M = 5) that needs to run 10 tasks (that is, D = 10). Computing requirements on MI. If task TD maps to VMj , the Expected Completion Time (ETC) of the assigned task is calculated using Equation (8).

$$ETC_{dj} = RT_j + EET_{dj} \tag{8}$$

where RTj is the ready time of VMj, i.e. the time it takes for H.VM to be ready Assigned workload. EETdj is the expected execution time (EET) of task TskD on VMj, determined by dividing the task computational requirement (in MIs) by the computational power of the VM (in MIPS), using equation (9).

$$EET_{dj} = \frac{T_D size}{VM_j \; computation \; power} \tag{9}$$

Equation (10) calculates the total time it takes VMj to complete all its assigned tasks represented by CTj.

$$CT_j = \sum_{D=1}^{S_j} (ETC_{Dj}) \tag{10}$$

where Sj is the number of tasks scheduled (assigned) to VMj. Makespan is the maximum completion time of all VMs and is calculated as: It is shown in equation (11). Minimizing makespan improves performance with respect to early execution of workloads.

$$Makespan = max(CT_1, CT_2, CT_3, ..., CT_m) \tag{11}$$

where m is the number of VMs. Throughput is another important metric in cloud computing. throughput It is the ratio of the total number of tasks executed per unit of time[8]. in our case, The overall cloud data center throughput is defined as the ratio of the total number of tasks running in the data center and the makespan (expressed in equation (12)). Higher throughput means better performance. H. Perform more tasks in unit time.

$$Throughput = \frac{D}{Makespan} \tag{12}$$

where D is the total number of tasks running in the cloud data center. The objective function of the proposed approach is based on the maximization problem and computed using Equation (13).

$$ObjectiveFunction = max(throughput + (1/makespan)) \tag{13}$$

where D is the total number of tasks running in the cloud data center. The objective function of the proposed approach is based on the maximization problem and computed using Equation (13). where makespan [10] represents the maximum execution time of the data center. H. A VM's execution time to finish executing its assigned tasks when all other VMs terminate, and throughput in this case is the ratio of the total number of tasks executed by the cloud to the makespan. Higher throughput and shorter processing time provide the most benefit to the objective function.

3.3 Inertia Weight Strategies for PSO Model

Any meta-heuristic optimization method based on swarm intelligence. It consists of two main phases. These phases include local search and global search. The balance between

global and local search plays an important role in finding the best solution. Ideally, the global search space should be larger than the local search space at the start of the search [11]. This allows population-based meta heuristics to explore more of the search space first and then more carefully to find the overall optimal position. Inertial weight is the most powerful control factor for balancing local and global searches.

According to literature reviews, inertial weight has been studied by various researchers A selection strategy for balancing local and global searches for particles. However, this balance still needs to be improved. Researchers have proposed several inertial weight strategies, some of which are popular in the research community. These inertia weighting strategies include simple random inertia weights (SRIW) [12], chaotic inertia weights (CIW) [11, 13], chaotic random inertia weights (CRIW) [13], linear descent inertia weights (LDIW) [11, 14] are included and adaptive inertia weights (AIW).

Optimized PSO inertia weight was proposed by represented by the formula (Shown in Equation (14)).

$$W_PA = \frac{w_1 - w_2}{P_s} + \frac{MAXitr - Itr}{MAXitr} * \frac{w_1 - (w_1 - w_2)}{P_s} \tag{14}$$

Where P_s represent particle success rate. And MAXitr is the maximum iteration.

The SRIW inertial weight was proposed by and is given by (Equation (15)).

$$W_{SRIW} = 0.5 + (0.5 * rand()) \tag{15}$$

where rand() represents a random value between 0 and 1. The equation (shown in Equation (16)) describes the CRIW proposed in [13].

$$W_{CDIW} = (w_1 - w_2) * (\frac{MAXitr - Itr}{MAXitr}) + (w_2 * z) \tag{16}$$

where z represents any value between 0 and 1 and rand() represents a random value between 0 and 1. Equation (17) shows the CDIW strategy proposed by [11, 13]

$$W_{LDIW} = (w_1 - w_2) * (\frac{MAXitr - Itr}{MAXitr}) + (w_2) \tag{17}$$

where w_2 and w_1 represent the initial and final inertia weights and z is a random value between 0 and 1. MAXitr represents the maximum number of iterations and Itr is the

current iteration. LDIW was proposed by [11, 14] and is given by Equation (17).

