

Synergizing Genetic Algorithms And Simulated Annealing For Cloud Load Balancing

Ali Malik Hammood¹, Ehsan Shoja^{2,*}, Parviz Rashidi khazaei³

^{1,2} Department of electrical and computer engineering, Urmia University, Urmia, Iran.

³ Department of Information Technology and Computer Engineering, Urmia University of Technology, Urmia, Iran

*Corresponding author: Ehsan Shoja

Abstract

The growing demand of cloud computing services using large data centers usually results in imbalances, greater consumption of energy, and escalated operating costs in cloud systems. The issue of load balancing and energy efficiency in clouds is a major problem. Efficient load balancing depends upon efficient resource allocation and scheduling of tasks. There are a number of strategies that can be used to address this problem and reduce the use of energy in cloud systems. Use of collective intelligence algorithms and metaheuristic strategy is one of the vehicles. This paper has shown that the genetic algorithm can be improved by the use of the simulated annealing technique which can have a profound effect in the reduction of the total time needed to provide the applications with the resources. This causes better performance of the system, better load balancing, lesser energy utilization and less operation costs. The results of the simulation show that the proposed strategy is better than the baseline load balancing algorithm of virtual machines by more than 60 percent with better energy efficiency because of the better resource allocation.

Keywords: cloud computing, task scheduling, load balancing, genetic algorithm, simulated annealing algorithm.

1- Introduction

Nowadays, intelligent environments with communication devices are related to numerous communication devices spread globally. Data collection in different areas like medical, transportation, smart cities among others are collected in intelligent environments. Such volume of data is transferred to cloud computing facilities. The time taken for data transmission and processing depends on various factors and delays can compromise the value of the data. For instance, significant delays in healthcare monitoring systems could lead to patient fatalities, or in transportation systems, they could result in accidents. For this purpose, Cisco has proposed another model for data processing called fog computing. Fog computing involves the collection, processing, and storage of data locally by being placed between the cloud and smart devices, sending data to the cloud only when necessary. Fog computing is physically nearer to the cloud and devices thus allowing quicker processing and lowers the network traffic. Fog computing is also less energy consumptive than centralized cloud computing systems which have many nodes. The fog networks have hierarchical structure consisting of devices, edge tools and the cloud. The lowest level of the network architecture is the sensors who relay the obtained information to higher levels. Control and environmental change is done by launch devices. The data transmission, processing in edge tools, subdivision of programs into modules and allocation of resources to execute them are directives in the fog. Data collection and storage Small data centers are used in programs, which are later processed and analyzed in future. Workload balancing is one of the major challenges in the system execution management that has been researched extensively. Load balancing simply refers to the proper allocation of activities to the cloud or the fog system servers to satisfy some performance criteria. The load balancing algorithms are aimed at optimization of such performance statistics as execution time, processing costs, transmission costs,

service quality and others. Load balancing in fog environment In fog environments, load balancing techniques will be used to assign each task to a available processing resource in a way that does not contravene the rules of workflow that govern them. Such rules contain task dependencies and service quality parameters (cost and task completion deadlines) which are normally spelt in service level agreements between the customer and the service provider. As is clear, the different environments such as the mog computing do have different service level agreements and thus, allocation and load balancing processes are more complicated. The intensive development of cloud computing has increased the necessity of effective tasks scheduling and load balancing in particular with contemporary applications producing ever more heterogeneous and time-sensitive workloads. Recent efforts have revealed that the quality of scheduling has a direct influence on the responsiveness of the system, resource utilization and the cost of operation. Indicatively, Chiang et al. showed that better task scheduling can greatly improve the load balancing and execution efficiency in the cloud environments [1]. Scheduling of workflow under deadline conditions is also investigated with references to the heuristic enhanced genetic methods which suggest that time consideration optimization is vital [2]. On the infrastructure level, profitable virtual machine (VM) placement has been found to be a major performance factor of data-center where poor placement can result in extreme fragmentation of resources [3]. The effects of server consolidation also affect system behavior of energy, and there are frameworks that have been suggested that are migration conscious in order to minimize overhead without compromising the performance guarantees [4]. Similarly, dynamic load-balancing algorithms have been developed to spread workloads among VMs in a more even manner, which has shown imbalance and execution delay reduction [5]. The initial study of constraint-based packing also offered the background knowledge regarding the ways of arranging tasks to maximize capacity that is limited [5]. VM placement through SLA is central in avoiding performance violations, since dynamic placement mechanisms have been found to be effective in alleviating SLA violation when the workload is changed [6]. Mathematical programming has also been used to make decisions of consolidation in order to guarantee systematic saving of resources [7] and performance and utilization have been balanced with heuristics like ant-colony-based placement, which has been successful [8].

In reference [9], Loukopoulos et al. improved the genetic algorithms with list scheduling methods of independent task scheduling in distributed systems. Their work showed the possibility of hybrid approaches to enhance the scheduling efficiency. In [10], Esnault et al. proposed an energy aware distributed ant colony based algorithm to virtual machine consolidation of IaaS clouds, which focuses on the allocation of resources in a more energy efficient manner. In [11], Abirami and Ramanathan suggested a linear scheduling approach in resource allocation and minimized response times and maximized system performance. Wu et al. have investigated the relevance of SaaS-providers resource allocation strategies based on SLA in [12], and demonstrated the significance of SLA compliance in cloud resource management. Banerjee et al. in [13] introduced a framework of service-based resource selection, where service parameters are important in maximizing resource allocation. In reference [14], Akana et al. suggested a distributed decision analysis model to optimize the allocation of resources in a dynamic environment and present a PROMethe-based model to improve the process of decision-making on a cloud platform. Xiao et al. in [15] emphasized on the dynamic allocation of resources that are allocated with the help of virtual machines and offer techniques to adjust the resource consumption with variable demands. B and colleagues in [16] discussed automated negotiation strategies to dynamically allocate resources, with the focus on market-based

strategies. In reference [17], Kim et al. suggested a multi-agent design model of virtual resource configuration that presents agent-based strategies in managing resources. Zhou et al. presented in [18] an agent-based service and resource management scheme in a wireless cloud environment that broadens the scope of resource optimization to mobile and wireless environments. In [19], Garcia and Sim talked about the self-organizing agents to service composition with autonomous strategies on cloud service optimization. In [20], Manvi et al. invented an agent-based resource allocation system model of computational grids and provided the efficiency of agent-based approach in a distributed setting. Jung and Sim in [21] developed adaptive agents-based resource allocation by concentrating on real-time flexibility to the change in workload. Yee-Ming and Hsin-mie (in [22]) considered the autonomous agents to market-based resource allocation, with the focus on adaptive mechanisms of the resource management.

