ABSTRACT

In the recent era of cloud computing, the huge demand for virtual resource provisioning requires mitigating the challenges of uniform load distribution as well as efficient resource utilization among the virtual machines in cloud datacenters. Salp swarm optimization is one of the simplest, yet efficient metaheuristic techniques to balance the load among the VMs. The proposed methodology has incorporated self-adaptive procedures to deal with the unpredictable population of the tasks being executed in cloud datacenters. Moreover, a sigmoid transfer function has been integrated to solve the discrete problem of tasks assigned to the appropriate VMs. Thus, the proposed algorithm binary self-adaptive salp swarm optimization has been simulated and compared with some of the recent metaheuristic approaches, like BSO, MPSO, and SSO, for conflicting scheduling quality of service parameters. It has been observed from the result analysis that the proposed algorithm outperforms in terms of makespan, response time, and degree of load imbalance while maximizing the resource utilization.

KEYWORDS

Binary, BSO, Cloud Computing, Load Balancing, Makespan, MPSO, Resource Utilization, Self-Adaptive Salp Swarm Optimization Algorithm, SSO

1. INTRODUCTION

With the advancement in the computing paradigm, cloud computing has evolved as a ubiquitous, internet-based computing solution for business organizations throughout the globe. Cloud computing offers a shared pool of virtualized resources that can be provisioned as well as de-provisioned and can be accessed as a metering service by the customers on a pay-per-use model. Therefore, these characteristics made cloud computing ubiquitous and a utility. It delivers all of its services through a series of as-a-service models, such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) (Mell & Grance, 2011). It deploys various cloud models, such as Public Cloud, Private Cloud, Community Cloud, and Hybrid Cloud to address the necessities of the...
customers. Cloud computing has become a compelling hotspot among many business organizations and researchers because of the virtualization technology. This feature enables users to create an abstract environment by expanding the capacity of the underlying resources. The underlying physical resources are encapsulated as abstract entities to provide on-demand services to the customers (Mishra, Sahoo, & Parida, 2020) (JoSEP, KAtz, KonWinSKi, Gunho, PAtTERSon, & D., 2010). Virtualization is a technique that assimilates the fluctuating demands of growing customers. While scheduling tasks on virtual machines (VMs), there comes the point, when VMs are getting overloaded due to the lack of a proper load-balancing mechanism. It is also found that some VMs are underloaded or idle. Hence, there is a need to determine whether the system is balanced before scheduling. A slight deviation in load-balancing makes the system less reliable and thereby causing the degradation in performance. The load balancing should be improved in a way that makes sure that the overall loads are well dispersed by utilizing the resources significantly, and the makespan and the response time are considerably reduced by intensifying the overall performance.

Load balancing is the uniform distribution of loads among VMs, and it is essential to maximize the performance by utilizing the underlying VMs efficiently, where the load can be the total assignment of tasks to a virtual machine (VM). A significant load balancing technique not only improves the overall performance, but also ameliorates the quality of service (QoS) parameters such as response time, makespan, resource utilization, Degree of Imbalance (DoI), and so on (Milan, Rajabion, Ranjbar, & Navimipour, 2019). It is non-deterministic polynomial-time (NP) hard (Ibarra & Kim, 1977) (Ullman, 1975) to find out an optimal solution for a complex problem like load balancing due to the involvement of numerous contradicting scheduling parameters. The solutions to this problem are narrowed down into static and dynamic algorithms (Shah & Farik, 2015) (Chaharrooghi & Kermani, 2008). Static and dynamic are often referred to as heuristic and metaheuristic algorithms in the literature. Static algorithms are not extensively used in solving the aforementioned problem, because these algorithms are problem dependent, and do not guarantee to provide even near to optimal solutions. On the flip side, dynamic or metaheuristic algorithms are widely used, especially when the nature of tasks and resources are diversified. Metaheuristic algorithms are generally used for solving dynamically constrained and unconstrained optimization problems because these are problem independent (Kalra & Singh, 2015). It is suggested in the literature (Singh, Dutta, & Aggarwal, 2017) that metaheuristics are used to escape from local optima, as most of the algorithms get trapped in local optima during exploration of the problem space. Moreover, cloud load balancing is a dynamic problem, and hence, a binary variant of metaheuristic algorithm could be used for solving the above problem.

In this contribution, a binary variant of self-adaptive salp swarm optimization algorithm-based load balancing technique is proposed that not only maintains the trade-off among VMs but also maps the tasks onto the appropriate VMs. The key contributions in this piece of work are as follows.

- Salp swarm optimization is one of the simplest metaheuristic techniques that requires a smaller number of tuning parameters and is easy to implement as well. In any metaheuristic technique, it is cumbersome to determine and fix the population size because of the fluctuating nature of convergence. It is hard to say that when a metaheuristic algorithm will converge. Hence, it requires a method to determine the population size for a specific problem. In this regard, here, a self-adaptive approach is considered that will automatically determine the population size for the given problem.
- A binary version of the salp swarm optimization algorithm is incorporated into the primary algorithm. The reason for transforming the continuously generated solutions into discrete solutions is as because originally the salp swarm optimization is a continuous approach, whereas the task assignment to resources is a binary problem.
- A single-objective fitness function based on the load balancing problem is considered to evaluate the fitness of the tasks.
A cosine similarity method is adopted to determine the compatibility between the tasks and the underloaded VMs, to migrate the tasks from the overloaded VMs.

The remainder of the paper is organized as follows. Section 2 deals with the related work in the context of load balancing in cloud computing. Section 3 briefs the overview of the salp swarm optimization algorithm by highlighting the binary variant of salp swarm optimization. Section 4 presents the application of the proposed metaheuristic-based load balancing algorithm for addressing the load balancing issue in cloud computing. Section 5 discusses the considered simulation environment, along with the analysis of the experiments that are being conducted in a heterogeneous environment. Finally, section 6 concludes the research with conclusive remarks and future insights.

2. RELATED WORK

In a distributed computing environment, it is crucial to achieve an optimum response time of the system, while maximizing resource utilization. Task scheduling enhances the system to be flexible and reliable. The prime objective of the task scheduling algorithm is the dynamic allocation of resources in the system for the execution of the tasks. The fluctuating incoming requests of the users to the system is the main constraint for the proper resource mapping dynamically. Hence, balancing load among the resources evenly becomes too cumbersome. In this section, a review in the field of load balancing is presented. This section offers two groups of analysis as (1) using metaheuristics, and (2) using the combined approach of both heuristic and metaheuristic.