3.4 Proposed Load Balancing Strategy

The load balancing performance of the modified PSO algorithm plays an important role in achieving efficient resource utilization and system optimization. Successful load balancing depends on implementing an optimal load balancing strategy that distributes the workload evenly across the available resources. In the algorithmic context, load balancing is performed to find the globally optimal solution. This represents the best solution found among all the particles in the swarm.

To achieve load balancing, the proposed approach is used, which consists of finding the largest global optimal solution considering the minimum expected execution time of all virtual machines. This is expressed mathematically in Equation (18).

$$lBest = max(min(VM_1ETC, VM_2ETC, ..., VM_mETC)) \tag{18}$$

The purpose of this approach is to identify the virtual machines with the highest workload among all available virtual machines.

By considering the maximum workload, the modified PSO algorithm can try to distribute tasks in a well-balanced manner among the virtual machines. This is important because an unbalanced workload can lead to underutilized or overloaded resources, which can degrade system performance. The algorithm attempts to minimize the variance between the expected execution times of virtual machines by assigning tasks in a workload-balancing manner.

The load-balancing strategy implemented in the modified PSO algorithm ensures that each virtual machine is optimally used and tasks are assigned in a manner that minimizes execution time and maximizes resource efficiency. The lBest value guides the algorithm in making decisions about task assignment and resource utilization with the goal of achieving an overall optimized solution.

Efficient load balancing can have a significant impact on system performance, scalability, and cost efficiency in cloud computing environments. By evenly distributing the

workload across your resources, you reduce the risk of resource bottlenecks and make better use of available computing power. This improves response time, reduces resource idle time, and increases overall system efficiency.

3.5 Proposed Scheduler

The suggested scheduling algorithm is described and shown in this section. The input parameters consist of a collection of jobs with computing needs measured in million of instructions (MI) and a set of VMs with computation capabilities measured in millions of instructions per second (MIPS). The suggested method's output entails the task mapping to virtual machines (VMs) for load balancing and throughput optimization [8].

Algorithm 1: Modified PSO scheduler

1. Define the function getExecutionTime(), which calculates the execution time for a task on a virtual machine based on their instructions.
2. Define the function getVmReadyTime(), which calculates the ready time for a virtual machine based on its total instructions and processing capacity.
3. Define the function getpBestMap(), which returns the best execution time and load balance for a given assignment of tasks to virtual machines.
4. Initialize the taskList and vmList variables.
5. Get the number of tasks (NoTask) and virtual machines (NoVm) from the lists.
6. Initialize the parameters for the particle swarm optimization (PSO) algorithm: position (pos), velocity (v), inertia weight factor (w), constant accelerator factors (c1, c2), weight factors (w1, w2) and the particle success rate (Ps).
7. Initialize the pBestMap, VmReadyTime, ParticleMap and gBestMap variables.
8. Initialize the particles' positions, pBestMap, ParticleMap and gBestMap using the initializeParticles() function.
9. Perform the PSO algorithm iterations until the maximum iteration count (Mxit) is reached.

10. Update the inertia weight factor (w) based on the current iteration.

11. For each particle (prts) in the swarm:

    (a) For each task (tsk) in the taskList:

        i. Generate random numbers (r1 and r2) between 0 and 1.

        ii. Update the velocity (v) based on the current velocity, cognitive and social terms, and the difference between the current position (pos) and the best position (pBestMap) and global best position (gBestMap).

        iii. Update the position (pos) by adding the velocity (v).

        iv. If the position (pos) is outside the valid range of virtual machines (NoVm), randomly assign a new position.

        v. Get the virtual machine (VMachine) for the current particle (ParticleMap).

        vi. Calculate the execution time for the current task (tsk) on the selected virtual machine (VMachine) using the getExecutionTime() function.

        vii. Calculate the virtual machine ready time for the selected virtual machine (VMachine) using the getVmReadyTime() function.

        viii. Update the VmReadyTime map with the new ready time for the VMachine.

        ix. Update the ParticleMap with the new assignment of the task (tsk) to the virtual machine (VMachine).