Rui et al. proposed practices of scaling cloud application resources lightweight in [23], with the main emphasis on efficiency and scalability. Giridas and Shajin, in [24], suggested an optimal technique of resource allocation to green cloud computing with regard to environmental sustainability. Ke et al. established a time-cost scheduling algorithm of workflow in cost-constrained cloud settings in [25] to determine trade-offs between performance and cost. Kardem et al. used a hybrid simulated annealing algorithm to minimize the graph profiles in [26] demonstrating its usefulness in optimization problems. The load balancing introduced in [27] is a heuristic task deployment method based on Bayes theorem and focused on optimization based on prediction. There has been a lot of interest in hybrid metaheuristic. Recent hybrid PSO-SA algorithm with built-in load-balancing is also reported to perform highly in the fog-cloud environment [28].

2- Problem statement

The aim of this research is to achieve load balancing on servers and optimize costs in cloud environments by utilizing a scheduling method for tasks and resource allocation based on optimized soft computing algorithms. To this end, we intend to initially test an enhanced genetic algorithm using Simulated Annealing (SA). The rationale behind this idea is that for solving highly complex problems, a commonly used and powerful method is the Genetic Algorithm (GA). Nevertheless, it is worth mentioning that this algorithm is characterized by two significant limitations namely, its processing speed is low and although it has outstanding general search abilities, it is weak in local search. We therefore resort to one of the best local search algorithms. Based on studies in this area, the SA algorithm seems to be a suitable solution.

Put differently, our intention is to define two-dimensional chromosomes that consist of tasks and resources as a novel idea in this method. In this setup, one row of chromosome genes represents tasks and the other row represents resources. To reach an optimal solution, we define a suitable upper limit and whenever the genetic algorithm reaches an optimum value, we switch to using the SA algorithm for a more precise value. Due to the NP-Hard nature of the problem and a review of previous methods in this domain, it appears that by introducing a new approach based on combining the GA and SA algorithms, we will be able to conduct a thorough search in the problem space and enhance results in this area.

The initial population consists of a number of solutions that are randomly generated in the standard genetic algorithm. In this approach, we aim to create the initial population purposefully, so that not only does it accelerate the discovery of solutions, but also the resulting answers are closer to the optimal solution compared to the conventional method. The workflow is as follows: for mapping a task to a virtual machine (VM), the conditions for this operation are assessed according to a specific formula. When this mapping is selected as a gene from the initial solution's chromosome, the same process is

carried out for other tasks until a complete chromosome is formed as one of the generated solutions. Each solution, or resource allocation model, is represented as a matrix of size $2 \times n$, where n denotes the number of tasks, and each row is considered as a chromosome within the genetic algorithm. In simpler terms, the first row of this matrix represents the tasks, while the other row indicates the resources. As observed, each gene denotes a task and its corresponding suitable resource. The best resource, in terms of the execution time for the task, is considered. If a resource is deemed not eligible, the next resource (the second-best resource) is tested, and this process continues for all genes. Subsequently, the virtual resource list of the chromosome is updated based on the remaining capacity of each resource in relation to the workload of the tasks. Due to the number of resources being less than the number of requests, each resource must process more than one request. So far, we have been able to effectively solve the problem using the genetic algorithm. However, due to the genetic algorithm's weakness in local search, the best possible solution is unlikely to be achieved. To optimize the presented model, we intend to utilize the Simulated Annealing (SA) algorithm at the end of the aforementioned process. In other words, by leveraging the high power of the SA algorithm in local search, we aim to find the best possible solution. It appears that by enhancing the genetic algorithm with the gradual algorithm, we can improve task scheduling and resource allocation in the realm of virtual machines and cloud environments. This way, we can offer a suitable and acceptable state of load balance and naturally minimize energy consumption to the best extent possible.

3- Proposed method

3.1 Introduction

Cloud computing is an emerging environment for high-performance, large-scale computing, consisting of a heterogeneous set of independent systems and flexible computational architectures. To improve the overall performance of cloud computing, with a completion time constraint, a job scheduling model has been established to reduce the energy consumption of cloud computing systems and enhance the profit for service providers. For the job scheduling model and load balancing, a solution method based on the combination of genetic algorithms and simulated annealing (GA-SA) has been designed. The research focuses on separation rules (encryption), crossover operations, selection operations, and the use of Pareto sorting methods (one of the SaaS service techniques). In fact, considering the topic that genetic algorithms, despite their high power in global search, are weak in local search operations, the aim is to eliminate this weakness of genetic algorithms using the properties of the simulated annealing algorithm and improve the efficiency of the problem.

Cloud computing, a long-standing dream, was created to be viewed as "computing as a tool" and as a new model for distributed computing at a large economic scale. It is emerging as a highly scalable, heterogeneous, virtual, and configurable system with high computational resources (such as network, storage, computing units, applications, and data) that can be rapidly deployed. From an economic perspective, customers only pay for what they actually use and need. Resources are available via the internet, and data from the cloud can be accessed at any time and from any location. However, data centers are facing significant and growing energy challenges, with a single data center typically consuming enough energy to power up to 25,000 households. As such, cloud computing system energy usage is critical. Different tasks required with time to utilize resources vary. The basic design of the job scheduling system that efficiently assigns resources to the necessary requests within the limits of the Service Level Agreement (SLAs) is geared towards producing high performance in cloud computing and enhancing resource loading balancing, safeguarding, dependability and consuming less energy and total resource

utilization of the system. Nonetheless, it is confronted with a big problem of creating an effective scheduling algorithm and deploying it in a cloud-computing setting.

To reduce energy consumption, Pinheiro and colleagues proposed a model to minimize energy usage in a heterogeneous clustering system, with multi-service computational nodes serving websites over a period. They monitored resource load and made decisions regarding the switching of nodes on/off to minimize overall power consumption. In another study, Raghavendra and colleagues combined and explored five different policies in the field of energy and power management, theoretically controlling the system. However, the system was unable to practically control the variable programs in alignment with SLAs. Li et al. proposed two algorithms based on a pricing model, using processor sharing to balance profit and resource usage. Additionally, Gang and colleagues proposed linear genetic algorithm programming, aiming to create the best plan for network tools by minimizing the combined costs for all users in a coordinated manner. All the methods mentioned above aimed at achieving higher efficiency or reducing energy consumption in their studies; however, no relationship exists between them. To overcome the limitations of the aforementioned algorithms, this research initially focuses on creating a macroscopic planning model that identifies and makes decisions about the components of cloud computing. This model considers the requirements of various tasks and the conditions of computational infrastructures. Subsequently, a job scheduling algorithm is proposed based on the combination of genetic algorithms and simulated annealing (GA-SA) with respect to energy consumption and the profits obtained by service providers. A dynamic selection tool is presented, which offers a recommended planning scheme to the users according to the current demands of the duty.

3.2 Job Scheduling Model for Load Balancing Improvement

The heterogeneity in resource demands is common in cloud computing where service requests may include such demands. Consequently, there can be services that use more processor power and services that use more input/output. The allocation of the cloud resources must be done in a manner that does not only assure optimal level of Quality of Service (QoS) as stipulated in the Service Level Agreements (SLAs) but also reduces the amount of energy used and enhances the energy efficiency of service providers to optimize the total amount of energy used in each step.