The first group discusses the work that is being carried out using the metaheuristic approach. Sharma et al. (Sharma, Rath, & Parida, 2020) have proposed a heuristic max select load balancing (MSLB) algorithm for both intra-datacenter and inter-datacenter systems without considering communication delay, as well as max select load balance with communication delay (MSLBCD) for different datacenters. Although the proposed techniques are providing better makespan and efficient resource utilization in comparison with some contemporary load balancing techniques, they are not considering multiple task allocation as well as the dynamic nature of the cloud system. If the cloud cross-zone load balancing is not enabled, then each load balancer node distributes the tasks across the VMs in its availability zone only. But on the contrary, each load balancer node distributes the tasks evenly across the VMs in all the availability zones with the cross-zone load balancing (aws, 2021). To reduce the makespan using the load balancing approach, (Mittal & Katal, 2016) have presented an optimized task scheduling algorithm combining RASA and Min-Min. Their algorithm outperforms some well-known heuristic methods such as Min-Min, Max-Min, improved Max-Min, and RASA. But traditional or heuristic methods are incapable of solving the task scheduling or load balancing problem in a real cloud environment (Mandal & Acharyya, 2015), therefore, it is recommended that metaheuristic algorithms like particle swarm optimization (PSO), simulated annealing (SA), firefly algorithm (FFA), cuckoo search algorithm (CSA), genetic algorithm (GA), ant colony optimization (ACO), JAYA, salp swarm optimization (SSO) and so on are most tailored approaches to solve these problems, which give near to optimal solutions. These approaches are also believed to escape the local optima and strive towards the global optima. Mandal and Acharyya have presented a firefly metaheuristic-based scheduling algorithm for solving the task scheduling in cloud computing by considering the makespan as an objective to be reduced. They have compared this algorithm with cuckoo search as well as simulated annealing and found it outperforming others. A study, inspired by the honey bee-based load balancing technique (HBB-LB) is developed to trade-off the loads among VMs (LD & Krishna, 2013). The priorities of tasks are considered for the selection of VM. However, the low-priority tasks remain in the queue for the longest time. The HBB-LB algorithm is compared with the first in first out (FIFO), weighted round robin (WRR), and dynamic load balancing (DLB) algorithms and showed better improvements in the overall performance. Li et al. have proposed a load balancing algorithm based on the ant colony optimization (ACO) technique that takes homogeneous
tasks (Li, Xu, Zhao, Dong, & Wang, 2011). Their approach outperformed in reducing the makespan and the response time but lacks scalability. Another study implements GA strategy for load balancing in dynamic cloud milieu (Dasgupta, Mandal, Dutta, Mandal, & Dam, 2013). It generates possible solutions by mapping tasks onto VMs and iterates to find the best solution. This strategy tries to eliminate the inappropriate distribution of loads onto the VMs while optimizing execution time. Kaur and Kaur have proposed an adaptive firefly algorithm (ADF) to balance the load in the VMs of multiple cloud datacenters (Kaur & Kaur, 2017). Their algorithm is found to outperform ACO in minimizing response time and processing time of the VMs successfully but has ignored the resource utilization. Mohanty et al. have proposed JAYA algorithm for load balancing in cloud computing to minimize average response time, but they have ignored the deadline constraints (Mohanty, Patra, Ray, & Mohapatra, 2019). An energy-aware cluster scheduling (EACS) technique has been proposed by Garg and Shukla for energy-efficient scheduling of multiple workflows in cloud computing (Garg & Shukla, 2018). Mishra and Majhi have applied the binary version of the bird swarm optimization (BSO) algorithm for proper load balancing among the VMs in a homogenous environment intending to minimizing makespan and response time while maximizing the VMs utilization (Mishra & Majhi, 2021). Kumar and Kumar have minimized migration time and energy consumption by implementing a salp swarm optimization (SSO) algorithm for dynamic resource management in the cloud milieu (Kumar & Kumar, 2021).

Table 1. Analysis of the related work

| Author(s)                    | Algorithm                          | Application                        | Advantages                                           | Drawbacks                                  | Performance Metrics                  |
|------------------------------|------------------------------------|------------------------------------|------------------------------------------------------|--------------------------------------------|---------------------------------------|
| Mishra and Majhi (2021)      | Binary Bird Swarm Optimization (BSO)| Load balancing                      | • Minimizing makespan                               | • Homogenous environment                   | • Makespan                           |
|                              |                                    |                                    | • Minimizing response time                           |                                            | • Response time                       |
|                              |                                    |                                    | • Maximizing resource utilization                   |                                            | • Resource utilization               |
| Kumar and Kumar (2021)       | Salp Swarm Optimization (SSO)      | Dynamic resource management         | • Minimizing migration time                          | • Ignored balancing of load               | • Response time                       |
|                              |                                    |                                    | • Minimizing energy consumption                     |                                            | • Resource utilization               |
| Sharma et al. (2020)         | Max select load balancing (MSLB) and Max select load balance with communication delay (MSLBCD) | Load balancing | • Minimizing makespan                               | • Static cloud                            | • Makespan                           |
|                              |                                    |                                    | • Maximizing resource utilization                   |                                            | • Resource utilization               |
|                              |                                    |                                    | • Multi Datacenter                                   |                                            |                                      |
|                              |                                    |                                    | • Heterogeneous cloud environment                    |                                            |                                      |
| Nayak and Tripathy (2019)    | Multi-criteria Decision-making (MCMD) and Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) | Task Scheduling | • Maximizing resource utilization                    | • Updating of lower bound slack value    | • Execution time                      |
|                              |                                    |                                    | • Minimizing tasks rejection                         |                                            | • Deadline of tasks                  |
|                              |                                    |                                    | • Scheduling deadline based tasks                    |                                            |                                      |
|                              |                                    |                                    | • Scheduling similar tasks                          |                                            |                                      |
| Mohanty et al. (2019)        | JAYA algorithm                      | Load balancing                      | • Minimizing average response time                   | • Ignored deadline constraints           | • Response time                       |
| Garg and Shukla (2018)       | Energy Aware Cluster Scheduling (EACS) | Energy efficient scheduling for multiple workflows | • Minimizing execution time                          | • Ignored resource utilization           | • Percentage of energy saving        |
|                              |                                    |                                    | • Minimizing energy consumption                      |                                            |                                      |
|                              |                                    |                                    | • Scheduling multiple workflows                      |                                            |                                      |

Table 1 continued on next page
Table 1 continued

| Author(s)                  | Algorithm                                                                 | Application                                      | Advantages                                                                 | Drawbacks                                                                 | Performance Metrics                                    |
|----------------------------|---------------------------------------------------------------------------|--------------------------------------------------|-----------------------------------------------------------------------------|---------------------------------------------------------------------------|----------------------------------------------------------|
| Ebadifard and Babamir (2018) | Improved Particle Swarm Optimization with load balancing (MPSO)         | Task scheduling and Load balancing               | • Minimizing makespan  
• Maximizing resource utilization   | • Homogenous environment   | • Makespan  
• Resource utilization  
• Response time                             |
| Kaur and Kaur (2017)        | Adaptive firefly algorithm (ADF)                                         | Load balancing                                   | • Minimizing response time and processing time  
• Multi Datacenter  | • Ignored resource utilization   | • Response time  
• Processing time                             |
| Khalili and Babamir (2017)  | Pareto Grey Wolf Optimizer (PGWO)                                        | Task scheduling                                  | • Minimizing cost and makespan  
• Maximizing resource utilization and throughput   | • Considered dependent tasks only   | • Coverage ratio  
• Maximum spread  
• Distance based distribution                             |
| Mittal and Katal (2016)     | Optimized task scheduling algorithm (Combination of RASA and Min-Min)   | Task scheduling                                  | • Minimizing makespan   | • Not implemented in cloud computing environment  
• Low reliability                             | • Makespan                             |
| Mandal and Acharyya (2015)  | Simulated annealing (SA), Firefly algorithm (FFA) and Cuckoo search algorithm (CSA) | Task scheduling                                  | • Minimization of total processing time   | • Implemented existing algorithms   | • Processing time                             |
| Bilgaiyan et al. (2015)     | Multi objective cat swarm optimization (MOCSO)                          | Task scheduling                                  | • Minimizing makespan and CPU idle time  
• Maximizing resource utilization   | • Ignored many real time conflicting objectives   | • Makespan  
• Resource utilization  
• CPU idle time                             |
| Kaur and Sharma (2014)      | Combination of Particle Swarm Optimization and Simulated Annealing       | Task scheduling                                  | • Minimizing Execution time  
• Maximizing resource utilization   | • Not simulated   | • Execution time  
• Resource utilization                             |
| LD and Krishna (2013)       | Honey bee behavior inspired load balancing (HBB-LB)                      | Load balancing and Task scheduling               | • Minimizing makespan, response time, degree of imbalance and number of task migration  
• Maximizing throughput   | • Waiting time of low priority tasks increase  
• Ignored resource utilization   | • Makespan  
• Response time  
• Degree of imbalance  
• Task migration                             |
| Dasgupta et al. (2013)      | Genetic algorithm (GA)                                                  | Load balancing and Task scheduling               | • Minimizing response time  
• Maximizing resource utilization   | • Homogeneous environment  
• Tasks are considered to be of same priority   | • Response time                             |
| Li et al. (2011)            | Load balancing ant colony optimization (LBACO)                          | Load balancing and Task scheduling               | • Minimizing makespan and degree of imbalance   | • Homogenous environment  
• Independent tasks  
• Computationally intensive tasks  
• Poor scalability   | • Makespan  
• Degree of imbalance                             |
The second group of researchers discusses the state-of-the-art literature that combines either two metaheuristics or a metaheuristic with heuristic ones. This is often regarded as a hybrid algorithm. The advantage of the hybrid approach is the drawback of one method will be surmounted with the strengths of another method. Mixing a load balancing approach into a task scheduling algorithm, Ebadifard and Babamir have presented a modified PSO-based task scheduling in cloud computing. This approach is inspired by honey bee behavior. They have taken makespan, response time, and resource utilization as scheduling parameters to validate their work. The algorithm is validated using a synthetic as well as a real-world dataset. The algorithm has shown considerable improvements over compared algorithms, but the algorithm has considered only a homogeneous environment (Ebadifard & Babamir, 2018). Combining PSO with SA, Kaur and Sharma have proposed a task scheduling method for cloud computing by considering resource utilization (Kaur & Sharma, 2014). Their work aimed at optimizing resource utilization, thereby maximizing the profit of the cloud service provider. By considering multi-objective scheduling, Bilgaiyan et al. have proposed a cat swarm optimization-based workflow scheduling algorithm (Bilgaiyan, Sagnika, & Das, 2015). Their work aims at reducing the makespan as well as the idle time of the VMs and the overall cost. They have claimed that their work outperforms others. Another study (Khalili & Babamir, 2017) showcases the reduction in makespan, cost, and improvement in throughput by considering the requirement of QoS parameters for the cloud service provider. This work uses a grey wolf optimizer (GWO) based multi-objective scheduling algorithm which uses a Pareto front, and it has been compared with the strengths of other Pareto-based evolutionary strategies. Nayak and Tripathy have proposed to implement multi-criteria decision-making (MCDM) and technique for order preference by similarity to ideal solution (TOPSIS) for scheduling the deadline sensitive tasks in a cloud environment by resolving the conflicts among the similar tasks (Nayak & Tripathy, 2019). Their proposed methodology has outperformed in reducing the task rejection as compared to the available backfilling algorithms.

