Computing and Informatics, Vol. 42, 2023, 364–391, doi: 10.31577/cai_2023_2_364

AN APPROACH BASED ON GENETIC AND GRASSHOPPER OPTIMIZATION ALGORITHMS FOR DYNAMIC LOAD BALANCING IN CLOUDIOT

Sofiane Benabbes, Sofiane Mounine Hemam

ICOSI Laboratory, Abbes Laghrour University
Khenchela, Algeria
e-mail: {benabbes.sofiane, hemam.sofiane}@univ-khenchela.dz

Abstract. *CloudIoT* is a new paradigm, which has emerged as a result of the combination of Cloud Computing (CC) and the Internet of Things (IoT). It has experienced a growing and rapid development, and it has become more popular in information and technology (IT) environments because of the advantages it offers. However, due to a strong use of this paradigm, especially in smart cities, the problem of imbalance load has emerged. Indeed, to satisfy the needs of the user, the intelligent objects send the collected data to the virtual machines (VMs) of the cloud in order to be processed. So, it is necessary to have an idea about the load of its VM. Thus, the problem of load balancing between VMs is strongly related to the technique used for the VMs selection. To tackle this problem, we propose in this paper a task scheduler called Scheduler Genetic Grasshopper Algorithm (SGGA). It allows to ensure a dynamic load balancing, as well as the optimization of the makespan and the resource usage. Our proposed SGGA is based on the combination of Genetic Algorithm (GA) and Grasshopper Optimization Algorithm (GOA). First, the tasks sent by the IoTs are mapped to the VMs in order to build the initial population, then SGGA performs the genetic algorithm, which has expressed a considerable performance. However, the weakness of the GA is marked by its heaviness caused by the mutation operator, especially when the number of tasks increases. Because of this insufficiency, we have replaced the mutation operator with the grasshopper optimization algorithm. The results of the experiments show that our approach (SGGA) is the most efficient, compared to the recent approaches, in terms of the response time to obtain the optimal solution, makespan, throughput, an average resource utilization rate and the hypervolume indicator.

Keywords: CloudIoT, dynamic load balancing, GA, GOA, task scheduler

1 INTRODUCTION

Nowadays, IoT and Cloud Computing are two new distributed computing technologies. On the one hand, the IoT allows to transform real-world objects into smart objects [20], which share their data, their situations and their interactions with other interconnected objects. The IoT is generally characterized by widely distributed objects with limited processing and storage capabilities. These objects suffer from performance, reliability, privacy and security issues [4]. On the other hand, Cloud Computing is a technology that has a network with unlimited storage capacities and computing power. Moreover, it offers the flexibility and robustness of dynamic data integration from heterogeneous sources [19].

CloudIoT is a new paradigm that has emerged as a result of the combination of cloud and IoT. It enables intelligent use of applications, information and infrastructure in a fair and reasonable manner. Although IoT and Cloud Computing are two different technologies, their functionalities are almost complementary [17], in terms of nature of existence, processing capacity, storage capacity, connectivity and Big Data [19].

In this environment, IoTs send their tasks to Cloud Computing for processing or storage, by mapping them to the various hardware and software resources represented by virtual machines. When distributing data to be processed to virtual machines (VMs), some of them will be overloaded while others will be unloaded or inactive [20, 11]. Thus, a load balancing mechanism is then necessary because it allows to manage the allocation of VMs to tasks sent by IoTs. It thus allows the optimization of makespan, throughput and the rentable resource usage.

Several scientific research on load balancing in Cloud Computing has focused on task allocation. Each scheduling algorithm is based on one or more parameters. The most targeted objectives are the overall execution time, the cost, and the use of resources (which also indicates the quality of service (QoS)) [22, 7, 9]. Several task scheduling algorithms based on metaheuristic algorithms, such as BGA, HGOW-ABC, GWO and ACO algorithms have been proposed for cloud load balancing [9, 14, 23, 25]. The researchers have developed techniques to reduce makespan, and assign tasks to VMs in a balanced way, using different optimization techniques such as genetic algorithm, gray wolf algorithm, the bee colony algorithm and the ant colony algorithm.

To deal with the above cited problem, we propose in this work a scheduler which, ensures a dynamic load balancing in the CloudIoT. The proposed work allows the improvement of the makespan, the throughput and the average rate of the resource usage. Our proposed scheduler allows the tasks distribution, sent by the IoTs, over the virtual machines. It ensures maximum throughput with a shorter execution time, as well as a more efficient use of resources. Our proposed approach called SGGA, is based on the combination between genetic algorithm and grasshopper optimization algorithm. So, the proposed selection and crossing operators phases of our approach allow to avoid the appearance of the same chromosome several times on one hand, and to avoid the heaviness caused by the mutation operator phase [18],

especially when the number of tasks increases, we have replaced this phase by the grasshopper optimization algorithm on the second hand.

The results obtained show that the proposed approach is more efficient compared to the most recent works (BGA, HGOW-ABC, GWO and ACO) in terms of the time to reach the optimal solution, the makespan, the throughput, the average resource utilization ratio and the hypervolume indicator.

Our paper is structured as follows: after the introduction, Section 2 is devoted to related works. In Section 3 we present the proposed approach and its different components. Section 4 presents case studies. Section 5 is reserved for experimental results and discussion, and we end this paper with a conclusion in Section 6.

2 RELATED WORKS

In this section, we review the most recent works that address the problem of load balancing in the cloud environment. The majority of the proposed works are mainly based on task scheduling to achieve the objective of ensuring load balancing between different components. Metaheuristic algorithms are classified into four categories [27, 28]: Firstly, the swarm-intelligence algorithms such as [30] which proposed a new metaheuristic algorithm called Giant Trevally Optimizer (GTO) inspired by the hunting behaviour of giant trevally.