    (b) Calculate the best execution time and load balance for the current ParticleMap (VmReadyTimeMap) using the getpBestMap() function.

    (c) If the calculated pBestValue.executionTime is better than the current pBest Map value for the particle (prts), update the pBestMap and increment the success count (ss).

    (d) If the calculated pBestValue.executionTime is better than the current gBest. executionTime or if it is equal to gBest.executionTime but has a better load balance (pBestValue.lBest $>$ gBest.lBest), update the gBestMap and gBest variables.

12. Update the particle success rate (Ps) by dividing the success count (ss) by the number of particles (noParticles).

13. If the particle success rate (Ps) is negative, set it to 1.

14. Increment the iteration count (it).

15. Repeat from step 9 until the maximum iteration count (Mxit) is reached.

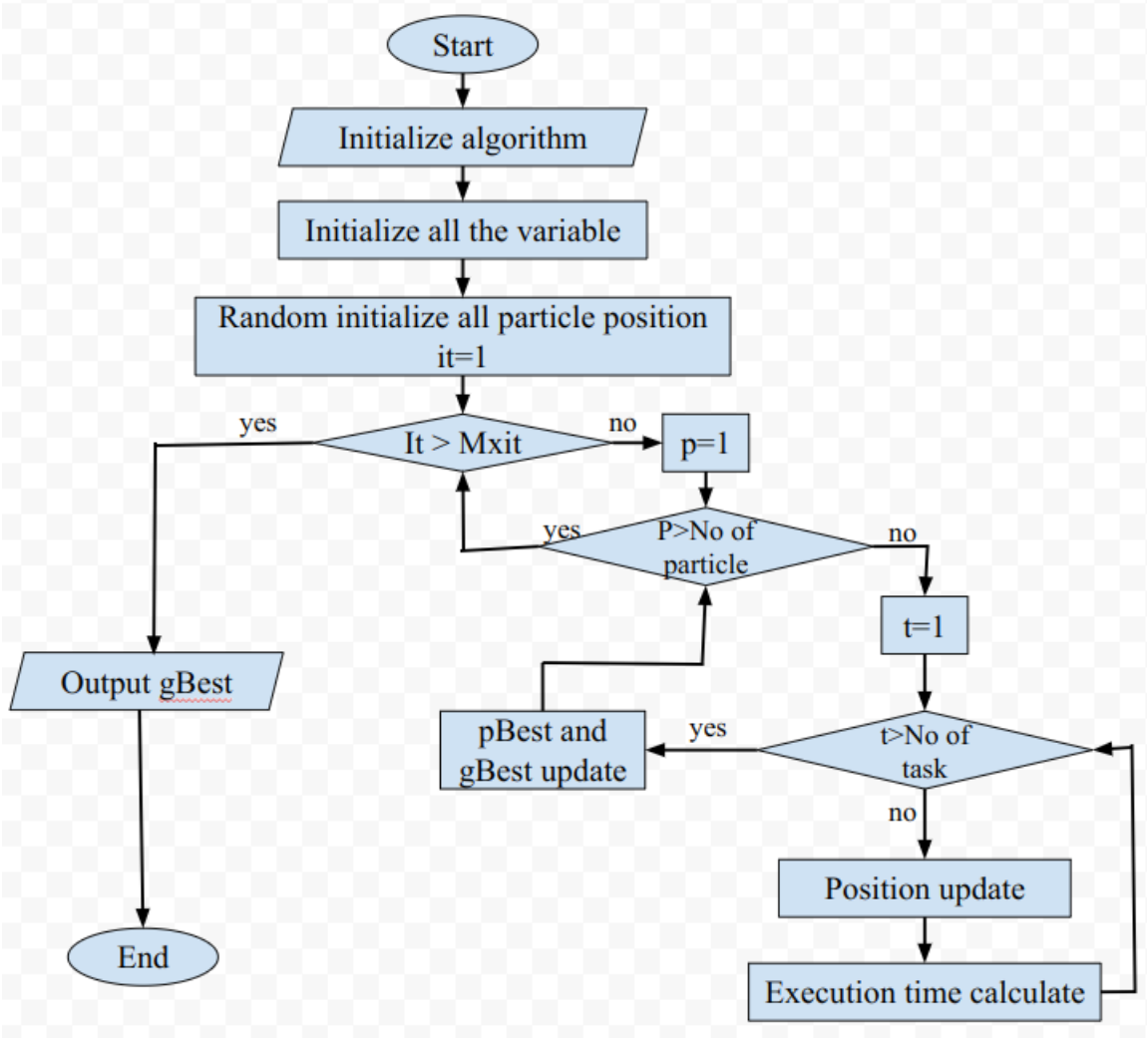

Figure 3.1: Flowchart of Modified PSO Algorithm

Chapter 4

Result and Discussion

4.1 Experimental Setup

In this section, we evaluate the performance of the proposed algorithm and compare it with standard particle swarm optimization. To run the simulations, the scheduling algorithm is implemented in Codeblocks software running on a PC with Intel i3-6100U (4) at 2.30 GHz. This PC has 4 GB of RAM and runs on Ubuntu 22.04.2 LTS x86_64 operating system. The virtual cloud computing environment and PSO algorithm parameters are shown in Table 4.1 and Table 4.2, respectively.

Table 4.1: Parameters of cloud system

Parameters	**Values**
No of Tasks	100-1000
Size of tasks (MI)	5000-10000
No of Virtual Machines	50-250
Virtual Machine Capacity (MIPS)	5000-10000

Table 4.2: Parameters of PSO

Parameters	**Values**
Total Iterations	100
Total Particles	50
Position bound	0-(Virtual Machine-1)
Inertia weight w1, w2	0.4, 0.9
Accelaration Factor c1, c2	1.49455, 2