The analysis of the related work is summarized in Table 1. The algorithms in the literature along with their applications are focused on for summarization. The advantages, drawbacks, and performance metrics of all the algorithms are analyzed briefly.

This research paper concentrates on using an SSO-based metaheuristic algorithm to decipher the problem of load balancing in cloud-based computing, which is elaborated in the following section. This load balancing approach takes inspiration from the PSO-based load balancing (Ebadifard & Babamir, 2018), and honey bee-inspired HBB-LB load balancing (LD & Krishna, 2013). Tasks are assigned to the VMs in a time-shared policy (Round Robin). The key idea of this contribution is to reduce the makespan, response time, and degree of load imbalance while maximizing VMs utilization. In this paper, a load balancing approach is incorporated into the task scheduling method to ameliorate the overall performance. It has been observed from the literature survey that several works have been done by different researchers in the field of load balancing in cloud computing. However, they have considered very few QoS parameters related to load balancing. Hence, this piece of work has incorporated most of the QoS parameters like makespan, response time, VM utilization, and degree of load imbalance to justify the effectiveness of the proposed approach. In addition to the QoS parameters, we have considered a metaheuristic algorithm called SSO for optimization. In this paper, this algorithm is updated to deal with the real world inconsistent, irregular tasks entering into the datacenter for execution, by introducing the self-adaptive procedure. Moreover, this proposed algorithm has also modified the continuous nature of the original SSO technique to handle the assignment of tasks to the resources problem of discrete nature.

3. THE IMPROVED BINARY SELF-ADAPTIVE SALP SWARM OPTIMIZATION (BSASSO) ALGORITHM

In this section, the improved version of binary self-adaptive salp swarm optimization algorithm followed by an overview of the standard salp swarm optimization algorithm is presented.
3.1 The Standard Salp Swarm Optimization Algorithm

In the year 2017, Mirjalili et al. have proposed a swarm-based nature-inspired technique named as salp swarm optimization (SSO) algorithm (Mirjalili, H, Mirjalili, Saremi, Faris, & Mirjalili, 2017). This algorithm is inspired by the swarming behavior of salps, which are found in the deep ocean. This creature forms a chain-like structure called the salp chain in the deep ocean to forage the food sources. They look for the food sources by dividing themselves into two groups: (1) leader, at the front of the chain, who navigates the whole population, and (2) follower, the rest of the population follow the leader. The fitness of all the salps is calculated, and the fittest one is chosen as the leader. This algorithm is best suitable for maintaining the balance between exploration and exploitation and for the faster convergence by evading through the local entrapment as well (Mohapatra & Rath, 2019) (Mohapatra & Rath, 2020). The principle of the SSO algorithm can be modeled to resolve the issue of balancing the load in a dynamic cloud environment. The salps swarming in the population are considered as tasks, and the food patches are as the destined VMs. As salps compete with themselves to become a leader and explore the best food sources, tasks also compete to get into the most compatible VMs. In the considered problem domain, the fitness of all the tasks is also evaluated using a fitness function. Mathematically, the location of the salps is represented in an N-dimensional search space, where N is the total number of salps in the entire salp-chain means population. According to Equation 1 (Mirjalili, H, Mirjalili, Saremi, Faris, & Mirjalili, 2017), the position of the salp leader is updated. The random value ($r_1$) responsible for establishing the balance between the exploration and exploitation is calculated using Equation 2. The follower salps’ positions are modified based on Equation 3.

\[
X_j^i = \begin{cases} 
F_j + r_1 \left( (UB_j - LB_j) r_2 + LB_j \right), & r_3 \geq 0 \\
F_j - r_1 \left( (UB_j - LB_j) r_2 + LB_j \right), & r_3 < 0 
\end{cases}
\] (1)

Where $X_j^i$ is the leader’s position in the salp chain at the $j$th -dimension, $F_j$ is the position of the food source in the $j$th -dimension, $UB_j$ and $LB_j$ represent the upper bound and lower bound of $j$th -dimension. $r_2$ and $r_3$ are uniformly generated random numbers between 0 and 1.

The random value ($r_1$) is the key parameter, which is responsible for the trading-off between exploration and exploitation. $r_1$ is calculated using the following equation (Mirjalili, H, Mirjalili, Saremi, Faris, & Mirjalili, 2017).

\[
r_1 = 2e^{\left( \frac{-t}{T_{max}} \right)}
\] (2)

Where $T_{max}$ represents the maximum number of iterations and $t$ imitates the current iteration.

This parameter $r_1$ descends along with the ascending values of the number of iterations, thereby; it maintains the uniformity in both exploration and exploitation. The position of the followers is updated as per the Equation 3 (Mirjalili, H, Mirjalili, Saremi, Faris, & Mirjalili, 2017), which is inspired by Newton’s law of motion.
\[ X_j^i = \frac{X_j^i + X_j^{i-1}}{2} \]  

(3)

Where \( i \geq 2 \) and \( X_j^i \) is the position of the \( i \)th follower in the \( j \)th dimension.

It is worth noting that the chain of the salps possesses the capability of striving ahead continuously towards the fluctuating global optima (food source) to give an optimal solution (Khan, Alghamdi, Jumani, Alamgir, Awan, & Khidrani, 2019). Some of the important characteristics of SSO are enlisted as follows, which makes this algorithm different from the conventional metaheuristic algorithms (Mirjalili, H, Mirjalili, Saremi, Faris, & Mirjalili, 2017).

- This algorithm keeps track of the best solutions obtained after each execution and allocates them to the global optima;
- The SSO modifies the position of a leader with respect to the food patch; therefore, the leader could explore and exploit it to obtain an improved solution;
- The SSO updates the position of the followers corresponding to each other, which boosts them up to aim towards the global optima;
- This algorithm uses very few tuning parameters, which makes this algorithm easy to implement and lessens the complexity;
- The tuning parameter \( r \) descends during the course of executions that helps the algorithm in exploring the problem domain at the beginning and exploiting it at the end.

3.2 The Self-Adaptive SalpSwarm Optimization Algorithm

Determining the population size in any metaheuristic is a burdensome task. Therefore, authors consider a self-adaptive SSO algorithm, which determines the population size automatically. Let’s presume that the random initial population is \((10 \times m)\), where \( m \) represents the number of VMs. Hence, the new population is formulated as given below in Equation 4 (Salgotra, Singh, Singh, Singh Gill, & Mittal, 2020).

\[ n_{new} = \text{round} \left( n_{old} + r \times n_{old} \right) \]  

(4)

Where \( r \) is a random value, which varies in a range of \([-0.5, 0.5]\) and serves as a progressing rate. Due to the positive or negative random value of \( r \), the population size may increase or decrease.