Secondly, human-based algorithms such as [31] which proposed a metaheuristic algorithm called Group Teaching Optimisation Algorithm (GTOA), where they adjusted additional control parameters for solving different optimisation problems. Thirdly, evolutionary algorithms such as [32] have proposed a new approach called the Tree Growth Algorithm (TGA). This approach is inspired by the competition of trees for light and food. And fourthly, science-based algorithms such as [28] who have proposed a new metaheuristic called Crystal Structure Algorithm (CryStAl). This approach is inspired by the principles underlying the formation of crystal structures from the addition of the base to lattice points. In this paper we focus on the first two categories for their impressive results when compared to each other.

Several swarm-intelligence algorithms and human-based algorithms have been proposed and applied for task scheduling in the cloud environment. There are two types of these algorithms:

1. based on the exploitation of the best solution among the previous results, called a local search, and
2. based on the exploration of new areas of the solution space or the sudden prospection of a new solution search space.

The most interesting work in this context is reviewed below. So, at first we present the human-based algorithms category, followed by the swarm-intelligence algorithms category.

Makasarwala and Hazari [24], Kaur and Sachdeva [22] used GA for load balancing in Cloud Computing. The proposal of Makasarwala and Hazari [24] provides load balancing and reduces response time without considering resource utilization rate and QoS. On the other hand, Kaur and Sachdeva [22] proposed an improved GA to reduce the execution time of task migration in Cloud Computing. This proposal not only ensures the proper use of resources, but it also saves energy. However, the response time is high.

Gulbaz et al. [9] presented the Balancer Genetic Algorithm (BGA) to improve makespan and load balancing. BGA relies on a load balancing mechanism that takes into account the actual load assigned to virtual machines. The need to opt for multi-objective optimization for the improvement of load balancing and Makespan is also highlighted. The simulation showed significant improvement on makespan, throughput and load balancing.

For the swarm-intelligence algorithms category, we can find several works. We cite in this paper the most recent and important ones.

The work presented by Li and Wu [10]; Shafahi and Yari [25] used the Ant Colony Algorithm (ACO) to dynamically schedule tasks. The scheduler acts as an ant looking for food. The experimental results of the simulations, of the two approaches, give better performance in comparison to others. They reduce task execution time and improve system resource utilization, and they keep the system balanced.

Muthsamy and Suganthe [12] and Shen et al. [26] used the Artificial Bee Colony Optimization (ABC) algorithm. Thereby, Muthsamy and Suganthe [12] proposed a task scheduler based on optimizing artificial bee foraging (TSABF) that takes in charge the QoS, makespan, response time, execution time and task priority. To achieve optimal scheduling, tasks are scheduled preemptively. Task preemption is done to reduce the response time and execution time of tasks belonging to different priorities. While the work of Shen et al. [26] presented a study to ensure load balancing in a cloud data center, based on efficient resource utilization and power consumption management. They have optimized the (ABC) method using a load balancing algorithm, and intelligent classification of virtual machines. This study was validated by a simulation on CloudSim.

Arulkumar [3] obtained their best simulation results from the Water Wave Algorithm (WWA). The latter was proposed for resource planning in a cloud environment. The proposed work takes into consideration the four parameters: throughput, response time, resource utilization, and scalability.

Alguliyev et al. [1] presented a novel multi-criteria optimization method for weighted task scheduling based on the Particle Swarm Optimization (PSO) algorithm. The simulation showed that the method migrates tasks from overloaded virtual machines to less loaded virtual machines, ensuring, thus, an overall balanced system.

Patel et al. [23] proposed a task allocation approach based on gray wolf optimization (GWO) for load balancing in the containerized cloud. The approach ensures the load balancing and makespan minimization. Gohil and Patel [21] proposed (IGWO)

which is an improvement of the GWO algorithm. They increased the coefficients of the best solutions α, β and δ to calculate the next solutions, which gives a perfect balance and guarantees a quasi-optimal solution.

Natesan and Chokkalingam [13] also improved (GWO). They proposed Performance-Cost Grey Wolf Optimization (PCGWO) to reduce both processing time and cost in accordance with the objective function. The simulation results of the proposed technique show a complete reduction in the time and cost of performing the tasks.

Ragmani et al. [15] proposed a hybrid algorithm, based on the concepts of Fuzzy Logic and Ant Colony Optimization (Fuzzy-ACO), to improve load balancing in Cloud Computing. This approach takes into account load balancing goals and response time. Simulations performed on CloudAnalyst have shown that the proposed approach improves load balancing in the Cloud, minimizing response time by up to 82 %, processing time by up to 90 % and total cost up to 9 %.

Ouhame et al. [14] integrated the GWO algorithm with Artificial Bee Colony (HGWOABC) to improve the cloud resource allocation system. This technique improved the parameters of load balancing in Cloud Computing by 1.25 %.

In Table 1, we conclude this section by specifying the different load balancing parameters of each approach.

Year	Approach	Makespan	Throughput	Res Utzt	QoS	Energy	Cost	HV
2016	[24]	Yes	No	No	No	No	No	No
2017	[22]	Yes	No	Yes	Yes	Yes	No	No
2019	[10]	Yes	No	Yes	Yes	No	No	No
2021	[25]	Yes	No	Yes	Yes	No	No	No
2020	[12]	Yes	Yes	Yes	Yes	No	No	No
2019	[26]	No	No	Yes	Yes	Yes	No	No
2021	[9]	Yes	Yes	Yes	Yes	No	No	No
2019	[1]	No	No	Yes	Yes	No	No	No
2020	[23]	Yes	Yes	No	No	No	No	No
2018	[21]	Yes	Yes	Yes	Yes	No	No	No
2019	[15]	Yes	No	Yes	Yes	No	Yes	No
2020	[14]	Yes	No	Yes	Yes	No	No	No

Table 1. Summary of balancing parameters for each approach

In this paper, a new dynamic load balancing approach has been proposed using a task scheduler based on the pairing between Genetic Algorithm (GA) and Grasshopper Optimization Algorithm (GOA) called Scheduler GA-GOA Algorithm (SGGA).