To perform a comprehensive evaluation, genetic algorithms (GA), teaching-learning-based optimization (TLBO), and particle swarm optimization (PSO) were selected as planning algorithms for comparison in this study. rice field. In addition, the proposed algorithm developed in this work is compared with three other his recently developed metaheuristic algorithms, namely GA, TLBO and PSO. These algorithms are gaining popularity in the field of optimization because they can effectively search for optimal or near-optimal solutions to various problem domains.

Algorithm performance is evaluated and compared against three key metrics: Makespan (MS), total execution time (TET), and load balancing. The program runs random task lengths and virtual machine (VM) execution speeds within a certain range: 500 to 2000 at million instructions (MI) and 100 to 1000 at millions of instructions per second (MIPS). to generate

This simulation setup takes into account random task lengths to assess the dynamic nature of the cloud computing environment. This allows a comprehensive assessment of the algorithm's performance, especially with respect to the three metrics mentioned above. By comparing the results of the evaluated algorithms, researchers and readers can gain insight into the effectiveness and suitability of algorithms for solving specific problems.

4.2 Performance Analysis

Performance evaluation results are determined based on two sets of simulation scenarios described below.

1. **Scenario 1:** Fixed number of tasks and variable number of virtual machines In this scenario, the number of tasks is fixed at 500 in 100 iterations, but the number of virtual machines (VMs) varies between 50 and 500. Algorithms are evaluated and compared using different configurations of VMs, allowing analysis of performance under different computing resources. Metrics such as makespan (MS), total execution time (TET), and load balancing are measured and compared across these different VM configurations.