The Elitism technique is implemented, when the size of the new population is compared with the size of the old population, whether it is greater than or less than or equal to old one. If the new population size is greater than the old population size \((n_{new} > n_{old})\), then all of the old population will be forwarded to the next generation and the best optimal solution in the new population is allocated to the rest of the solutions \((n_{new} - n_{old})\). When the new population size is less than the old population size \((n_{new} < n_{old})\), then only the best solution in the current population will be carried to the next generation. If the population sizes of new and old population are equal to each other \((n_{new} = n_{old})\), then no changes will be taken place. If the population size becomes less than the number of design variables or VMs \((m)\), then the population size is treated equal to the number of VMs \((n_{new} = m)\). Hence, the solution will not be trapped in local optima.
3.3 The Binary Self-Adaptive Salp Swarm Optimization Algorithm

Originally, the standard SSO algorithm is developed to solve the continuous optimization problem. As the task assignment to resources is modeled as a binary problem in cloud computing, it is essential to transform the generated solutions into discrete values. In binary SASSO, the tasks (salps) are moved around in a bounded direction that is confined to only 0 and 1. One of the convenient ways to transform continuous solutions into discrete solutions is the use of transfer function (TF). Equation 5 (Rizk-Allah, Hassanien, Elhoseny, & Gunasekaran, 2019) is a sigmoid transfer function which is used in this research for conversion.

\[
S\left( X_{j}^{i+1} \right) = \text{sig}\left( X_{j}^{i+1} \right) = \frac{1}{1 + e^{-X_{j}^{i+1}}}
\]

Where

\[
X_{j}^{i+1} = \begin{cases} 
1, & \text{if } S\left( X_{j}^{i+1} \right) > \text{ran.nextInt}(2) \\
0, & \text{otherwise}
\end{cases}
\] (5)

4. APPLICATION OF BSASSO ALGORITHM IN SOLVING LOAD BALANCING FOR CLOUD COMPUTING

This section describes the mathematical model and problem formulation of the application of binary self-adaptive salp swarm optimization algorithm in cloud load balancing.

4.1 Mathematical Model and Problem Formulation

A cloud consists of many datacenters which house many physical machines to service the on-demand requests of users. In this paper, one datacenter is considered that handles the responsibility of processing the requests. Cloud-based computing uses virtualization technology for creating an isolated, configurable, and secured execution environment to process the high-end requests concurrently. This technology enables the host machines to extend their capacities and emulate the configurations to create an abstract environment. Thus, a host machine consists of a number of virtual machines which process a series of incoming high-end requests.

The load balancing problem is formulated as follows. It is considered that a datacenter encompasses a series of physical machines. A number of VMs \( VM = \{ VM_1, VM_2, VM_3, \ldots, VM_m \} \) are hosted under the series of hosts. Consider a task set \( T = \{ T_1, T_2, T_3, \ldots, T_n \} \) where \( T_i, 1 \leq i \leq n \) is the \( i^{th} \) task with a million sets of instructions (MI) in \( T_i \) of non-preemptive and independent nature is going to be scheduled on VMs. The QoS parameters are examined to assess the performance of the proposed algorithm. Response time, makespan, resource utilization, and degree of load imbalance are treated as the QoS parameters.

The completion time of a task \( T_i \) on \( VM_j \) is denoted as \( CT_{ij} \). It is the time taken to execute a task on a VM. This paper aims at reducing the completion time to improve the users’ satisfaction. In order to reduce the \( CT_{ij} \), the processing time of all tasks on \( VM_j \) ought to be reduced. The processing time of a task \( T_i \) on \( VM_j \) is denoted as \( PT_{ij} \). Hence, the processing time of all tasks on \( VM_j \) can be illustrated as follows in Equation 6.
\[ PT = \sum_{i=1}^{n} PT_{ij}, \quad j = 1, 2, 3, \ldots, m \]  

(6)

During the process of load balancing, tasks will be migrated from overloaded VMs to underloaded VMs to minimize the overall completion time (makespan) as well as the response time. Therefore, the overall resources will also be utilized efficiently through the process of balancing the load. In order to reduce the makespan of a VM which is denoted as \( CT_{\text{max}} \), the completion time of all tasks needs to be minimized. Hence, the makespan can be represented as the maximum of the overall completion time of all tasks on \( VM_j \). It can be calculated using Equation 7.

\[ CT_{\text{max}} = \max \{ CT_{ij} | i = 1, 2, \ldots, n; j = 1, 2, \ldots, m \} \]  

(7)

For instance, there are 3 tasks (\( T_1, T_2, \text{ and } T_3 \)) which are scheduled to be executed on \( VM_1 \). The completion times of \( T_1 \), \( T_2 \), and \( T_3 \) are 3.84, 4.62, and 2.89 respectively. Hence, the makespan (\( CT_{\text{max}} \)) of \( VM_1 \) is the maximum completion time of all tasks, i.e. 4.62.

Response time can be specified as the time taken to acknowledge the users’ request. It is denoted as \( R_t \) and expressed in s (seconds). It can be determined using Equation 8, where \( n \) is the number of tasks.

\[ R_t = n \times CT_{ij} \]  

(8)

In order to improve performance, the VMs also should be significantly utilized. The efficient utilization of VMs could be possible only through a proper load balancing approach. The utilization of VM (\( VM_{\text{util}} \)) can be defined using Equation 9 (Mapetu, Chen, & Kong, 2019)(Khorsand, Ghabaei Arani, & Ramezanpour, 2019).

\[ VM_{\text{util}} = \sum_{i=1}^{n} \frac{CT_{ij}}{CT_{\text{max}}} \]  

(9)

The average utilization of all VMs can be expressed using Equation 10, where \( m \) is the number of VMs (Rafieyan, Khorsand, & Ramezanpour, 2020).

\[ AVG_{VM_{\text{util}}} = \frac{\sum_{i=1}^{m} vm_{\text{util}}}{m} \]  

(10)

4.1.1 Capacity of VMs

The capacity of a VM can be defined using the following Equation 11 (LD & Krishna, 2013).

\[ Cap_j = \text{num}(PEs)_{VM_j} \times MIPS(PEs)_{VM_j} + BW(VM_j) \]  

(11)
Where \( \text{num} \left( PEs \right)_{VM_j} \) is the number of processing elements in a VM, \( MIPS \left( PEs \right)_{VM_j} \) is the million instructions per second of all the processing elements of a VM, and \( BW \left( VM_j \right) \) is the bandwidth of a VM.

The capacity of all the VMs can be expressed as follows.

\[
Cap = \sum_{j=1}^{m} Cap_j
\]

(12)

4.1.2 Loads of VM

The load of a VM (\( Load_{VM_j} \)) is the assignment of total workloads to a VM (Polepally & Chatrapati, 2019). It can be described as the ratio of the number of tasks on the job queue at time \( k \) divided by the service rate of VM at time \( k \).

\[
Load_{VM_j} = \frac{Num \left( T^k \right)}{SR \left( VM_j^k \right)}
\]

(13)

The load of all the VMs can be expressed as follows.

\[
Load \left( VMs \right) = \sum_{j=1}^{m} Load_{VM_j}
\]

(14)

4.1.3 Grouping of VMs

The whole VMs are grouped into three groups according to their loads. The group can be formed by comparing the load of a VM (\( Load_{VM_j} \)) with the capacity of all the VMs (\( Cap \)). If \( Load_{VM_j} > Cap \) then it can be identified as overloaded VM (OVM), else if \( Load_{VM_j} < Cap \) then it can be identified as underloaded VM (UVM) and otherwise it is considered as balanced VM (BVM). During the process of balancing the load, the tasks from OVM will be migrated to the UVM based on the availability of loads in UVM. In this approach, these tasks are taken as salps, and under loaded VMs are referred to as food patches.