We have realized several hybridizations tests to replace the mutation by other algorithms, and the GA-GOA hybridization give the best result. As shown in the following Table 2 for example the makespan.

Our balancing based on the improvement of three parameters: makespan, throughput and resource utilization (QoS).

Tasks	GA	GA-GOA	GA-PSO	GA-ABC	GA-ANTLion	GA-ACO
100	0.32584	0.15874	0.32101	0.31524	0.18524	0.25471
200	0.78954	0.17543	0.78814	0.76214	0.20574	0.31241
500	1.01458	0.20145	1.01247	1.00024	0.26472	0.43120

Table 2. Testing the "makespan" result of the different GA hybrids

3 THE PROPOSED APPROACH

In this section, we will present the proposed global architecture; and the detailed architecture that contains the components of our scheduler.

3.1 The Overall Architecture Proposed

Our proposed approach is developed to provide load balancing in the CloudIoT. It contains several components on the IoT and Cloud sides. In our approach, we focus our interest in tasks that will be processed at the Cloud level. The smart objects send their tasks to the scheduler (SGGA) which will map and assign them to the different virtual machines. The role of the scheduler will be detailed in the next section. Figure 1 shows the overview of the overall architecture proposed in CloudIoT.

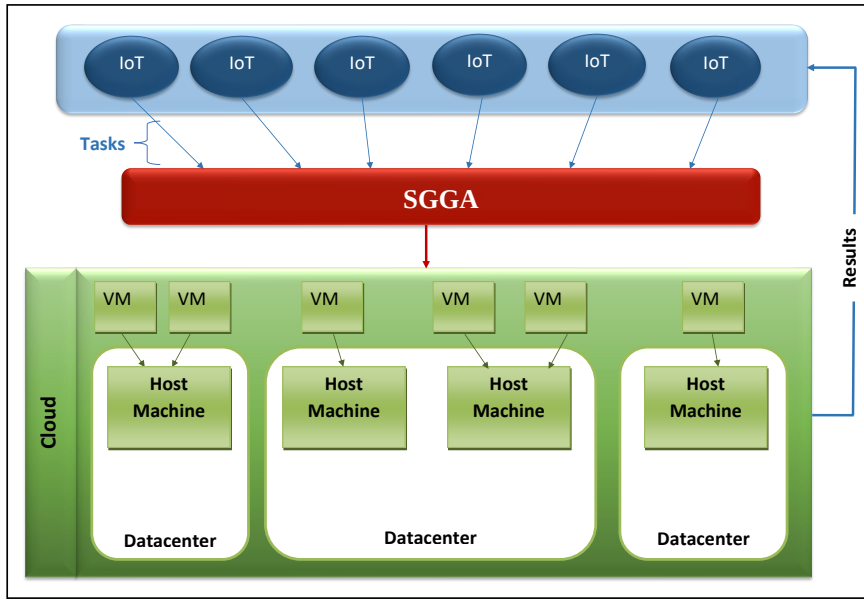


Figure 1. Presentation of the global architecture proposed in CloudIoT

3.2 Detailed Architecture of the Proposed Approach "SGGA"

An effective load balancing is relied to a robust and reliable task scheduler. To this end, we have developed a task scheduler based on both the grasshopper optimization algorithm and the genetic algorithm. The latter gives very satisfactory results mainly because of its flexibility and robustness. However, this algorithm suffers from a heaviness at the level of its mutation operator [18], especially when the tasks number is increased. For this reason, and in order to deal with this limitation, we propose to replace the mutation operator with the grasshopper optimization algorithm (see Figure 2). The latter is classified among the new meta-heuristic algorithms, and it is inspired by the behavior of grasshoppers [16].

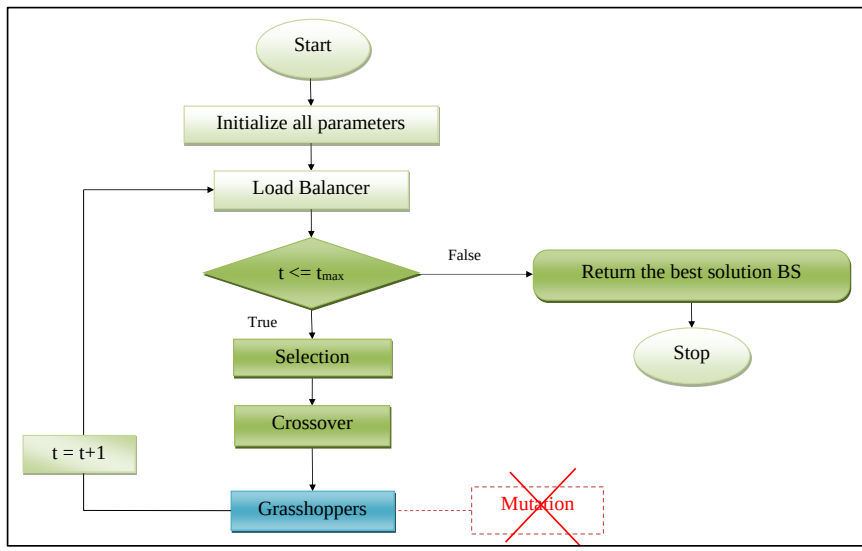


Figure 2. Flow diagram of the Scheduler GGA

The algorithm of the proposed approach; which will be detailed later, initializes randomly the position matrix. The rows and columns of this matrix are respectively solutions and tasks (Table 3).