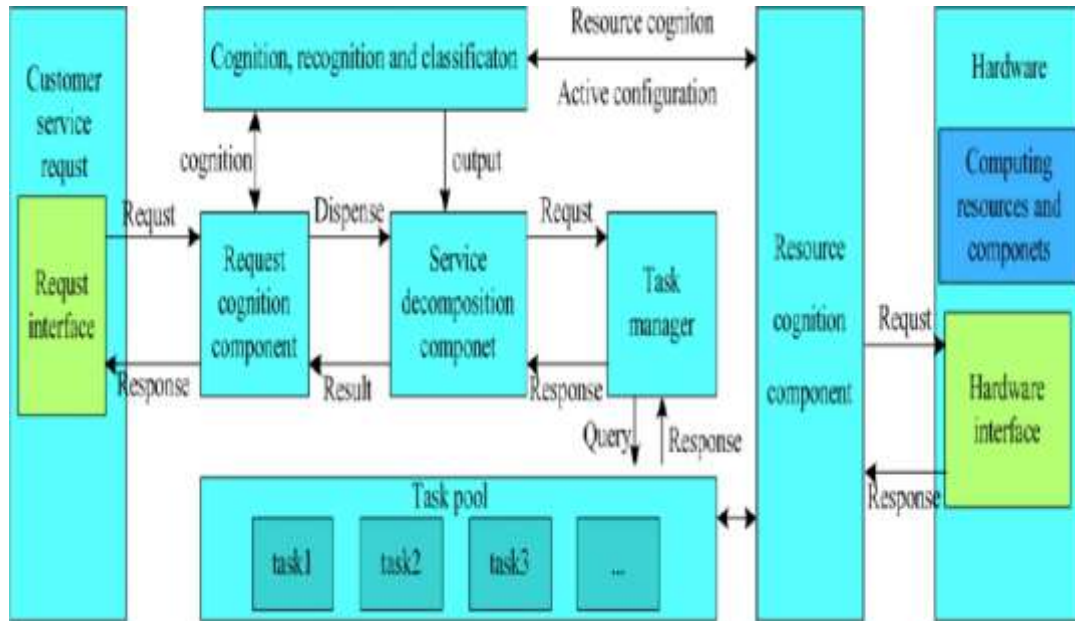
3.3 Architecture Model

As shown in Fig. 1, the architecture model works through developing a scheduling system. The finer workings of the core elements are presented as follows: Request Recognition component recognizes in detail the concrete needs of the various business functions, which might involve computations, storage, and communication requirements to process, entry laws, concurrency guidelines, security, privacy needs, service quality, among others. The (Service Decomposition Component) decomposes services into various (granularities), which have different processor configurations. The second one is (Task Manager) that optimizes the use of resources required by each granularity with effective solutions, analysis and mapping to the necessary processors.

Besides, it is also the duty of the Task Manager to manage the status of the tasks (start, stop, cancel, etc.), assign tasks to tasks as per the schedule, and allocate resources to the tasks so that the tasks are assigned the right resources. This is the component that aids the scheduling algorithm.

Resource cognition component 43 is also important in the management of resources by observing resource performance, dynamically optimization of the scheduling strategy and error reporting.

Fig. 1: Functionality of the Architecture Model [34].



3.4 Problem Formulation

Our model will be a cloud application system whereby there is a set of tasks or jobs that are involved with complex computational work done using available resources within the cloud. $A = (a_1, a_2, \dots, a_n)$ is a set of applications that are being entered in a period. In the process of scheduling, the requester submits service request of the software in the form of $a_i (1 \leq i \leq n)$, the resources required, (T_i, N_i, d_i) , where T_i denotes the reservation time of the software on Virtual Machines (VMs) which is the computational elements in the cloud and uses virtualization technology, N_i represents the number of VMs required in running a_i , d_i is the deadline for the program, after which the program fails. The problem to be solved by this algorithm is how to schedule these M applications to N clouds, such that the objective function is optimized. Here, N refers to clouds distributed in various geographic regions around the world, which are typically heterogeneous. However, within a VM, all clouds are considered homogeneous using virtualization techniques.

3.5 Objective Function

Suppose that the program a_i is scheduled and executed on cloud C_j , where P_j is the processing power of each VM in cloud C_j . The energy consumption of its execution is given by the following equation:

$$E_{ij} = P_j n_i t_i \quad (1)$$

And its efficiency will be:

$$R_{ij} = n_i * t_i * p_r - co_{ij} \quad (2)$$

Where p_r is the unit service cost for a_i , and co_{ij} is the cost to the provider for executing a_i . At the end of the execution, the following conditions should hold:

$$\min E = \min \left(\sum_{i=1}^1 \sum_{j=1}^N E_{ij} \right) \quad (3)$$

$$\min R = \max \left(\sum_{i=1}^M \sum_{j=1}^N R_{ij} \right) \quad (4)$$

Here, E and R represent the total energy consumption and the profit obtained from M applications across N clouds, respectively.

3.6 GA-SA Scheduling Algorithm

Initial Population Initialization:

The initialization of the population will have a significant impact on the quality of future generations and is a crucial step in the entire algorithm. In this research, this stage is carried out by combining random initialization methods and the SA algorithm. Using a greedy initialization method in scheduling algorithms, by rejecting programs with specific deadlines, may cause the entire algorithm to encounter failure. However, this type of initialization ensures diversity in the initial population and prevents bias in the search and selection of programs.

Encryption Rules:

For each scheduling, we will have a two-dimensional matrix of size 2 by M, where M is the length of our chromosomes. In this matrix, the first row contains the program numbers that are requested to be executed on the cloud, and the second row contains the clouds that receive the service requests. The rule followed in this matrix is that each program can only be executed on one cloud, but each cloud can execute multiple programs. The figure below shows how the chromosomes of the genetic algorithm are fully encoded.

Fig. 2: Chromosome Structure as a Two-Column Matrix of Programs and Servers Defined in the Cloud Environment.

Number of programs	3	8	2	...	...	...	16	9
Number of programs	4	2	2	...	...	...	3	1

As shown in Fig. 2, in the proposed method, each chromosome is defined as a sequence of program executions across different hosts. In fact, the purpose of defining the chromosomes in this manner is to find the optimal sequence for executing programs by the servers defined in the cloud environment.

After defining the chromosomes, the evaluation of the chromosomes is carried out using the objective functions defined in Section 3-5. In simple terms, the suitability of the chromosomes is determined using two criteria: the amount of energy consumed and the level of efficiency. Naturally, a more suitable chromosome is one that consumes less energy and exhibits higher efficiency.

Genetic Algorithm:

The genetic algorithm is an exploratory search method, similar to the natural evolutionary process, based on a population that creates its population through candidate solution selection. This method is typically used for optimization and solving problems to produce useful solutions. In the evolutionary process, modification occurs through operators applied to each individual. Each chromosome represents a scheduling result, and an evaluation operator is used to assess the offspring.