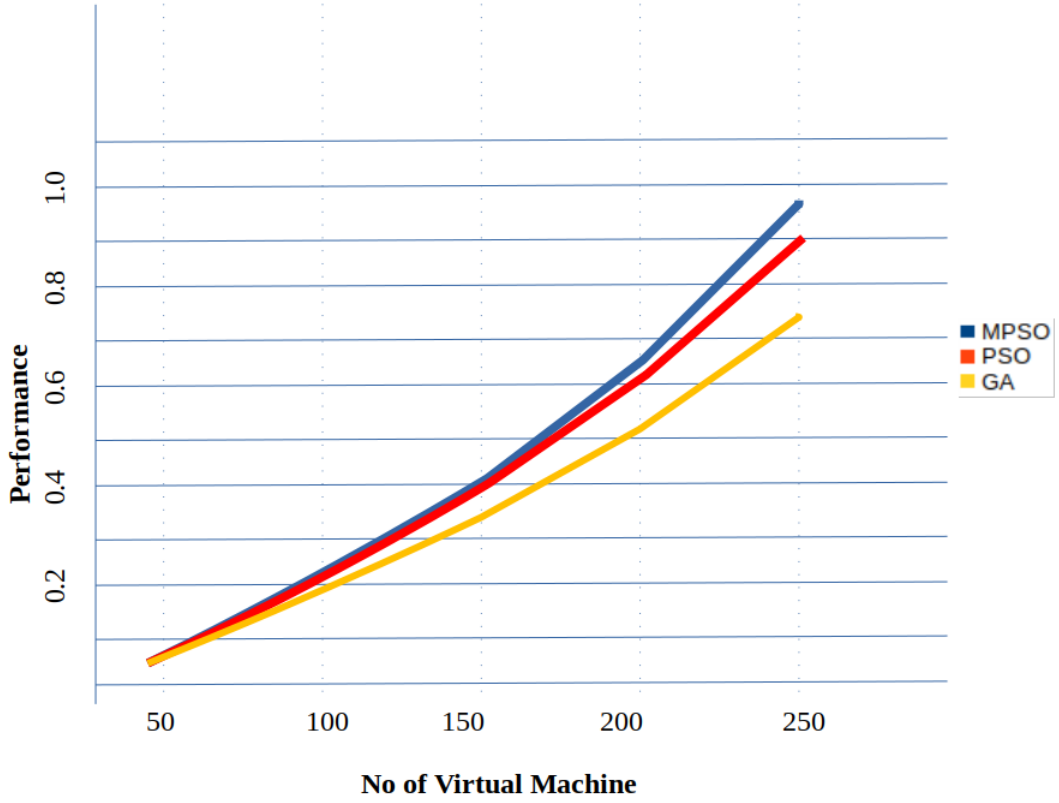

Figure 4.1: Performance for the different Virtual Machine

2. **Scenario 2:** Variable number of tasks, fixed number of virtual machines In this scenario, the number of tasks is between 100 and 1000, but the number of VMs is fixed at 200. Algorithms are tested and compared against different sets of tasks, so you can evaluate their performance in handling different workloads. We use the same metrics as Scenario 1 to evaluate and compare the performance of algorithms such as makespan, total execution time, and load balancing.