The solutions number is equal to 100, and each matrix cell contains the CPU speed of a VM. After the initialization phase the SGGA, in each iteration, calculates the fitness function of each solution (row). then, it sorts the solutions in ascending order according to the calculated fitness functions. Thus, the best solution will be at the top of the population (100 solutions). To improve the population quality of the position matrix, we select the 7 % of the best solutions, and then we apply the selector and crossover operators of GA to obtain 50 % of the new populations (as indicated in Figure 3). At the end of an iteration, the grasshopper optimization algorithm is applied to obtain the second half-population (the second 50 %) from

M Positions	T1	T2	T3	T4	T5	T6	T7	...	Tm
P1	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	...	S-VM
P2	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	...	S-VM
P3	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	...	S-VM
P4	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	...	S-VM
P5	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	...	S-VM
P6	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	...	S-VM
P7	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	...	S-VM
...	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	...	S-VM
Pn	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	S-VM	...	S-VM

Table 3. The population Matrix loaded by CPU speed of VMs (S-VM)

the value of the first half-population.

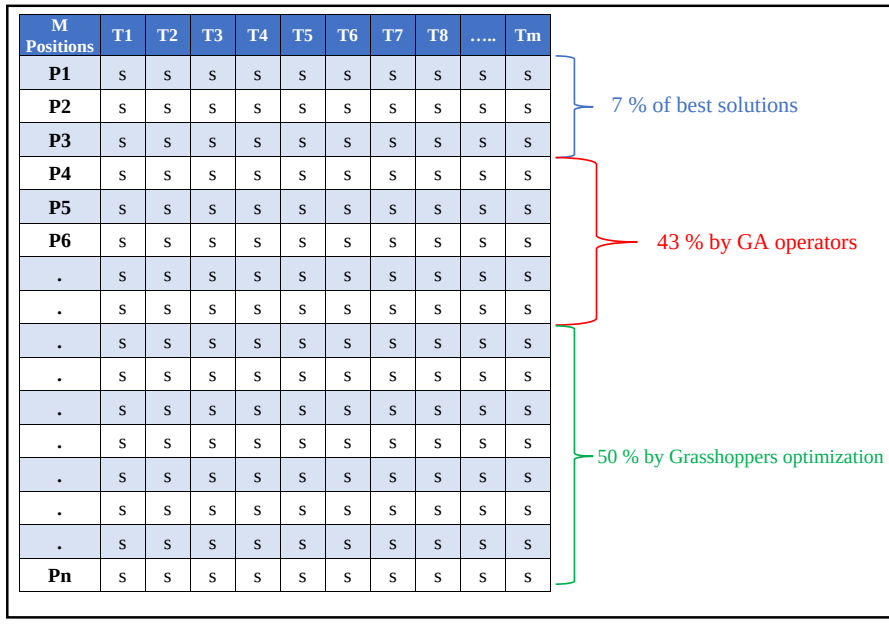


Figure 3. The new population Matrix composition

3.3 The Components of the SGGA

In this sub-section, we present, in details, the roles of the fourth components of our approach, as well as their algorithms.

Algorithm 1: SGGA

Input: Tasks vector, VMs vector
Output: Mapping of Tasks to VMs
$M[x, n] \leftarrow$ random(CPU, VMs);
$t \leftarrow 1$;
while $t \leq t_{max}$ **do**
  for $i \leftarrow 2$ *to* n **do**
    Fitness$[i] \leftarrow$ LoadBalancer$(M[x, n])$;
  end
  BS $\leftarrow M[1, n]$;
  newM $\leftarrow 1$;
  $j \leftarrow 1$;
  SelectionAlgo (Fitness$[x], M[x, n]$, M_Select$[y, n]$, Array_Select$[y^2 - y]$);
  for $i \leftarrow 1$ *to* y **do**
    $M[i, n] \leftarrow$ M_Select$[i, n]$;
  end
  while $newM < s$ **do**
    Parent1 $\leftarrow$ M_Select[Array_Select[newM].part1, n];
    Parent2 $\leftarrow$ M_Select[Array_Select[newM].part2, n];
    CrossoverAlgo(Parent1, Parent2, newParent1, newParent2);
    $M1[j, n] \leftarrow$ newParent1;
    $M1[j + 1, n] \leftarrow$ newParent2;
    newM $\leftarrow$ newM + 1;
    $j \leftarrow j + 2$;
  end
  while $newM \leq x$ **do**
    $M2[] \leftarrow$ GrasshoppersAlgo$(M1)$;
    newM $\leftarrow$ newM + 1;
  end
  $M[] \leftarrow M1[] + M2[]$; (concatenate the two matrices M1 and M2 in M);
  $t \leftarrow t + 1$;
end

In this context, we assume that we have a set of tasks $T = \{T_1, T_2, \ldots, T_n\}$, where each of them is characterized by its size in Kilobytes (KB), and a set of virtual machines $VM = \{VM_1, VM_2, \ldots, VM_m\}$, where each of them is characterized by its computing capacity (CPU) in million instructions per second (mips).