Independent Evaluation:

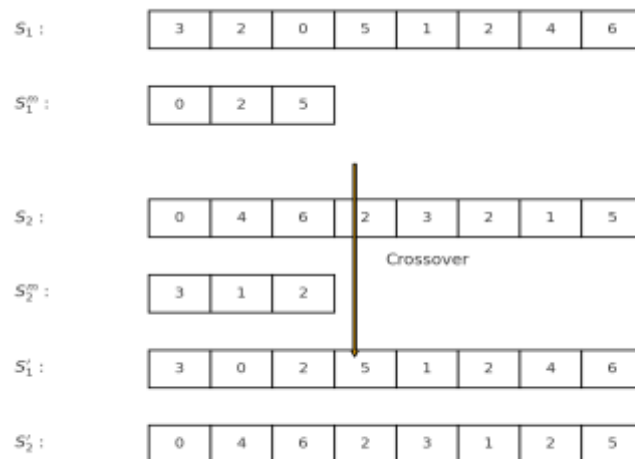
In this paper, the focus is on energy consumption and the benefits obtained from service providers. Only the highest-ranked solution is retained after evaluating the fitness function in the Pareto archive, which includes non-dominated solutions, for generating subsequent generations.

Selection Operation:

Elitism and Crowding: The selected operation is an operator for competing with K individuals, utilizing two strategies. The elitism strategy uses individuals from the Pareto archive and selects the best individuals from previous generations, ensuring that dominance over subsequent generations is avoided. The convergence of the evolutionary process with the crowding strategy will be a longer strategy, utilizing the crowding distance at a point to estimate the density around it and eliminate solutions that are far from this point. The crowding distance is defined as the distance in a rectangular form determined by its left and right neighbors, and if no neighbors exist, it is defined as infinite.

Crossover Operation:

The crossover operator uses the rows 1S and 2S to generate two new individuals. It randomly selects two numbers such that these two numbers satisfy the condition $i < j < N$. Then, the algorithm selects the i -th element of individual 1S to the j -th element and stores it in a temporary location. Similarly, for individual S_2 , the elements are selected in the same manner and placed in the first available spot. The values taken from the temporary location are then used for replacement in the second individual. The entire operation is illustrated in the diagram below.

Fig. 3: Crossover.**Mutation Operation:**

Two tasks are selected randomly from a given case, and their positions are swapped. The following algorithm was proposed by (Russell). You can find it in many books, where it is introduced as one of the best local search algorithms:

Algorithm pseudo-code:

```

FUNCTION SIMULATED_ ANNEALING (problem, schedule)
    Return a solution state
    Input:
        problem, a problem
        Schedule, a mapping from time to temperature
    Local variables:
        Current, a node
        Next, a node
    T, a "temperature" controlling the probability of down ward steps
    Current = make_node (intial_state[problem])
    For t = 1 to ∞ do
        T = schedule [t]
        If T = 0 then
            Return current
        Next = a randomly selected success of current
        ΔE = VALUE [Next] - VALUE [current]
        If ΔE > 0 then
            Current = Next
        Else
            Current = Next only with probability  $\exp\left(\frac{-\Delta E}{T}\right)$ 

```

Cooling Schedule:

The components of the cooling schedule include:

Initial temperature

Final temperature

rate of temperature reduction at each iteration.

Repeat at any temperature

Initial Temperature:

The initial temperature must be sufficiently high to allow movement to neighboring states. If the initial temperature is too high, the search may move to any neighbor, and the search will turn into a random search until the temperature cools down sufficiently. Determining the appropriate initial temperature is a structured process and does not have a specific method for all problems. If we know the maximum distance (cost of various functions) among a neighborhood and other neighborhoods, we can use this information to calculate the initial temperature. (In other words, we can calculate the required energy.)

Final Temperature:

Typically, the temperature is allowed to decrease until it reaches zero. Additionally, a stopping criterion can be a suitably low temperature.

Temperature Reduction at Each Step:

It is generally possible to obtain the temperature reduction with a simple linear relationship (a geometric progression):

$$t_i = \alpha t_{i-1} \quad (5)$$

Experience shows that the value of α should be between 0.8 and 0.99 to achieve the best result and prevent the algorithm from becoming too lengthy.

Iterations at Each Temperature:

relationship used is as follows:

$$t = t(1 + \beta t) \quad (6)$$

β is a small appropriate value.

At lower temperatures, the number of iterations should be large, and at higher temperatures, the number of iterations can be small.

Cost Function:

The approach for solving a problem should include methods for measuring the quality of the solution. The cost function is typically complex and is represented by ΔE :

ΔE : Evaluation of the difference between the current solution and the neighboring solution.

3.7. Execution of the GA-SA Algorithm

As mentioned, using the Genetic Algorithm (GA) and the Simulated Annealing (SA) algorithm, each program is initially sorted based on the size of tasks and the execution time of each task. The sorting process is as follows: each program or service requestor can have multiple tasks or services, and these tasks require a certain amount of time to complete. For simplicity, we assume that the tasks of each program or service requestor have the same execution time, and then the value of this program for the requestor is determined by multiplying the number of tasks by the execution time of each task. This value is then returned. We repeat this operation for all tasks in the system. Afterward, the results are sorted from largest to smallest.

Next, using the SA algorithm, we retrieve the largest unexecuted tasks from the list, depending on the input, which determines how many programs can be executed simultaneously. Then, we assign a server to each program. This server executes part or all of the program. Using crossover and then mutation among the available programs, we implement the process. Afterward, we use the SA algorithm to select the larger remnants for the next generation and repeat the selection operation. This process is repeated for a desired number of iterations, for example, 100 iterations. After this period, we can separate and display the fully executed programs and the incomplete ones.

Our fitness function for task execution and selecting the next generation can be expressed as follows: completed tasks are discarded, and the remaining larger tasks from the next generation will be the input for our functions. The operations of crossover and mutation allow the currently executing programs to be included in the next generation's program set.

This is because several client service programs are selected for the next generation using crossover and mutation, while others are selected from the largest tasks of the next generation.

3.8. Optimization Explanation

In this algorithm, compared to previous methods, we have introduced two major changes. First, we did not use a greedy approach for the input data. In a way, we have broken this method and defined the priority order for entry and service, such that we will first serve programs that have the most tasks. These programs may not necessarily have the highest total time, but their number of tasks is greater than the rest. As a result, the largest program will not always be at the top, and the order will not necessarily be strictly ascending or descending.

Secondly, for this algorithm, like the round-robin method, a quantum has been considered. Based on the quantum value, the processor will execute as much of the task as possible. After the allocated time for that task is completed, the processor or service provider will stop executing it and give control to another program. It should be noted

that we have also used the round-robin algorithm with some modifications. The modification involves adding a quantum value after every 10 programs that are processed. However, it must be mentioned that this time must be less than the total time required to complete a service on a server.

4- Results and Analysis

4.1 Introduction

Based on the previously mentioned points and the explanations provided about the algorithm, we can conclude that, in general, we have reached a certain level of optimization. However, it cannot be claimed that this state will always be 100% accurate in all situations, as it depends on the conditions of the servers and the service provider and receiver systems.