4.1.4 State of the VM

The state of the VM can be recognized by calculating the standard deviation of loads (\( \sigma \)). It is a measure to detect the lack of load deviation in the system (LD & Krishna, 2013), and it is expressed as follows.

\[
\sigma = \sqrt{\frac{1}{m} \sum_{j=1}^{m} \left( PT_j - PT \right)^2}
\]

(15)
Where $PT_j$ is the processing time of VM$_j$ and $PT$ is the processing time of all the VMs and expressed as follows.

$$PT_j = \frac{Load_{VM_j}}{Cap_j} \quad (16)$$

$$PT = \sum_{j=1}^{m} PT_j \quad (17)$$

4.1.5 Load Balancing decision

The system should take the decision, whether to trigger the process of load balancing. In order to do so, there are two possible solutions.

1. Finding the state of the VMs: the value of the $\sigma$ will be compared with a threshold value ($Tsh_v$) which ranges in between 0 and 1 (De Mello, Senger, & Yang, 2006). If the value of $\sigma$ will be greater than $Tsh_v$ ($\sigma > Tsh_v \ [0-1]$) then the system will be identified as imbalanced, and the desired load balancing procedure will be triggered.

2. Finding out of the capacity condition: is a condition to check whether load balancing is possible. If the value of VMs’ load ($Load(VMs)$) exceeds the overall capacity of all VMs ($Load(VMs) > Cap$), then it is not possible to balance the load, otherwise the load balancing process will be triggered.

4.1.6 Task Migration

The task migration process is initiated to ensure the balanced state of the VMs as well as the overall system. When a VM is identified as an OVM, then the tasks of OVM are transferred to a compatible UVM based on the available resources and total resources used. A compatible UVM is selected based on the cosine similarity (Nanduri, Maheshwari, Reddyraja, & Varma, 2011). The greater the similarity index, greater is the compatibility between the tasks of OVM and UVM. Then the tasks are transferred to the compatible UVM and checks for the balanced state of the VMs. If a balanced state is achieved, scheduling of tasks will be carried out. Otherwise, the load balancing process will be executed.

In this research, a resource usage pattern (CPU, BW, Storage) of the task denoted as $T_i$ is used. In order to migrate the tasks onto UVM, it is essential to find out the total consumption of resources and total availability of resources in UVM, which are denoted as $TR_{consume}$ and $TR_{avail}$ respectively. $TR$ is used to express the total resources of UVM. Therefore, the $TR_{consume}$ and $TR_{avail}$ can be calculated using the following equations.

$$TR_{consume} = \sum_{i=1}^{n} T_i \quad (18)$$
TR_{avail} = TR - TR_{consume} \tag{19}

The similarity between the task of OVM (T_{OVM}) and available resources of UVM (TR_{avail}) can be calculated using cosine similarity (angle). The smaller value indicates the greater similarity between T_{OVM} and TR_{avail}. It is calculated at each iteration, and it is expressed using Equation 20.

\[
\text{angle} = \cos^{-1}\left(\frac{T_{OVM} \times TR_{avail}}{|T_{OVM}| \cdot |TR_{avail}|}\right) \tag{20}
\]

4.1.7 Degree of Imbalance

The degree of imbalance (DoI) is used to measure the load imbalance among the VMs, which should be minimized to make the system balanced. It is calculated as follows (Li, Xu, Zhao, Dong, & Wang, 2011).

\[
\text{DoI} = \frac{T_{\text{max}} - T_{\text{min}}}{T_{\text{avg}}} \tag{21}
\]

Where \(T_{\text{max}}\), \(T_{\text{min}}\) and \(T_{\text{avg}}\) are the maximum, minimum and average total execution time (\(T_i\)) of all VMs respectively. These are calculated from each \(T_i\) using the following equation.

\[
T_i = \frac{L_i}{P_{\text{numj}} \times P_{\text{mipsj}}} \tag{22}
\]

Where \(L_i\), \(P_{\text{numj}}\) and \(P_{\text{mipsj}}\) are total length of all tasks submitted to VM_j, the total number of processors of VM_j and the MIPS of each processor of VM_j respectively.

4.1.8 Fitness Function

The primary goal of this research is to reduce the makespan and response time by improving VM utilization. In order to notice significant improvements over all the aforementioned constraints, a proper load balancing technique should be implemented. In this paper, a single-objective fitness function is considered that includes minimization of makespan (\(C T_{\text{max}}\)), the response time (\(R_t\)) and degree of imbalance (DoI), and maximization of VM utilization (\(VM_{\text{util}}\)). Fitness function is problem-specific, and hence, it can be defined using Equation 21.

\[
F_{\text{val}} = \frac{1}{CT_{\text{max}}} \times \frac{1}{R_t} \times VM_{\text{util}} \times \frac{1}{\text{DoI}}
\]
subject to:

\[
\begin{align*}
&\min \left( C T_{\text{max}} \right) \\
&\min \left( R_t \right) \\
&\max \left( V M_{\text{util}} \right) \\
&\min \left( D o I \right)
\end{align*}
\]

The problem of load balancing is formulated in a way that it maps the tasks onto the corresponding VMs \(( T \rightarrow VM )\) by satisfying the following objectives. The key objectives of this research are: (1) reduce the overall makespan, (2) reduce the response time, (3) reduce the load imbalance, and (4) improve the overall VM utilization.

4.2 The BSASSO Based Load Balancing Algorithm

In this sub-section, the proposed load balancing algorithm using Binary Self Adaptive Salp Swarm Optimization (BSASSO) is presented. This algorithm not only balances the overall loads across VMs but also schedules the tasks on VMs.

| Algorithm 1 Pseudo code for load balancing and scheduling tasks onto VMs using BSASSO algorithm |
| --- |
| **Input:** Set of Tasks, Set of VMs, Maximum Iteration \( T_{\text{max}} = 1000 \);  
**Output:** A balanced state of VMs |

**Start**

1. Set salps’ dimension to the total number of tasks and food patches to VMs set, and initialize population size \( n = 10 \times m \);  
2. Calculate Makespan, Response time, Average resource utilization, capacity, and load of all VMs, using Equations 7, 8, 10, 12, and 14 and check whether load balancing is possible;  
3. Calculate Standard deviation of load by Equation 15;  
4. Organize the VMs into groups, such as OVM, UVM and BVM, based on their load.
Algorithm 1 Pseudo code for load balancing and scheduling tasks onto VMs using BSASSO algorithm

5. Tasks Migration Procedure:

\[ \text{while} \left( \text{UVM IsNot Null} \right) \]

Estimate \( TR_{\text{consume}} \) and \( TR_{\text{avail}} \) of all the under loaded VMs by Eqs. 18 and 19;

Find the similarity \((angle)\) of tasks of overloaded VMs with available resources of under loaded VMs by Equation 20

6. Task scheduling using BSASSO:

For all tasks \((T_i)\), evaluate their fitness by Equation 23;

\[ \text{if} \left( F_{\text{val}} < F_{\text{best}} \right) \]

Set the new fitness value as the \( F_{\text{best}} \);

For each task \((T_i)\),

\[ \text{if} \left( i == 1 \right) \]

Update the position of leading task by Equation 1;

\[ \text{else} \]

Update the position of follower tasks by Equation 3;

7. Modify position by transforming continuous new solution to discrete solution by Equation 5;

8. Repeat until terminating condition;

\[ \text{if} \left( \text{termination condition not met} \right) \]

Produce new set of population using Equation 4;

\[ \text{if} \left( n_{\text{new}} > n_{\text{old}} \right) \]

Amend best solutions in the new population to the \( n_{\text{new}} - n_{\text{old}} \) solutions;

\[ \text{else if} \left( n_{\text{new}} < n_{\text{old}} \right) \]

Amend only the best solutions in the current population to the \( n_{\text{new}} \) solutions;

\[ \text{else if} \left( n_{\text{new}} < m \right) \]

set \( n_{\text{new}} = m \)

\[ \text{else} \]

Display the optimum solution;

9. Return best possible solution to the proposed mapping of tasks to VMs;

\text{End}

The time complexity of the above algorithm is of \( O(n_{\text{new}} \times UVM \times f_{\text{max}}) \), where \( n_{\text{new}} \) is the population size, \( UVM \) is the dimension of UVM set and \( f_{\text{max}} \) is the maximum number of function evaluations.
5. EXPERIMENTAL SETUP AND RESULTS ANALYSIS

In this section, the authors present the experimental setup to carry out the desired experiments using a real-world dataset followed by an analysis of obtained results.