The population is represented by positions (solutions), each of them has its own fitness function. The solution can be represented using binary Mapping Matrix (MP), where rows indicate VMs and columns indicate Tasks. In the example below, the mapping matrix (MP) represents the set $VM_1(T_2, T_6)$, $VM_2(T_4, T_5)$, $VM_3(T_1, T_3)$.

$$\mathrm{MP} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{1}$$

SGGA starts by initializing the population randomly, each chromosome is represented by a speed line of the virtual machines and the indices are the indices of the tasks.

In the rest of the section, we will detail the role of each component of the proposed approach.

3.3.1 The Load Balancer Component

The role of this component is to calculate the fitness functions of each solution by using formulas from (2 to 10) then, it stores them in the fitness vector. After that, it sorts the fitness vector in ascending manner as well as the solutions matrix according to the fitness vector. Thus, the best solution (BS) will be at the top of the solutions matrix.

Since our SGGA is a multi-objective thus, to improve load balancing, the proposed fitness function must combine between two objectives (the makespan and the average load utilization) as indicated in the formula (2).

$$f(P_i) = (\mathrm{makespan} + \mathrm{AvgLoad}), \tag{2}$$

where makespan is the maximum time taken by any virtual machine, given by:

$$\mathrm{makespan} = (\max(T_{V_M})_{j\in 1,\ldots,M}), \tag{3}$$

where T_{VMj} is the processing time of a specific VM_j. AvgLoad is between 0 and 1. It is the average load of all virtual machines for a specific position, it is calculated according to the formula (4):

$$\mathrm{AvgLoad} = \left(1 - \frac{\sum_{j=1}^{M} \mathrm{Load}_{VMj}}{m}\right). \tag{4}$$

The value of Load_{VMj} is calculated to find the part used by the virtual machine VM_j, according to the following equation:

$$\mathrm{Load}_{VMj} = \left(\frac{\mathrm{TaskMap}_{VMj}}{\mathrm{Mappe}_{VMj}} * 100\right). \tag{5}$$

Formula (6) normalizes the value of Load_{VMj} to avoid negative values that affect AvgLoad: [9].

$$\text{Load}_{VMj} = \begin{cases} \frac{100-(Load_{VMj}-100)}{100}, & \text{if } \text{Load}_{VMj} > 100, \\ 0, & \text{if } \text{Load}_{VMj} < 0, \\ \frac{\text{Load}_{VMj}}{100}, & \text{else.} \end{cases} \tag{6}$$

The Equation (7) presents the size of the tasks mapped by the virtual machine VM_j, using the binary Mapping Matrix already presented in (1):

$$\text{TaskMap}_{VMj} = \left(\sum_{i=1}^{N} \text{SizeTask}[i] * \text{MP}[i,j] \right). \tag{7}$$

Since each task is characterized by its value size and, each virtual machine is characterized by its computing power thus, these values are used to calculate the load according to formula (8):

$$\text{Mappe}_{VMj} = \left(\sum_{i=1}^{N} (\text{SizeTask}[i]) * \text{MappeRatio}_{(VMj)} \right), \tag{8}$$

where $\text{MappeRatio}_{(VMj)}$ is a ratio used to calculate the maximum size of tasks that can be mapped to the VM_j, it is calculated as follows:

$$\text{MappeRatio}_{(VMj)} = \left(\frac{\text{CPU}_{VMj}}{\sum_{j=1}^{M} \text{CPU}_{VMj}} \right). \tag{9}$$

Finally, formula (10) is used to calculate the average resource utilization rate (AVG_{UR}). This rate is between [0.1] and it is calculated as follows:

$$\text{AVG}_{UR} = \left(\frac{(\sum_{j=1}^{M} T_{VMj})/M}{\text{makespan}} \right). \tag{10}$$

The pseudo Algorithm 2 **LoadBalancer** presents the different actions that allow to calculate the fitness function and the combination between the two objectives of our approach.

3.3.2 The Selector Component

This component generates another matrix, called M_Select, and a vector, called Array_Select. At the beginning, this matrix contains, the best 7 % chromosomes. The Array_Select vector contains the indices of all chromosomes that can be generated and used for crossing. This vector makes it possible to avoid the crossing

Algorithm 2: Load Balancer component behavior

Input: $M[x, n]$
Output: Fitness$[x]$, $M[x, n]$
Initialize parameters;
LoadMP[];
Calculate:
SizeTask$[i]$, CPUVM$[j]$;
Mappe$_{VM}[j]$;
TaskMap$_{VM}[j]$;
Load$_{VM}[j]$;
AvgLoad;
Makespan;
Fitness $\leftarrow$ Makespan + AvgLoad;
Descending sort(vector of fitness functions);
Descending sort(matrix of positions) according Fitness sort;

between the same chromosome on one side, and also to avoid the crossing of two chromosomes several times on the other side (Figure 4).

The pseudo Algorithm 3 **SelectorAlgo**, allows to present the different actions which constitute the functionalities of this component.

Algorithm 3: Selector component behavior

Input: Fitness$[x]$, $M[x, n]$
Output: MSelect$[y, n]$, ArraySelect$[y^2 - y]$
Select the 7 % of best solutions;
Create the matrix of best solutions;
Create the array of parent index who go to crossover;