Initially, with the change we made in sorting the input data, we have broken the rule that the largest or the greatest should always be placed first. This has allowed programs with shorter execution times to have the chance to be executed and reach their service goals. Secondly, our intention behind assigning execution slots was to avoid overloading a system or systems with a specific set of programs that involve extensive operations, and instead allow other programs to execute and reach their objectives. In general, our ultimate goal is to ensure that tasks are executed in a cloud environment within a shorter time frame, resulting in lower energy consumption for task execution.

The proposed method has been simulated to the best extent possible using the MATLAB programming language. General information about the scenario in question can be found in Table 1.

Table 1: Initial Simulation of the Scenario by the User

Item	Quantity
Number of programs	100
Task interval	2~5
Time interval for each task (milliseconds)	3~6

The "application numbers" refer to the number of service requests from our service receivers, while the "tasks" represent the number of services each application requires, and "time" specifies the execution time for each service. In this context, we have also generated server information, which has been created randomly. There are 10 servers, each displaying the processing power for each task execution, along with their power consumption and the associated cost. Below, the tables utilized in the program are presented to you.

Table 2: Table tbl for storing tasks.

Field name	Description
App	Request Service Number
Task	Specifies the number of tasks required by a requester
Time	Determines the duration of each task

Table 3. the "srvr" table, is used for storing server information.

Field name	Description
Id	Server Number
Calc	Server Calculation Rate at Any Given Time
Pwr	Energy Consumption
amount	Cost of Using Each Server

Additionally, the parameters related to the initialization of the genetic algorithm can be observed in Table 4.

Table 4 - Initialization Parameters of the Genetic Algorithm

40	Population Number
0.9	Crossover Probability (Pc)
0.05	Mutation Probability (Pm)
1200	Termination Condition (Iteration Number)

4.2. Experiments

Since this algorithm increases its time period cyclically, it is possible to achieve a situation in the future where the processor's allocation to a program decreases proportionally, allowing it to dedicate a longer duration to completing its tasks. Finally, we will observe and compare the two algorithms.

Table 5: The output of the optimized algorithm.

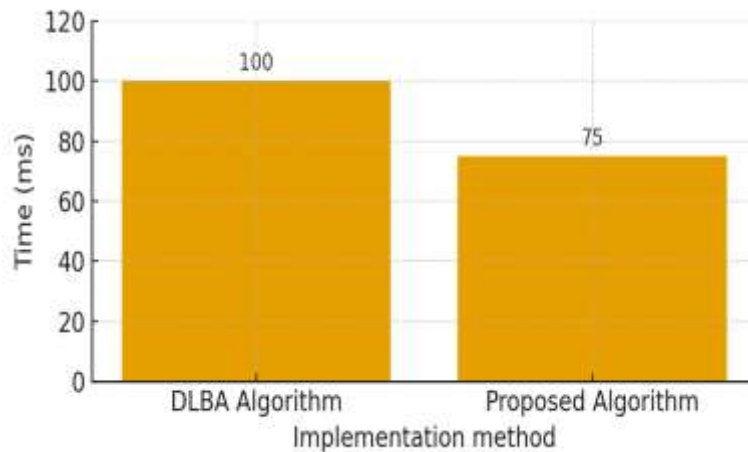
Item	Value
Number of rounds	100
Number of programs per round	20
Time slice (ms)	6
Mean execution time (ms)	75
Execution time standard deviation (ms)	±3
Total cost	503115
Number of unfinished tasks	2

Table 6. Display of DLB Algorithm Output

Item	Value
Number of rounds	100
Number of programs per round	20
Time slice (ms)	6
Mean execution time (ms)	100
Execution time standard deviation (ms)	±4
Total cost	504707
Number of unfinished tasks	3

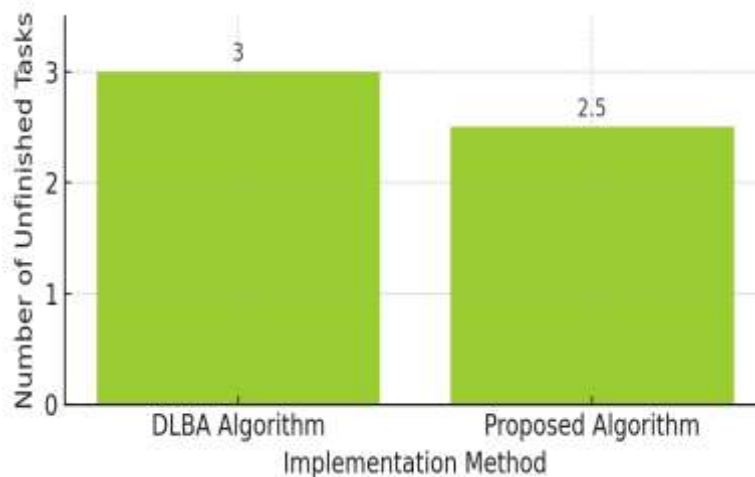
After defining the scenario, both algorithms are executed and their outputs are examined. The results of running the proposed algorithm can be observed in Table 4, and the results of running the baseline algorithm, named Dynamic Load Balancing Algorithm (DLBA), can be seen in Table 5.

Fig. 4: The time comparison graph between two algorithms, DLB and the proposed algorithm.



As depicted in Fig. 4, the execution time of the proposed algorithm is significantly lower compared to DLBA, which is attributed to the efficient local search operations. As a matter of fact, due to the character of the gradual cooling algorithm, the suggested algorithm will reach the optimal solution much faster than DLBA, which results in the decrease of the search time.

Fig. 5: A comparison chart of incompletes between the DLBA colony algorithm and the proposed algorithm.



The amount of remaining tasks within the system is also represented by Fig. 5, which in particular reveals that the amount of unfinished tasks within the system in the proposed method is significantly smaller than in the case of DLBA, which is an acceptable performance of the proposed approach.

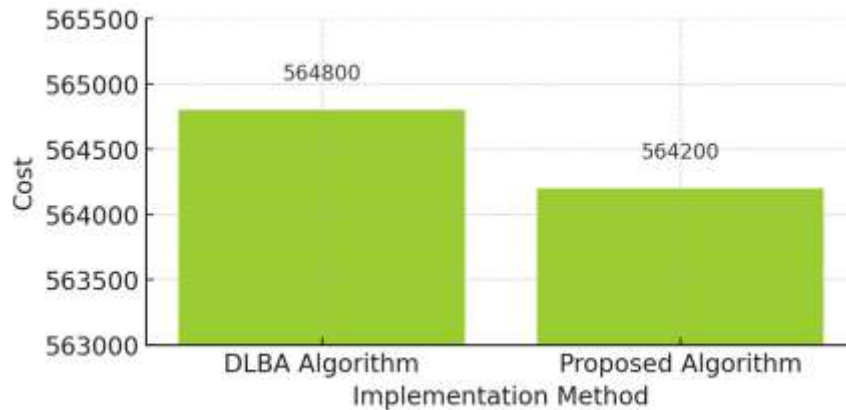
Fig. 6: A cost comparison between the DLBA algorithm and the proposed algorithm.

Fig. 6 indicates the cost levels of the two algorithms in question. As evident from the two bar charts within the figure, the cost of the proposed method is significantly lower than that of the DLBA method. Although the cost difference is not substantial, this discrepancy becomes more pronounced in more complex problem spaces.