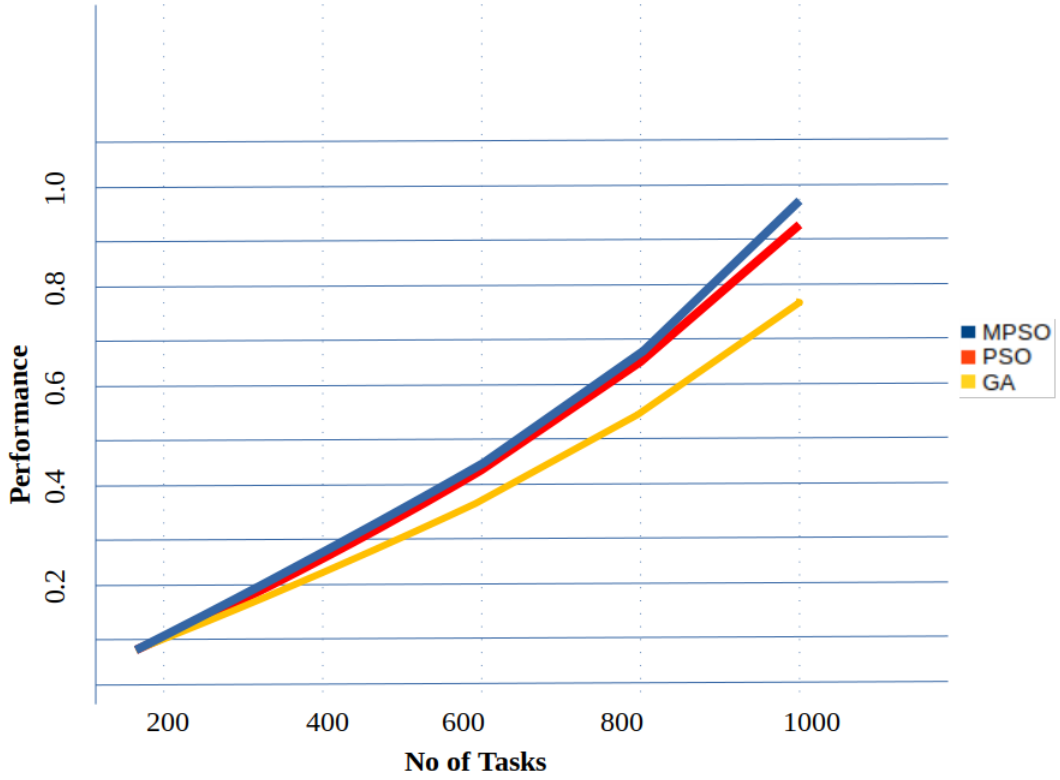

Figure 4.2: Performance for the different Tasks

Conducting performance evaluations in these two simulation scenarios provides a comprehensive understanding of the algorithm's capabilities, strengths, and limitations. This approach enables informed analysis and comparison of performance under different conditions, providing insight into suitability for different workload and resource allocation scenarios.

Chapter 5

Conclusion and Future Work

The utilization of the Particle Swarm Optimization (PSO) method for task scheduling in cloud computing settings has been examined in this work. It is a challenging optimization problem, as task scheduling in the cloud requires the efficient allocation of tasks to available resources while considering various constraints and objectives. By harnessing the concept of swarm intelligence, PSO, a population-based meta-heuristic algorithm, has demonstrated potential in addressing this issue.

We have discussed the essential stages for applying PSO to task scheduling in cloud computing during the course of our research. The concept of the issue, particle representation, fitness evaluation, PSO startup, update rules, and termination criteria were all covered. By carefully structuring the task and resource sets and constructing an objective function that captures pertinent performance measures, we may direct the PSO algorithm to find work schedules of high quality.

The majority of IoT applications [15, 16] demand real-time replies in order to make precise decisions. Future work will focus on response time optimization and scheduling to give delay-sensitive applications real-time or nearly real-time responses.

References

[1] A. E. Youssef, "Exploring cloud computing services and applications," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 3, no. 6, pp. 838–847, 2012.

[2] M. Zhang, L. Liu, C. Li, H. Wang, and M. Li, "A particle swarm optimization method for ai stream scheduling in edge environments," *Symmetry*, vol. 14, no. 12, 2022. [Online]. Available: https://www.mdpi.com/2073-8994/14/12/2565

[3] B. N. A. Samed and A. A. Aldair, "Design tunable robust controllers for unmanned aerial vehicle based on particle swarm optimization algorithm." *Iraqi Journal for Electrical & Electronic Engineering*, vol. 15, no. 2, 2019.

[4] S. S. PABBOJU and D. ADILAKSHMI, "An improved approach for scheduling in cloud using ga and pso," *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 18, 2022.

[5] J. Meena, M. Kumar, and M. Vardhan, "Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint," *IEEE Access*, vol. 4, pp. 5065–5082, 2016.

[6] S. Mousavi, A. Mosavi, and A. R. Varkonyi-Koczy, "A load balancing algorithm for resource allocation in cloud computing," in *Recent Advances in Technology Research and Education: Proceedings of the 16th International Conference on Global Research and Education Inter-Academia 2017 16*. Springer, 2018, pp. 289–296.

[7] E. S. Alkayal, N. R. Jennings, and M. F. Abulkhair, "Survey of task scheduling in cloud computing based on particle swarm optimization," in *2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*. IEEE, 2017, pp. 1–6.

[8] S. Nabi, M. Ahmad, M. Ibrahim, and H. Hamam, "Adpso: adaptive pso-based task scheduling approach for cloud computing," *Sensors*, vol. 22, no. 3, p. 920, 2022.