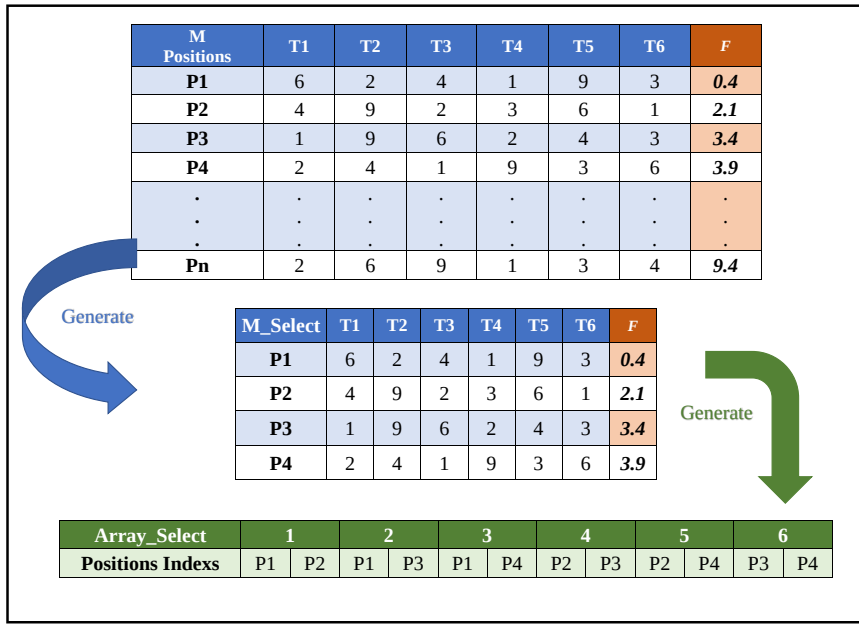
3.3.3 The Crossover Component

From the indices values of the Array_Select vector, this component performs the crossing between two parents. So, it randomly takes a series of genes from one parent and concatenates it with the remained genes from the second parent. At the end of the crossover operator execution, we obtain a new half-population (the first 50 % of the solutions).

The pseudo Algorithm 4 **CrossoverAlgo**, presents the actions of the crossing between two parents, to obtain new two parents.

3.3.4 The Grasshoppers Component

In genetic algorithm, to obtain a new generation, the mutation operator is based on random selection of genes, and the replacement of the latter by others closer to



M Positions	T1	T2	T3	T4	T5	T6	F
P1	6	2	4	1	9	3	***0.4***
P2	4	9	2	3	6	1	***2.1***
P3	1	9	6	2	4	3	***3.4***
P4	2	4	1	9	3	6	***3.9***
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
Pn	2	6	9	1	3	4	***9.4***

M_Select	T1	T2	T3	T4	T5	T6	F
P1	6	2	4	1	9	3	***0.4***
P2	4	9	2	3	6	1	***2.1***
P3	1	9	6	2	4	3	***3.4***
P4	2	4	1	9	3	6	***3.9***

Array_Select	1		2		3		4		5		6	
Positions Indexs	P1	P2	P1	P3	P1	P4	P2	P3	P2	P4	P3	P4

Figure 4. Relationship between M_Select and Array_Select

Algorithm 4: CrossoverAlgo

Input: Parent1, Parent2
Output: newParent1, newParent2
Randomly select part of the chromosome(cutPart);
Load the first cutPart of Parent1 and Parent2 to newParent1 and newParent2;
Load the rest of Parent1 and Parent2 to newParent2 and newParent1;

them. This allows a global optimal solution [9] to be found instead of a local optimal solution, and this is the strength of the genetic algorithm for global optimization.

The disadvantage of this operator is its heaviness [18], especially when the number of tasks increases. To overcome this problem, we replaced this operator by a component based on the grasshopper optimization algorithm, which allows to propose local optima [16]. The latter makes it possible to obtain the second half-population (a second 50 % of the solutions) from the first half-population. The obtained values of the second half-population will be normalized so that they correspond with the values of the CPU speeds of virtual machines. The normalization consists of bringing values of the matrix closer to those of the CPU speeds defined in the VMs vector. For example, an obtained value 6.3623, it will be normalized to 6 according to the list of CPU speeds VMs.

Finally, the two half-populations will be concatenated to form the new population. The advantage of this algorithm is that it offers meaningful exploration and exploitation [16].

In the remainder of this section, we explain the swarming behavior of grasshoppers which is mathematically modeled as follows:

$$P_i = (S_i + G_i + A_i)\,, \tag{11}$$

where P_i indicates the position of the grasshopper, S_i is the social interaction between the grasshoppers, G_i indicates the force of gravity on the grasshopper, and A_i is the advection of the wind. To produce random grasshopper behavior, Equation (11) can be rewritten as:

$$P_i = ((r_1 * S_i) + (r_2 * G_i) + (r_3 * A_i))\,, \tag{12}$$

where r_1, r_2 and r_3 are random numbers in the range $[0, 1]$. The social interaction S_i is defined as follows:

$$S_i = \left(\sum_{j=1, j\neq i}^{N} S(d_{ij})\widehat{d_{ij}} \right), \tag{13}$$

where N denotes the number of grasshoppers, $d_{ij} = |\mathbf{P}_j - \mathbf{P}i|$ defines the Euclidean distance between the i^{th} and the j^{th} grasshopper, and $\widehat{d}ij = \frac{\mathbf{P}_j - \mathbf{P}i}{dij}$ is a unit vector from the i^{th} to the j^{th} grasshopper, and S represents the social forces denoted by the following equation:

$$S(r) = \left(f * e^{\frac{-r}{l}} - e^{-r} \right), \tag{14}$$

where f and l are the attraction intensity and the attraction length scale respectively.