5.1 Experimental Test cases

The classes of CloudSim (Calheiros, Ranjan, Beloglazov, De Rose, & Buyya, 2011) have been extended to process the tasks by emulating the physical infrastructure. Cloud-based computing consists of a datacenter that encompasses four numbers of hosts with disparate configurations which further expanded to create a virtualized environment. Each host comprises of some processing cores, RAM, Storage, bandwidth (BW) and speed of processing. The technical configurations of each host has been specified in Table 2. 300 VMs with different specifications are considered for scheduling non-preemptive tasks. The VM technicalities are presented in Table 3. Independent and non-preemptive nature of 3000 numbers of disparate tasks with varied lengths [0-10000] are considered for the scheduling under the series of VMs. In this research, authors consider a heterogeneous environment, where both the number of tasks and VMs vary simultaneously in order to validate the effectiveness of the proposed algorithm. Each of the considered QoS parameters is evaluated against a set of tasks × VMs set. The tasks × VMs set consists of six sets of instances as (1) 500 × 50, (2) 1000 × 100, (3) 1500 × 150, (4) 2000 × 200, (5) 2500 × 250, and (6) 3000 × 300. The QoS parameters such as Makespan, Response time, VM Utilization, and Degree of Imbalance are considered to validate the proposed algorithm. A real-world dataset, i.e. GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures by Google published in September 2018 (Hussain & Aleem, GoCJ: Google Cloud Jobs Dataset for Distributed and Cloud Computing Infrastructures, 2018) is used to carry out the simulation. This dataset has 19 text files which consist of varying lengths of tasks. Out of which, authors consider a combination of GoCJ_DataSet_Monte_Carlo.xlsx (Hussain & Aleem, GoCJ: Google Cloud Jobs Dataset, 2018) and GoCJ_DataSet_950.txt (Hussain & Aleem, 2018) to validate the efficacy of the proposed algorithm.

Table 2. Host Technicalities

| Host ID | Processing Cores | Speed, MIPS | RAM, GB | Storage, GB | BW, MIPS |
|---------|-----------------|-------------|--------|-------------|----------|
| 1       | 4               | 4500        | 200    | 1024        | 102400   |
| 2       | 3               | 4000        | 150    | 1024        | 102400   |
| 3       | 2               | 3500        | 100    | 1024        | 102400   |
| 4       | 1               | 3000        | 50     | 1024        | 102400   |

Table 3. VM Technicalities

| CPU | Number of Cores | Speed, MIPS | RAM, GB | Storage, GB | BW, MIPS |
|-----|-----------------|-------------|--------|-------------|----------|
| Core_i7_Extreme_Edition | 1-4 | 500 | 2 | 20 | 1024 |
5.2 Results Analysis

This section presents a meticulous layout on the obtained results for makespan, response time, VM utilization, and load-balancing, respectively. The experiments are conducted in a heterogeneous environment where varied lengths of tasks and an increasing number of VMs are considered. The proposed algorithm is compared with some well-known metaheuristic load balancing algorithms such as (1) Bird Swarm Optimization (BSO), (2) Modified Particle Swarm Optimization (MPSO), and (3) the standard Salp Swarm Optimization (SSO) by considering above conflicting scheduling parameters. It is obvious from the results that the proposed algorithm outperforms over other compared algorithms.

In this contribution, authors have considered a binary version of the self-adaptive salp swarm optimization algorithm to address the load balancing and task scheduling problems, respectively. The self-adaptive variant is incorporated into the standard SSO algorithm to generate the population size automatically. Hence, users will not take the burden of determining the population size randomly. Besides, an elitism technique is used, when the size of the new population varies according to the size of the old population. As load balancing and task scheduling are binary in nature, a binary SASSO is implemented to transform the continuously generated solutions to binary solutions. A load balancing approach based on the similarity is executed before the initiation of the task scheduling procedure to ensure a balanced state. The random number \( r_1 \) used in the Equation 1 is calculated in a way that it maintains the trade-off between exploration and exploitation. Due to this value, at first, all the particles evade through the local optima and converge to the global optima slowly at the end.

The figures have portrayed the influence of makespan, response time, VM utilization, and degree of load imbalance for the disparate number of tasks and VMs. 10 independent executions are conducted in that environment, and the mean values of 10 independent executions for the above considered QoS parameters are recorded. The reason for taking mean values is each execution gives rise to a different set of values for the considered objectives. Figure 1 depicts the influence of makespan for the considered tasks×VMs set. It has been observed from the figure that the proposed algorithm significantly reduces overall makespan, thereby resulting in a better performance. This metric needs to be minimized in order to maximize user satisfaction. Figure 2 presents the impact of Response Time for the considered algorithms. Users’ request should get serviced as soon as possible by the cloud service provider. Therefore, the response time should be minimized to deliver good QoS to users. Figure 3 draws the influence of VM utilization for the considered dynamic tasks set. This parameter needs to be improved in order to maximize the overall performance of the system. The effectiveness of cloud-based computing totally relies on the efficient utilization of its resources. Moreover, utilization of resources greatly depends on the balanced state of the VMs. Hence, an efficient load balancing technique needs to be implemented to maintain the trade-off among VMs, thereby causing the improvement in the makespan, response time, and VM utilization. Figure 4 depicts the degree of load imbalance over dynamic tasks and VMs. It has been evident from the Figure 3 and Figure 4 that the proposed algorithm outperforms other considered algorithms for VM utilization and load-balancing. Figure 5 shows the comparison of the makespan between before and after implementing the proposed algorithm. It is obvious from the figure that makespan is significantly reduced corresponding to the dynamic tasks. Table 4 presents the impact of makespan over Tasks×VM instances and 5 presents the impact of response time over tasks×VM instances. Table 6 presents the impact of VM utilization over Tasks×VM instances. Table 7 presents the impact of load-balancing over Tasks×VMs instances. Table 8 presents the comparison of makespan before and after load balancing using BSASSO.
Table 4. Impact of Makespan over Tasks × VMs instances

| ALGORITHMS | MAKESPAN |
|------------|----------|
|             | Set 1    | Set 2    | Set 3    | Set 4    | Set 5    | Set 6    |
|             | (500×50) | (1000×100)| (1500×150)| (2000×200)| (2500×250)| (3000×300)|
| BSO         | 85.2     | 112.4    | 161.9    | 187.64   | 211.56   | 250.61   |
| MPSO        | 59.61    | 105.3    | 147.32   | 163.26   | 202.83   | 232.82   |
| SSO         | 58.21    | 102.4    | 154.82   | 169.2    | 205.87   | 231.62   |
| BSASSO      | 52.81    | 93.72    | 143.2    | 158.6    | 194.82   | 222.78   |
Table 5. Impact of Response Time over Tasks × VMs instances

| ALGORITHMS | Set 1 (500×50) | Set 2 (1000×100) | Set 3 (1500×150) | Set 4 (2000×200) | Set 5 (2500×250) | Set 6 (3000×300) |
|------------|----------------|------------------|------------------|------------------|------------------|------------------|
| BSO        | 1.35E+05       | 3.09E+05         | 3.25E+05         | 3.89E+05         | 4.90E+05         | 6.24E+05         |
| MPSO       | 1.23E+05       | 2.07E+05         | 2.12E+05         | 1.59E+06         | 4.22E+05         | 3.59E+06         |
| SSO        | 1.29E+05       | 2.82E+05         | 2.69E+05         | 3.15E+05         | 4.48E+05         | 5.64E+05         |
| BSASSO     | 1.16E+05       | 1.90E+05         | 2.05E+05         | 3.00E+05         | 3.98E+05         | 5.02E+05         |

Figure 2. Impact of Response Time over Tasks × VMs instances

![Graph showing response time for different algorithms and set sizes](image)