[9] A. Al-Maamari and F. A. Omara, "Task scheduling using pso algorithm in cloud computing environments," *International Journal of Grid and Distributed Computing*, vol. 8, no. 5, pp. 245–256, 2015.

[10] M. Ibrahim, S. Nabi, A. Baz, H. Alhakami, M. S. Raza, A. Hussain, K. Salah, and K. Djemame, "An in-depth empirical investigation of state-of-the-art scheduling approaches for cloud computing," *IEEE Access*, vol. 8, pp. 128 282–128 294, 2020.

[11] X. Huang, C. Li, H. Chen, and D. An, "Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies," *Cluster Computing*, vol. 23, pp. 1137–1147, 2020.

[12] Z. Wu, Z. Ni, L. Gu, and X. Liu, "A revised discrete particle swarm optimization for cloud workflow scheduling," in *2010 international conference on computational intelligence and security*. IEEE, 2010, pp. 184–188.

[13] Y. Feng, Y.-M. Yao, and A.-X. Wang, "Comparing with chaotic inertia weights in particle swarm optimization," in *2007 international conference on machine learning and cybernetics*, vol. 1. IEEE, 2007, pp. 329–333.

[14] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*, vol. 1. IEEE, 2000, pp. 84–88.

[15] Imran, Z. Ghaffar, A. Alshahrani, M. Fayaz, A. M. Alghamdi, and J. Gwak, "A topical review on machine learning, software defined networking, internet of things applications: Research limitations and challenges," *Electronics*, vol. 10, no. 8, p. 880, 2021.

[16] O. Cheikhrouhou, R. Mahmud, R. Zouari, M. Ibrahim, A. Zaguia, and T. N. Gia, "One-dimensional cnn approach for ecg arrhythmia analysis in fog-cloud environments," *IEEE Access*, vol. 9, pp. 103 513–103 523, 2021.