Improvements have been made to formula (12) so that it can be used to solve optimization problems, because grasshoppers quickly reach the comfort zone and the swarm does not converge on the objective [16].

$$P_i^d = c_1 \left(\sum_{j=1, j\neq i}^{N} c_2 \frac{ub_d - lb_d}{2} s(|P_j - P_i|) \frac{P_j - P_i}{d_{ij}} \right) + \widehat{T}_d, \tag{15}$$

where ub_d and lb_d respectively represent the upper and lower bounds in the d^{th} dimension (where d represents the objective number on the fitness function). $\widehat{T}_d$ denotes the best solution found so far in the d-dimensional space.

c_1 and c_2 are considered as a single parameter called c which is expressed in the following equation: [16].

$$c = \left(c_{max} - t \frac{c_{max} - c_{min}}{t_{max}} \right), \tag{16}$$

where c_{max} and c_{min} represent the maximum and minimum values of c, respectively, t is the current iteration, and t_{max} is the maximum number of iterations. The

algorithm coefficients are initialized as follows [16]: $c_{max} = 1$, $c_{min} = 0.00004$, $f = 0.5$ and $l = 1.5$.

The formulas (11), (12), (13), (14) and (15) are translated into pseudo Algorithm 5 **GrasshoppersAlgo**.

Algorithm 5: Grasshopper Algorithm

```
Input: M1[s, n]
Output: M2[x − s, n]
Initialize parameters;
while not last position do
    if i ≠ j then
        M2 ← P_i^d;
    end
end
while not last position do
    M2[] ← (M2[] ≈ CPUVM[]); /* Normalized matrix of positions */
end
```

4 CASE STUDY

In this part, we explain the steps to follow, of our approach, in order to achieve the objective. For this, we assume that we have an environment containing a set of 6 tasks $T = \{T_1, T_2, T_3, T_4, T_5, T_6\}$, and a set of 3 virtual machines $VM = \{VM_1, VM_2, VM_3\}$. The operation of the components of our approach is illustrated through the following steps:

Step 1: In this step, the SGGA prepares Tables 4 and 5 with the necessary information. Thus, Table 4, contains the characteristics of the tasks namely the size (Size) and the worst execution time (WCET) [29]. The Table 5 contains virtual machine information: CPU speed, and Storage capacity.

T	T1	T2	T3	T4	T5	T6
Size (KB)	6	2	7	12	3	1.5
WCET	4	2	4.5	5.3	2.3	1.1

Table 4. Tasks characteristics

VM	VM1	VM2	VM3
CPU (mips)	5	4	6
Storage	7	12	22

Table 5. VMs characteristics

We also assume that the size of the population is equal to 20 ($x = 20$), the maximum time for the execution of a given task is 100 ($t_{max} = 100$).

Once the two tables above are prepared, the SGGA randomly initializes the matrix by M chromosomes $[20, 6]$ as shown in Table 6.

M	T1	T2	T3	T4	T5	T6	F
P1	4	6	5	5	5	6	12.80
P2	5	5	4	6	5	6	8.40
P3	5	4	5	6	6	6	9.14
P4	5	4	6	4	4	4	11.39
P5	6	4	5	6	4	6	4.89
P6	4	6	4	5	6	6	9.69
P7	4	4	5	4	5	6	12.07
P8	6	6	4	5	5	4	7.87
P9	6	6	4	4	4	5	13.05
P10	6	6	4	6	5	6	6.53
P11	5	6	5	4	6	5	10.07
P12	5	4	5	6	4	6	8.74
P13	4	5	6	4	5	6	9.96
P14	6	4	5	4	6	5	7.68
P15	6	6	5	4	5	5	8.20
P16	4	5	6	4	6	6	9.98
P17	5	4	5	5	6	4	14.20
P18	5	5	4	6	6	6	8.94
P19	6	5	6	4	6	6	12.44
P20	4	6	5	6	5	4	7.39

Table 6. The matrix M before sorting with Fitness

Then, it invokes the LoadBalancer component to create the fitness vector. The latter contains the fitness of each $Pi \in [1, \ldots, 20]$. Finally, the matrix M will be sorted according to the values of the fitness vector (ascending order) as show in Table 7.

Thus, the best solution (BS $= (6, 4, 5, 6, 4, 6)$), having as fitness the smallest value, will be at the head of the matrix.

Step 2: In this step, the Selector component is invoked. It allows to create the MSelect matrix.

This latter contains, at the beginning, the 7 % best solutions. In this case study, they are the four best chromosomes sectioned from the position matrix (see Table 8). From the MSelect array, the Selector component creates the ArraySelect vector which contains the chromosomes indices that will participate in the crossover step, as shown in Table 9.

Step 3: Once step 2 is completed, the SGGA, invokes the Crossover component. The latter makes it possible to create the first half-population (matrix M1$[10, 6]$)

M	T1	T2	T3	T4	T5	T6	F
P1	6	4	5	6	4	6	4.89
P2	6	6	4	6	5	6	6.53
P3	4	6	5	6	5	4	7.39
P4	6	4	5	4	6	5	7.68
P5	6	6	4	5	5	4	7.87
P6	6	6	5	4	5	5	8.20
P7	5	5	4	6	5	6	8.40
P8	5	4	5	6	4	6	8.74
P9	5	5	4	6	6	6	8.94
P10	5	4	5	6	6	6	9.14
P11	4	6	4	5	6	6	9.69
P12	4	5	6	4	5	6	9.96
P13	4	5	6	4	6	6	9.98
P14	5	6	5	4	6	5	10.07
P15	5	4	6	4	4	4	11.39
P16	4	4	5	4	5	6	12.07
P17	6	5	6	4	6	6	12.44
P18	4	6	5	5	5	6	12.80
P19	6	6	4	4	4	5	13.05
P20	5	4	5	5	6	4	14.20

Table 7. The matrix M after sorting with Fitness

	T1	T2	T3	T4	T5	T6
P1	6	4	5	6	4	6
P2	6	6	4	6	5	6
P3	4	6	5	6	5	4
P4	6	4	5	4	6	5

Table 8. MSelect for the best chromosomes

as indicated in Table 10. The first half-population is obtained from the MSelect matrix and the MSelect vector resulting from step 2. Thus, the M1 matrix contains the 7 % of the population of the M-Select array, and the rest of the population is obtained by crossing chromosomes (Pi, Pj) of the ArraySelect vector.