In these charts and this comparative analysis, you observe the relative superiority of the new algorithm. Indeed, it is apparent that the proposed algorithm, which is based on a combination of genetic algorithms and simulated annealing, has succeeded in executing tasks in less time, with lower costs, and overall with reduced energy consumption. Subsequent experiments in this same area will be reviewed. In the previous comparison conducted for 100 programs, each program requiring between 2 to 5 tasks, and each task taking between 3 to 6 milliseconds.

In our new comparison, we consider this value for 500 service programs, each program with 2 to 5 tasks, and each task lasting between 6 to 12 milliseconds. Ultimately, we will observe the desired results. In the new comparison, we iterate 500 times, retaining 80 programs in each iteration.

Given the implemented changes, the outputs of both methods are observable in Tables 7 and 8.

Table 7: The output of the DLBA algorithm.

Item	Value
Number of rounds	100
Number of programs per round	500
Time slice (ms)	6-12
Mean execution time (ms)	600
Execution time standard deviation (ms)	±5
Total cost	564200
Number of unfinished tasks	2.5

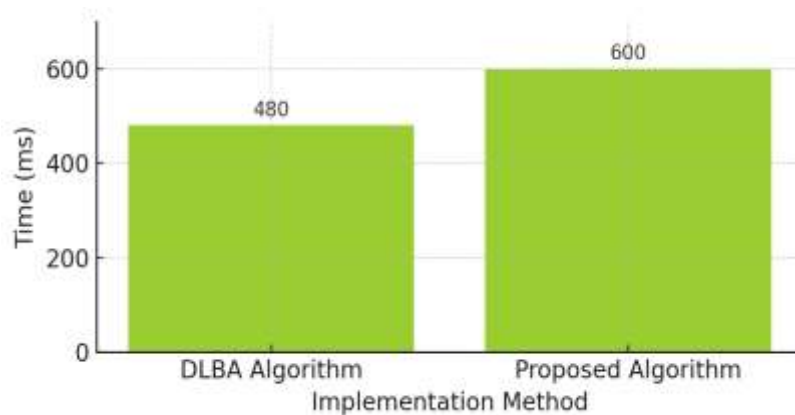
Table 8: Display of Proposed Algorithm Output

Item	Value
Number of rounds	100
Number of programs per round	500
Time slice (ms)	6-12

Mean execution time (ms)	480
Execution time standard deviation (ms)	± 6
Total cost	564800
Number of unfinished tasks	3

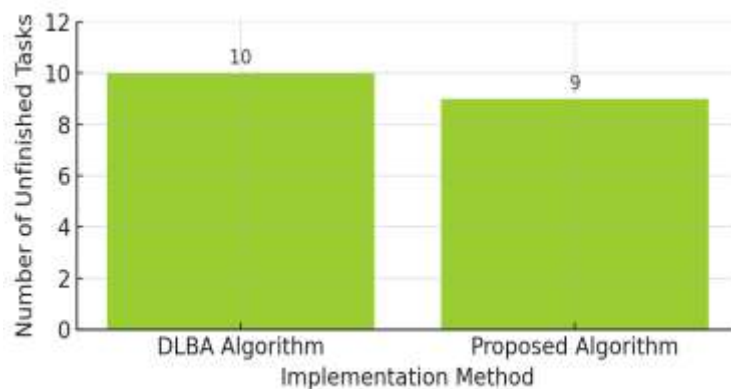
Based on Tables 4-7 and 4-8, we endeavor to proceed with an analysis of the obtained results. In Algorithm 2, we only changed the rotational period to 12, and you can observe the outcome. Essentially, with the increase in the number of tasks and programs, the time has also been extended from 6 to 12 to allocate less time for Context Switch. We will present a comparison between these two programs again in the following graphs.

Fig. 7: A comparative analysis of the time between the DLBA algorithm and the proposed algorithm.



Looking at Fig. 7, one can note that the given algorithm takes more time than DLBA to locate the required solution. Basically, it is arguable that the more complex the problem space, the more time would be required by the genetic algorithm to run a search to find a relevant solution than in the past. This problem causes the proposed algorithm to spend a significant part of time searching an optimal solution.

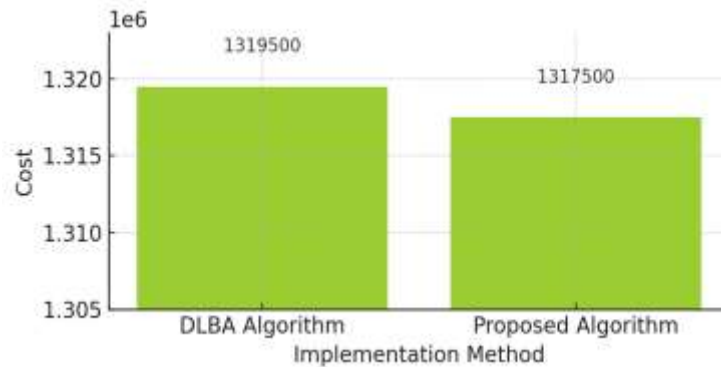
Fig. 8: Comparison of the number of incompletes between the DLBA algorithm and the proposed algorithm.



As it was observed in the chart presented in Fig. 8, the suggested algorithm failed to provide improvement over DLBA in the metrics of the execution time. Still, when

comparing Fig. 8 and evaluating how many tasks are still left to be carried out in the system, it can be seen that once again, the offered technique has performed better and it is offering a better result compared to DLBA.

Fig.9: A cost comparison between the DLBA algorithm and the proposed algorithm.



As mentioned, and illustrated in the charts and programs, it can be inferred that an optimized state of this algorithm is conceivable. However, achieving this optimal state universally across all scenarios is not feasible due to varying environmental and geographical conditions, especially considering that factors such as spatial distance between the service provider and receiver and the nature of their communication can significantly vary.

Here, we have compared ten additional instances with different timings to the initial algorithm, which are displayed in the graphs and tables below.

We executed the desired algorithm with 100 programs, each program containing 2 to 5 tasks, with each task running between 3 to 6 milliseconds. The output provided by this program is as follows.

Table 9: output results.

Number of rounds	Number of programs executed per round	Execution time in milliseconds	Uncompleted programs at the end of 100 loop times
100	20	80	2

However, the execution of the optimized program is conducted 10 times with different rendering times as follows. The number of cycles remains constant at 100 each time, with 20 programs executed in each cycle under the same previous conditions.

Table 10: the execution time of the optimized program with different rendering times.

Execution time of each program per round	Total execution time in 100 rounds milliseconds	Uncompleted programs
2	120	0
4	82	5

6	89	1
8	78	1
10	78	2
12	78	0
14	67	3
16	82	0
18	86	3
20	78	1

It can also be represented graphically as follows: the horizontal line is a representation of the main algorithm, which we will consider as the benchmark, while the other line illustrates the execution of our program at each point in time.

Fig. 10: The total time comparison of ten executions with different run times compared to the main algorithm.