Table 6. Impact of VM Utilization over Tasks × VMs instances

| ALGORITHMS | Set 1 (500×50) | Set 2 (1000×100) | Set 3 (1500×150) | Set 4 (2000×200) | Set 5 (2500×250) | Set 6 (3000×300) |
|------------|----------------|------------------|------------------|------------------|------------------|------------------|
| BSO        | 7.33           | 8.22             | 10.92            | 15.21            | 8.28             | 9.45             |
| MPSO       | 13.11          | 13.47            | 21.38            | 24.31            | 22.35            | 30.92            |
| SSO        | 11.6           | 12.82            | 19.81            | 25.02            | 21.98            | 29.82            |
| BSASSO     | 16.47          | 23.55            | 25.62            | 31.13            | 43.69            | 49.42            |
Figure 3. Impact of VM Utilization over Tasks×VMs instances

![Figure 3: Impact of VM Utilization over Tasks×VMs instances](image)

Table 7. Impact of Degree of Load Imbalance over Tasks×VMs instances

| ALGORITHMS | DEGREE OF IMBALANCE |
|------------|---------------------|
|            | Set 1 (500×50)      | Set 2 (1000×100) | Set 3 (1500×150) | Set 4 (2000×200) | Set 5 (2500×250) | Set 6 (3000×300) |
| BSO        | 58.62               | 89.9             | 152.45           | 177.62           | 198.24           | 215.42           |
| MPSO       | 52.42               | 80.14            | 137.41           | 161.78           | 190.24           | 206.7            |
| SSO        | 53.31               | 82.69            | 139.81           | 163.46           | 191.26           | 207.77           |
| BSASSO     | 51.38               | 73.2             | 129.26           | 154.3            | 184.63           | 199.98           |

Figure 4. Impact of Degree of Load Imbalance over Tasks×VMs instances

![Figure 4: Impact of Degree of Load Imbalance over Tasks×VMs instances](image)
Table 8. Comparison of Makespan before and after load balancing using BSASSO

| NUMBER OF TASKS | MAKESPAN BEFORE | MAKESPAN AFTER |
|-----------------|-----------------|----------------|
| 500             | 112.4           | 49.21          |
| 1000            | 161.9           | 89.61          |
| 1500            | 202.64          | 138.58         |
| 2000            | 249.56          | 151.02         |
| 2500            | 295.61          | 186.72         |
| 3000            | 362.23          | 217.63         |

Figure 5. Comparison of Makespan before and after load balancing using BSASSO

6. CONCLUSION AND FUTURE SCOPE

In this piece of work, a load balancing technique in the dynamic cloud milieu based on the swarming behavior of the salps has been proposed. Migrating the tasks from the overloaded virtual machines (VMs) in the cloud datacenter to the underloaded VMs, is analogous to the salps in the swarm being switched from an empty food patch to an abundantly available food source. This swarm optimization algorithm is one of the simplest metaheuristic approaches with less number of tuning parameters to implement. In any metaheuristic technique, it is cumbersome to determine and fix the number of population size because of the fluctuating nature of convergence. Hence, a self-adaptive approach has been incorporated in the proposed methodology that automatically determines the population...
size for the fluctuating tasks entering into the datacenter for execution. Originally the salp swarm optimization is a continuous phenomenon, whereas the tasks assignment to the virtual machines in cloud datacenter is a binary problem. In this regard, a sigmoid transfer function has been integrated to the salp swarm optimization algorithm in order to cater the purpose of solving the discrete problem of task scheduling. The tasks are assigned to the appropriate VMs in such a way that the overall load of the system remains balanced. This technique works well in heterogeneous cloud-based computing systems and is capable enough to handle the non-preemptive, independent, dynamic workloads. The proposed algorithm is compared with some of the recent metaheuristic approaches to evaluate its efficacy. After the simulation in CloudSim, it has been observed from the result analysis that the proposed methodology outperforms its counterparts in minimizing the makespan, response time and degree of imbalance, while maximizing the resource utilization and throughput.

In future, this work can be extended to consider the priority of the tasks, while scheduling to the respective VMs, keeping the overall system load balanced. Inter datacenter task migration and cross-zone load balancing may be incorporated to make the methodology more relevant to real cloud scenario. Fault tolerance mechanism can be included to make the cloud service more reliable and robust. In current scenario, the energy consumption in cloud datacenters along with load balancing is also an emerging area of research to explore.
REFERENCES

AWS. (2021, September 16). https://docs.aws.amazon.com/ elasticloadbalancing/latest/classic/enable-disable-crosszone-lb.html

Bilgaiyan, S., Sagnika, S., & Das, M. (2015). A Multi-Objective Cat Swarm Optimization Algorithm for Workflow Scheduling in Cloud Computing Environment. *Intelligent Computing, Communication and Devices*, 73-84. 10.1007/978-81-322-2012-1_9

Calheiros, R., Ranjan, R., Beloglazov, A., de Rose, C., & Buyya, R. (2011). Cloudsim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software, Practice & Experience, 41*(1), 23–50. Advance online publication. doi:10.1002/spe.995

Chaharsoghi, S., & Kermani, A. (2008). An Effective Ant Colony Optimization Algorithm (ACO) for Multi-Objective Resource Allocation Problem (morap). *Applied Mathematics and Computation, 200*(1), 167–177. doi:10.1016/j.amc.2007.09.070

Dasgupta, K., Mandal, B., Dutta, P., Mandal, J. K., & Dam, S. (2013). A Genetic Algorithm (GA) Based Load Balancing Strategy for Cloud Computing. *Procedia Technology, 10*, 340–347. doi: 12.36910.1016/j.protcy.2013

de Mello, R., Senger, L., & Yang, L. (2006). A Routing Load Balancing Policy for Grid Computing Environments. 20th *International Conference on Advanced Information Networking and Applications*. doi:10.1109/AINA.2006.54

Ebadifard, F., & Babamir, S. M. (2018). A Pso Based Task Scheduling Algorithm Improved Using a Load Balancing Technique for the Cloud Computing Environment. *Concurrency and Computation, 30*(12), e4368. doi:10.1002/cpe.4368

Garg, R., & Shukla, N. (2018). Energy Efficient Scheduling for Multiple Workflows in Cloud Environment. *International Journal of Information Technology and Web Engineering, 13*(3), 14–34. doi:10.4018/ IJITWE.2018070102

Hussain, A., & Aleem, M. (2018). *GOCJ: Google Cloud Jobs Dataset*. Mendeley., doi:10.17632/b7bp6xhrcd.1

Hussain, A., & Aleem, M. (2018). *GOCJ: Google Cloud Jobs Dataset for Distributed and Cloud Computing Infrastructures*. MDPI., doi:10.3390/data3040038

Ibarra, O., & Kim, C. (1977). Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the Association for Computing Machinery, 24*(2), 280–289. doi:10.1145/322003.322011

Josep, A., Katz, R., Konwinski, A., & Gunho, L. (2010). A View of Cloud Computing. *Communications of the ACM, 53*(4), 50–58. doi:10.1145/1721654.1721672

Kalra, M., & Singh, S. (2015). A Review of Metaheuristic Scheduling Techniques in Cloud Computing. *Egyptian Informatics Journal, 16*(3), 275–295. doi:10.1016/j.eij.2015.07.001

Kaur, G., & Kaur, K. (2017). An Adaptive Firefly Algorithm for Load Balancing in Cloud Computing. *Proceedings of Sixth International Conference on Soft Computing for Problem Solving*. doi:10.1007/978-981-10-3322-3_7

Kaur, G., & Sharma, E. (2014). Optimized Utilization of Resources Using Improved Particle Swarm Optimization Based Task Scheduling Algorithms in Cloud Computing. *International Journal of Emerging Technology and Advanced Engineering, 4*(6), 110–115. 10.1.1.640.8685

Khalili, A., & Babamir, S. (2017). Optimal Scheduling Workflows in Cloud Computing Environment Using Pareto Based Grey Wolf Optimizer. *Concurrency and Computation, 29*(11), e4044. doi:10.1002/cpe.4044

Khan, I., Alghamdi, A., Jumani, T., Alamgir, A., Awan, A., & Khidrani, A. (2019). Salp Swarm Optimization Algorithm-Based Fractional Order PID Controller for Dynamic Response and Stability Enhancement of an Automatic Voltage Regulator System. *Electronics (Basel)*, 8(12), 1472. doi:10.3390/electronics8121472