Appendix

```
#include<bits/stdc++.h>
using namespace std;

// Function to generate a random number between min and max
double getRandomNumber(double min, double max)
{
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> dis(min, max);
    return dis(gen);
}

// Structure to represent a task
struct Task
{
    int taskId;
    double length;
};
bool operator<(Task a, Task b)
{
    return a.taskId<b.taskId;
}

// Structure to represent a VM
struct VM
{
    int vmId;
    double processingCapacity;
};
```

```
vector<Task> taskList = {{0, 10.0}, {1, 5.0}, {2, 15.0},
 {3, 14}};
vector<VM> vmList = {{0, 7.0}, {1, 10.0}, {2, 8.0}};

// Function to evaluate the fitness of a particle's mapping.
// Fitness value is maximum Ready time.
pair<double, double> evaluateFitness(const map<int, double>
 &vRTMap)
{
    double fitness = 0.0;
    double minimum = 1e18;
    for (const auto& entry : vRTMap)
    {
        fitness = max(fitness, entry.second);
        minimum = min(minimum, entry.second);
    }
    return {fitness, minimum};
}

// Function to calculate the execution time
// of a task on a VM
double getExecutionTime(const Task& task, const VM& vm)
{
    return
    taskList[task.taskId].length / vm.processingCapacity;
}

// Function to update the readiness time of a VM
// after scheduling a task
void updateVmReadinessTime(map<int, double>& vRTMap,
 const VM& vm, const Task& task, double execTime)
{
    vRTMap[vm.vmId] += execTime;
```

```
}

// Function to update the task-to-VM mapping
// after scheduling a task
void updateTaskVmMapping(map<int, map<Task, VM>>& pMap,
 int p, int position, const Task& task, const VM& vm)
{
    pMap[p][task] = vm;
}

// Function to initialize the particles and their mappings.
map<int, map<Task, VM> > initializeParticles(int taskCount,
 int vmCount,int noParticles,
map<int, pair<double, double> >& pbMap, map<int,
 map<Task, VM>>& pMap, map<int, map<Task, VM>>& gbFMap,
 map<int,double>& vRTMap, const vector<VM>& vmList,
 pair<double, double> &gBest)
{
    map<int, map<Task, VM>> particles;
    for (int p = 0; p < noParticles; ++p)
    {
        map<Task, VM> particle;
        vRTMap.clear();
        for (int c = 0; c < taskCount; ++c)
        {
            int pos = getRandomNumber(0, vmCount);
            Task task;
            task.taskId = c;
            VM vm = vmList[pos];
            particle[task] = vm;
            double execTime = getExecutionTime(task, vm);
            updateVmReadinessTime(vRTMap, vm,
             task, execTime);
            updateTaskVmMapping(pMap, p, pos, task, vm);
```

```
            }
            pair<double,double> pBestValue =
             evaluateFitness(vRTMap);
            pbMap[p] = pBestValue;
            particles[p] = particle;
            pMap[p] = particle;
            if(pBestValue.first < gBest.first)
            {
                gBest = pBestValue;
                gbFMap[0] = particle;
            }
        }
        return particles;
}

void out(int ct, map<int, map<Task, VM>>& gbFMap,
 map<int, double> &vRTMap)
{
    cout <<ct<< " Global Best Mapping: " << endl;
    for (const auto& entry : gbFMap[0])
    {
        cout << "Task " << entry.first.taskId << " -> VM "
         << entry.second.vmId << " "
         <<vRTMap[entry.second.vmId] << endl;
    }
    cout <<endl;
}

int main()
{
    int taskCount = taskList.size();
    int vmCount = vmList.size();

    // Personal best value for particle. initially 0.
```

```
map<int, pair<double, double> > pbMap;
// Virtual machine id ready time
map<int, double> vRTMap;
// Mapping for every tast for every particle.
map<int, map<Task, VM>> pMap;
// Global best function map.
map<int, map<Task, VM>> gbFMap;
int itr = 1;
int maxItr = 1000;
int noParticles = 5;
int Ps = 1;
pair<double,double> gBest={1e18, 0.0};
map<int, map<Task, VM>> particles =
 initializeParticles(taskCount, vmCount, noParticles,
 pbMap, pMap, gbFMap, vRTMap, vmList, gBest);

double w1 = 0.9;
double w2 = 0.4;
double c1 = 2.0;
double c2 = 1.49455;

while (itr <= maxItr)
{
    double w = ((w1 - w2) / Ps) +
     ((maxItr - itr) / maxItr) * (w1 - (w1 - w2)) / Ps;
    int ss = 0;

    for (int p = 0; p < noParticles; ++p)
    {
        map<Task, VM>& particle = particles[p];
        vRTMap.clear();
        pMap.clear();

        for (int c = 0; c < taskCount; ++c)
```

```
{
    double r1 = getRandomNumber(0, 1);
    double r2 = getRandomNumber(0, 1);
    Task task = taskList[c];
    double &v=particle[task].processingCapacity;
    int pos = particle[task].vmId;

    v = (w * v) + (c1 * r1 *
     (pbMap[p].first - pos)) + (c2 * r2 *
     (gbFMap[0][task].processingCapacity-pos));

    pos = pos + (int)v;
    if (pos >= vmCount || pos < 0)
    {
       pos = getRandomNumber(0, vmCount);
    }

    VM vm = vmList[pos];
    double execTime = getExecutionTime(task,vm);
    updateVmReadinessTime(vRTMap, vm,
     task, execTime);
    updateTaskVmMapping(pMap, p, pos, task, vm);
}

pair<double, double> pBestValue =
 evaluateFitness(vRTMap);
if (pBestValue < pbMap[p])
{
    pbMap[p] = pBestValue;
    ss++;
}
if(pBestValue.first < gBest.first ||
 (abs(pBestValue.first-gBest.first)<=1e9 &&
 pBestValue.second > gBest.second))
```

```
            {
                gbFMap[0] = pMap[p];
                gBest = pBestValue;
            }
        }
        Ps = ss/noParticles;
        if(Ps <= 0)   Ps = 1;
        itr++;
    }

    // Print the global best task-to-VM mapping
    cout << "Global Best Mapping: " << endl;
    for (const auto& entry : gbFMap[0])
    {
        cout << "Task " << entry.first.taskId << " -> VM "
         << entry.second.vmId << endl;
    }

    return 0;
}
```