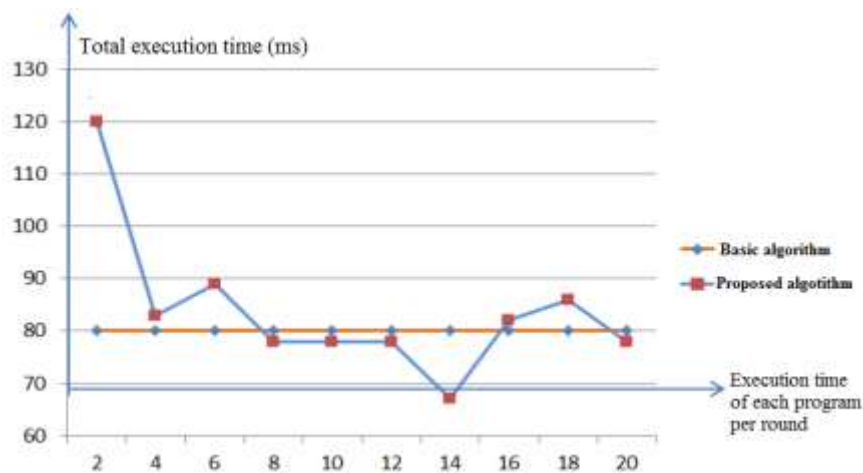
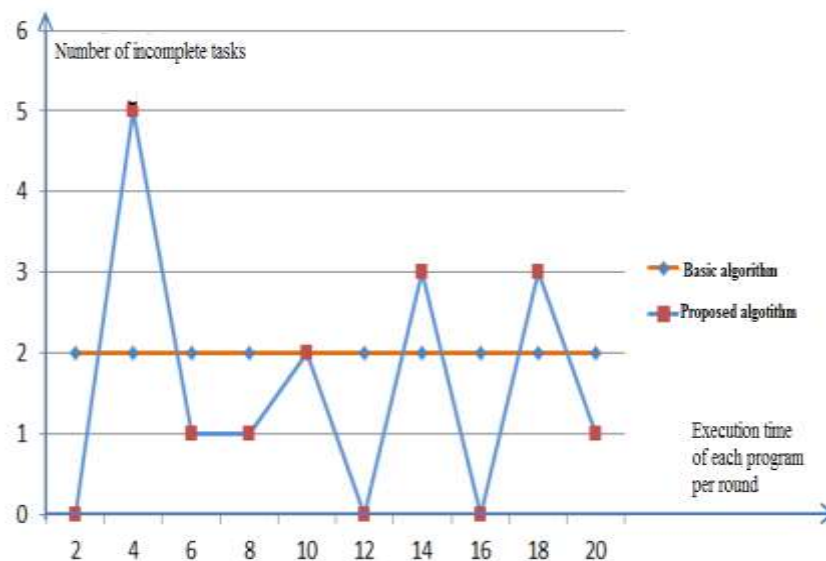


Fig. 11: Graph of total uncompleted programs comparing ten executions with different execution times with the basic algorithm



Regarding the graphs related to Figs. 10 and 11, it is necessary to note that in the baseline method of the article, namely the DLBA algorithm, we are not able to define the execution time of each program in each round. The structure of the desired algorithm is such that it services the defined programs in the cloud environment with constant values. However, in the proposed method, in different rounds, the execution times of programs are varied and naturally, both the total execution time and the number of unfinished tasks are also variables. Therefore, for the execution of 100 programs, each containing 2 to 5 tasks where each task runs between 3 to 6 milliseconds, the output of the DLBA algorithm remains constant while the output of the proposed method varies based on the defined times in different rounds.

In the comparison of time in the first graph, it can be stated that 5 out of 10 cases fall below the criterion line, with 2 cases aligning with the criterion, showing optimal results in approximately 50% of cases.

In the second graph, illustrating the number of incomplete programs and the remaining tasks, we observe that 6 cases fall below our criterion line, 2 cases align with the line, and only 1 case exceeds our criterion, indicating an optimization of around 60% and improved energy efficiency.

the two graphs, there are cases where the optimality not met, probably because of context switches that also consume time.

5- Conclusion

The cloud computing energy consumption is a technical matter to debate upon since it involves the approach of resource allocation and task scheduling, and therefore the issue of energy consumption in cloud computing is an essential issue in cloud computing. Hence, there is a lot of research on the same. Nonetheless, the way of distributing resources and scheduling activities to improve workload balance can be regarded as NP-hard. A notable problem with cloud computing is that the prioritization of resources that are allocated to tasks is virtually unattainable to users themselves. The adequate scheduling of various requests can allow utilizing a certain number of resources, which contributes to high levels of computational performance, optimal operational capacity, quality of services, and reduced energy usage. As such, it is evident that resource allocation and task management as well as security and reliability and customer trust are major security issues in cloud computing that directly affect energy consumption. Apparently, tasks and resource number in a cloud environment could be enormous; hence, the sequence of task execution and resource allocation has a great impact on the efficiency of the server. In a cloud environment, we have concurrent requests at any moment, each requiring one or more tasks to be executed, each of which demands specific resources. Various methods have been proposed in this regard. Therefore, optimizing energy consumption in servers and reducing costs pose important challenges in this area of study.

In the present study, we introduced a scheduling method in the cloud computing environment based on the standard genetic algorithm and the simulated annealing algorithm, aimed at minimizing energy consumption in the cloud. The proposed method was compared with the foundational algorithm of the research, namely the DLBA algorithm, demonstrating improved performance through reduced task completion time (achieving optimal solutions in a shorter period) as well as enhanced fitness of individuals and minimized energy consumption in the proposed algorithm.

In fact, our research proposes an optimization method for task scheduling in cloud computing and formulates a model for scheduling tasks to minimize problem duration and energy consumption. This model was solved by enhancing the standard genetic algorithm using the simulated annealing algorithm.