Khorsand, R., Ghobaei Arani, M., & Ramezanpour, M. (2019). A Self Learning Fuzzy Approach for Proactive Resource Provisioning in Cloud Environment. *Software, Practice & Experience, 49*(11), 1618–1642. doi:10.1002/ spe.2737
Kumar, N., & Kumar, S. (2021). A Salp Swarm Optimization for Dynamic Resource Management to Improve Quality-of-Service in Cloud Computing and IoT Environment. *International Journal of Sensors, Wireless Communications and Control, 11*. Advance online publication. doi:10.2174/2210327911666210126122119

Li, D., & Krishna, P. (2013). Honey Bee Behavior Inspired Load Balancing of Tasks in Cloud Computing Environments. *Applied Soft Computing, 13*(5), 2292–2303. doi:10.1016/j.asoc.2013.01.025

Li, K., Xu, G., Zhao, G., Dong, Y., & Wang, D. (2011). Cloud Task Scheduling Based on Load Balancing Ant Colony Optimization. In *2011 Sixth Annual Chinagrid Conference* (pp. 3-9). IEEE. doi:10.1109/ChinaGrid.2011.17

Mandal, T., & Acharyya, S. (2015). Optimal Task Scheduling in Cloud Computing Environment: Meta Heuristic Approaches. In *2nd International Conference on Electrical Information and Communication Technologies (EICT)* (pp. 24–28). IEEE. doi:10.1109/EICT.2015.7391916

Mapetu, J., Chen, Z., & Kong, L. (2019). Low-Time Complexity and Low-Cost Binary Particle Swarm Optimization Algorithm for Task Scheduling and Load Balancing in Cloud Computing. *Applied Intelligence, 49*(9), 3308–3330. doi:10.1007/s10489-019-01448-x

Mell, P., & Grance, T. (2011). *The Nist Definition of Cloud Computing*. 10.6028/NIST.SP.800-145

Milan, S., Rajabion, L., Ranjbar, H., & Navimipour, N. (2019). Nature Inspired Meta-Heuristic Algorithms for Solving the Load-Balancing Problem in Cloud Environments. *Computers & Operations Research, 110*, 159–187. doi:10.1016/j.cor.2019.05.022

Mirjalili, S., Gandomi, A. H., Mirjalili, S. Z., Sareni, S., Faris, H., & Mirjalili, S. M. (2017). Salp Swarm Algorithm: A Bio-Inspired Optimizer for Engineering Design Problems. *Advances in Engineering Software, 114*, 163–191. doi:10.1016/j.advengsoft.2017.07.002

Mishra, K., & Majhi, S. K. (2021). A Binary Bird Swarm Optimization Based Load Balancing Algorithm for Cloud Computing Environment. *Open Computer Science, 11*(1), 146–160. doi:10.1515/comp-2020-0215

Mishra, S., Sahoo, B., & Parida, P. (2020). Load Balancing in Cloud Computing: A Big Picture. *Journal of King Saud University-Computer and Information Sciences, 32*(2), 149–158. doi:10.1016/j.jsuci.2018.01.003

Mittal, S., & Katal, A. (2016). An Optimized Task Scheduling Algorithm in Cloud Computing. In *IEEE 6th International Conference on Advanced Computing (IACC)* (pp. 197–202). IEEE. doi:10.1109/IACC.2016.45

Mohanty, S., Patra, P. K., Ray, M., & Mohapatra, S. (2019). An Approach for Load Balancing in Cloud Computing Using Jaya Algorithm. *International Journal of Information Technology and Web Engineering, 14*(1), 27–41. doi:10.4018/IJITWE.2019010102

Mohapatra, H., & Rath, A. (2019). Detection and Avoidance of Water Loss Through Municipality Taps in India by Using Smart Taps and ICT. *IET Wireless Sensor Systems, 9*(6), 447–457. doi:008110.1049/iet-wss.2019

Mohapatra, H., & Rath, A. (2019). Fault Tolerance in WSN Through Pe-Leach Protocol. *IET Wireless Sensor Systems, 9*(6), 358–365. doi:10.1049/iet-wss.2018.5229

Mohapatra, H., & Rath, A. (2020). Survey on Fault Tolerance-Based Clustering Evolution in WSN. *IET Network, 9*(4), 145–155. doi:10.1049/iet-net.2019.0155

Nanduri, R., Maheshwari, N., Reddyraja, A., & Varma, V. (2011). Job Aware Scheduling Algorithm for Mapreduce Framework. In *Third International Conference on Cloud Computing Technology and Science* (pp. 724-729). IEEE. doi:10.1109/CloudCom.2011.112

Nayak, S. C., & Tripathy, C. (2019). An Improved Task Scheduling Mechanism Using Multi-Criteria Decision Making in Cloud Computing. *International Journal of Information Technology and Web Engineering, 14*(2), 92–117. doi:10.4018/IJITWE.2019040106

Polepally, V., & Chatrpati, K. (2019). Dragonfly Optimization and Constraint Measure-Based Load Balancing in Cloud Computing. *Cluster Computing, 22*(S1), 1–13. doi:10.1007/s10586-017-1056-4

Rafieyan, E., Khorsand, R., & Ramezanpour, M. (2020). An Adaptive Scheduling Approach Based on Integrated Best-Worst and Vikor for Cloud Computing. *Computers & Industrial Engineering, 140*, 106–272. doi:10.1016/j.cie.2020.106272
Rizk-Allah, R., Hassanien, A., Elhoseny, M., & Gunasekaran, M. (2019). A New Binary Salp Swarm Algorithm: Development and Application for Optimization Tasks. *Neural Computing & Applications, 31*(5), 1641–1663. doi:10.1007/s00521-018-3613-z

Salgotra, R., Singh, U., Singh, S., Singh Gill, G., & Mittal, N. (2020). Self-Adaptive Salp Swarm Algorithm for Engineering Optimization Problems. *Applied Mathematical Modelling, 188*–207. doi:.2020.08.01410.1016/j.apm

Shah, N., & Farik, M. (2015). Static Load Balancing Algorithms in Cloud Computing: Challenges & Solutions. *International Journal of Scientific & Technology Research, 4*(10), 365–367.

Sharma, S., Rath, A., & Parida, B. (2020). Efficient Load Balancing Techniques for Multi-Datacenter Cloud Milieu. *Int. J. Inf. Tecnol. 10.1007/s41870-020-00529-2*

Singh, P., Dutta, M., & Aggarwal, N. (2017). A Review of Task Scheduling Based on Meta-Heuristics Approach in Cloud Computing. *Knowledge and Information Systems, 52*(1), 1–51. doi:10.1007/s10115-017-1044-2

Ullman, J. (1975). NP-Complete Scheduling Problems. *Journal of Computer and System Sciences, 10*(3), 384–393. doi:10.1016/S0022-0000(75)80008-0

**Bivasa Ranjan Parida is a Ph.D. Scholar at Veer Surendra Sai University of Technology, Burla, India.**

Amiya Kumar Rath received the B.E. degree in computer from Dr. Babasaheb Ambedkar Marathwada University, Aurangabad, in 1990, the M.B.A. degree in systems management from Shivaji University in 1993, the MTech. degree in computer science from Utkal University in 2001, and the Ph.D. degree in computer science from Utkal University, in 2005, with a focus on embedded systems. He is currently a Professor with the Department of Computer Science and Engineering, Veer Surendra Sai University of Technology, Burla, India. He has contributed over 80 research level papers to many national and international journals and conferences. He has published seven books by reputed publishers. His research interests include embedded systems, ad hoc networks, sensor network, power minimization, evolutionary computation, and data mining. Currently, he has been deputed as an adviser to National Assessment and Accreditation Council (NAAC), Bangalore, India.

Hitesh Mohapatra received the B.E. degree in Information Technology from the Biju Patnaik University of Technology, Odisha in 2006, the MTech degree in CSE from Biju Patnaik University of Technology, Odisha in 2009. He has received his Ph.D. in Computer Science & Engineering in 2021 from Veer Surendra Sai University of Technology, Burla, India. He has contributed 20 SCI and Scopus indexed research papers, 15 international/ national conferences, and 2 books on Software Engineering and C Programming respectively. He has 11 years of teaching experience both in industry and academia. His research interests include wireless sensor networks, smart city, smart grid, and smart water.