REFERENCES

1. Chiang, Mao-Lun, Hui-Ching Hsieh, Yu-Huei Cheng, Wei-Ling Lin, and Bo-Hao Zeng. "Improvement of tasks scheduling algorithm based on load balancing candidate method under cloud computing environment." *Expert Systems with Applications* 212 (2023): 118714.
2. Amandeep Verma, Sakshi Kaushal, "Deadline constraint heuristic-based genetic algorithm for workflow scheduling in cloud", *International Journal of Grid and Utility Computing* Volume 5 Issue 2, March 2014.
3. Shi. W and Hong. B, Towards Profitable Virtual Machine Placement in the Data Center. IEEE, International Conference on Utility and Cloud Computing, 138-145, 2011.
4. Ferreto. T, Netto. M, Calheiros. R and De Rose. R, Server consolidation with migration control for virtualized data centers. Elsevier, *Future Generation Computer Systems* 27, 1028-10301, 2011.
5. Kumar. M and Sharma S. C, Dynamic load balancing algorithm for balancing the workload among virtual machine in cloud computing. *International Conference on Advances in Computing & Communications*. (115): 322-329. 2017.
6. Shaw. P, A constraint for bin packing. *Principles and Practice of Constraint Programming_CP*. Lecture Notes in Computer Science Volume 3258, May, 648-662, 2004.
7. Bobroff. N, Kochut. A and Beaty. K.A, Dynamic placement of virtual machines for managing sla violations. the 10th IFIP/IEEE International Symposium on Integrated Network Management, IM'07, 119-128, 2007.
8. Speitkamp. B and Bichler. M. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE Transactions on Services Computing*, 266-278, 2010.
9. Y. Gao, H. Guan, Z. Qi, Y. Hou and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing", *Journal of Computer and System Sciences*, Vol 79, pp 1230-1242, 2013.
10. Loukopoulos T., Lampsas P., Sigalas P., "Improved Genetic Algorithms and List Scheduling Techniques for Independent Task Scheduling in Distributed Systems", IEEE, Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies, Adelaide, SA, pp. 67-74, 2007.
11. Esnault. A, Feller. E and Morin. C, Energy-Aware Distributed Ant Colony Based Virtual Machine Consolidation in IaaS Clouds, dumas-00725215, version 1 - 24 Aug, 4-6, 2012.
12. Abirami SP, Ramanathan S, "Linear Scheduling Strategy for Resource Allocation in Cloud Environment", *International Journal of Cloud Computing: Services and Architecture*, Volume 2, Number 1, pp. 9-17, 2012.
13. Wu L, Garg SK, Buyya R, "SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments", Published in 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp 195-204, 2011.
14. Banerjee C, Kundu A, Bhaumik S, Babu RS, Gupta RD, "Framework on Service based Resource Selection in Cloud Computing", Published in *International Journal of Information Processing and Management*, Volume 3, Number 1, pp. 17-25, 2012.
15. Akana CM, Kumar S, Divakar C, Satyanarayana Ch, "Dynamic Resource Allocation in Computing Clouds through Distributed Multiple Criteria Decision Analysis using PROMETHEE Method", Published in *International Journal of Advanced Networking and Applications*, Volume 03, Issue 02, pp. 1060-1069, 2011.
16. Xiao Z, Song W, Chen Q, "Dynamic Resource Allocation using Virtual Machines for Cloud Computing Environment", Published in *IEEE Transaction on Parallel and Distributed Systems (TPDS)*, June, Volume 24 Number 6, pp. 1107-1117, 2013.

17. A B, Lesser V, Irwin D, Zink M, "Automated Negotiation with Decommitment for Dynamic Resource Allocation in Cloud Computing", Published in proceedings of 9th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS-2010), pp 981-988, 2010.
18. Kim M, Lee H, Nam W, "A Model of Multi-Agent Design for Virtualization Resource Configuration in Cloud Computing", Published in 1st IEEE ACIS International Conference on Computers, Network, Systems, and Industrial Engineering, pp. 234-239, 2011.
19. Zhou Y, Yang Y, Liang L, Sun Z, "An Agent-Based Scheme for Supporting Service and Resource Management in Wireless Cloud", Published in proceedings of 9th IEEE International Conference on Grid and Cloud Computing, UK, pp 34-39, 2010.
20. Garcia JO, Sim, KM, "Self-Organizing Agents for Service Composition in Cloud Computing", Published in 2nd IEEE International Conference on Cloud Computing Technology and Science, pp. 59-66, 2010.
21. Manvi SS, Birje MM, Prasad B, "An Agent – based Resource Allocation Model for Computational grids", Published in Multiagent and Grid System- An International Journal, pp. 17-27, 2005.
22. Jung G, Sim KM, "Agent based Adaptive Resource Allocation on the Cloud Computing Environment", Published in IEEE conference on Parallel Processing Workshop, 2011, pp. 345-351, 2011.
23. Yee-Ming, C, Hsin-mie, Y, "Autonomous Adaptive Agents for Market-Based Resource allocation of Cloud Computing", Published in proceedings of the 9th International Conference on Machine Learning and Cybernetics, Qingdao, 11-14 July, pp. 2760-2764, 2010.
24. Rui H, Li G, Moustafa MG, Yike G, "Lightweight Resource Scaling for Cloud Applications", Published in proceeding of IEEE/ACM International Symposium on Cluster Cloud and Grid Computing, pp 644-651, 2012.
25. Giridas K L, Shajin N, "Optimal Resource Allocation Technique (ORAT) for Green Cloud Computing", Published in Journal of Computer Applications, Volume 55, Number 5, pp 20-26, 2012.
26. Ke L, Hai J, Jinjun C, Xiao L, Dong Y, Yun Y, "A Compromised-Time-Cost Scheduling Algorithm in SwinDeW-C for Instance-Intensive Cost-Constrained Workflows on Cloud Computing Platform", Published in International Journal of High Performance Computing Applications, Volume 24, Issue 4, pp 445-456, 2010.
27. Y. Kardem, K. Sirvastava and R. Sharma, "Minimizing Profile of Graphs Using a Hybrid Simulating Annealing Algorithm", Elsevier Journal of Electronic Notes in Discrete Mathematics, Vol. 63, pp. 381-388, 2017.
28. J. Zhao, K. Yang, X. Wei, Y. ing, L. Hu, and G. Xu "A Heuristic Clustering-Based Task Deployment Approach for Load Balancing Using Bayes Theorem in Cloud Environment", IEEE transactions on parallel and distributed systems, vol. 27, no. 2, february 2016.
29. Shaik, Mahaboob Basha, Kunam Subba Reddy, K. Chokkanathan, Sardar Asad Ali Biabani, P. Shanmugaraja, and DR Denslin Brabin. "A Hybrid Particle Swarm Optimization and Simulated Annealing with Load Balancing Mechanism for Resource Allocation in Fog-Cloud Environments." IEEE Access (2024).
30. P., Manoj Kumar, Sanjay Misra, Bibhudatta Sahoo, and Ashok Kumar Turuk. "GWO-based simulated annealing approach for load balancing in cloud for hosting container as a service." Applied Sciences 12, no. 21 (2022): 11115.
31. Jena, Uttam Kumar, Pradipta Kumar Das, and Manas Ranjan Kabat. "Hybridization of meta-heuristic algorithm for load balancing in cloud computing

- environment." *Journal of King Saud University-Computer and Information Sciences* 34, no. 6 (2022): 2332-2342.
32. Kruekaew, B. and Kimpan, W., 2022. Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning. *IEEE Access*, 10, pp.17803-17818.
 33. Fu, Xueliang, Yang Sun, Haifang Wang, and Honghui Li. "Task scheduling of cloud computing based on hybrid particle swarm algorithm and genetic algorithm." *Cluster Computing* 26, no. 5 (2023): 2479-2488.
 34. Malathi, K., and K. Priyadarsini. "Hybrid lion-GA optimization algorithm-based task scheduling approach in cloud computing." *Applied Nanoscience* 13, no. 3 (2023): 2601-2610.
 35. Liu, Jing, Xing-Guo Luo, Xing-Ming Zhang, Fan Zhang, and Bai-Nan Li. "Job scheduling model for cloud computing based on multi-objective genetic algorithm." *International Journal of Computer Science Issues (IJCSI)* 10, no. 1 (2013): 134.