Step 4: In this step, the Grasshoppers component uses the matrix M1 to create the second half-population and stores it in M2[10, 6]. For example, according to formula (15), the position P11 of M2 is calculated by taking into consideration

1	2	3	4	5	6
P1, P2	P1, P3	P1, P4	P2, P3	P2, P4	P3, P4

Table 9. ArraySelect for all positions chromosomes

	T1	T2	T3	T4	T5	T6
P1	6	4	5	6	4	6
P2	6	6	4	6	5	6
P3	4	6	5	6	5	4
P4	6	4	5	4	6	5
P5	6	4	4	6	5	6
P6	6	4	5	6	5	4
P7	6	4	5	4	6	5
P8	6	6	5	6	5	4
P9	6	6	4	6	6	5
P10	4	6	5	4	6	5

Table 10. The matrix of the first half population M1

the position P1 with all the positions Pi where $i \in [1, \ldots, 10]$ and the best position.

And so on for positions from P12 to P20. The obtained values from the M2 matrix, presented in Table 11, do not correspond to the values of the CPU speeds of the VMs as indicated in Table 5. The normalization operation makes it possible to bring the values of the M2 matrix closer to values of Table 5, as shown in Table 12.

M2	T1	T2	T3	T4	T5	T6
P11	6.36	3.74	5.10	5.80	3.80	5.79
P12	6.05	4.05	4.49	5.57	3.80	5.79
P13	6.05	4.26	5.10	5.57	4.41	5.79
P14	6.05	4.26	5.51	6.42	3.59	6.22
P15	5.64	4.26	5.10	5.57	3.59	5.79
P16	6.36	3.74	4.49	5.80	4.41	5.79
P17	6.36	4.26	5.10	6.42	3.80	5.79
P18	5.64	3.74	4.49	5.80	3.80	6.22
P19	5.64	3.74	4.59	6.42	3.59	6.40
P20	5.64	3.74	4.49	5.57	3.80	5.79

Table 11. The matrix of the second half population M2

Step 5: In the last step, the SGGA merges the two half-populations in the M matrix. The latter represents a new generation of the solutions for the first iteration ($t = 1$) as show in Table 13.

Then, it invokes the LoadBalancer component for the next iteration, as shown in Table 14, until reaching the maximum number of iterations ($t = 100$) as proposed at the beginning.

At the end of the last iteration ($t = 100$), the optimal solution is at the top of the M2 matrix, and it is indicated by the position P1: BS $= (6, 4, 5, 6, 4, 6)$.

M2	T1	T2	T3	T4	T5	T6
P11	6	4	5	6	4	6
P12	6	4	5	6	4	6
P13	6	4	6	6	4	6
P14	6	4	5	6	4	6
P15	6	4	5	6	4	6
P16	6	4	6	6	4	6
P17	6	4	5	6	4	6
P18	6	4	4	6	4	6
P19	6	4	6	6	4	6
P20	6	4	5	6	4	6

Table 12. The M2 normalized

M2	T1	T2	T3	T4	T5	T6	F
P1	6	4	5	6	4	6	19.89
P2	6	6	4	6	5	6	19.93
P3	4	6	5	6	5	4	28.33
P4	6	4	5	4	6	5	19.97
P5	6	4	4	6	5	6	21.09
P6	6	4	5	6	5	4	21.07
P7	6	4	5	4	6	5	22.01
P8	6	6	5	6	5	4	21.59
P9	6	6	4	6	6	5	20.16
P10	4	6	5	4	6	5	27.34
P11	6	4	5	6	4	6	19.89
P12	6	4	5	6	4	6	19.89
P13	6	4	6	6	4	6	20.22
P14	6	4	5	6	4	6	19.89
P15	6	4	5	6	4	6	19.89
P16	6	4	6	6	4	6	20.22
P17	6	4	5	6	4	6	19.89
P18	6	4	4	6	4	6	21.12
P19	6	4	6	6	4	6	20.22
P20	6	4	5	6	4	6	19.89

Table 13. The new matrix of positions M and vector of Fitness ($t = 1$)

5 EXPERIMENTAL RESULTS AND DISCUSSION

The purpose of this section is to verify the effectiveness of our approach. In order to validate our proposal, we used the CloudSim 3.0.3 simulator [6]. The latter is a framework used to model and simulate the Cloud Computing environment and services. It is developed by the CLOUDS Lab organization and is written entirely in Java.

M2	T1	T2	T3	T4	T5	T6	F
P1	**6**	**4**	**5**	**6**	**4**	**6**	**19.89**
P2	6	4	5	6	4	6	19.89
P3	6	4	5	6	4	6	19.89
P4	6	4	5	6	4	6	19.89
P5	6	4	5	6	4	6	19.89
P6	6	4	5	6	4	6	19.89
P7	6	4	5	6	4	6	19.89
P8	6	6	4	6	5	6	19.93
P9	6	4	5	4	6	5	19.97
P10	6	6	4	6	6	5	20.16
P11	6	4	6	6	4	6	20.22
P12	6	4	6	6	4	6	20.22
P13	6	4	6	6	4	6	20.22
P14	6	4	5	6	5	4	21.07
P15	6	4	4	6	5	6	21.09
P16	6	4	4	6	4	6	21.12
P17	6	6	5	6	5	4	21.59
P18	6	4	5	4	6	5	22.01
P19	4	6	5	4	6	5	27.34
P20	4	6	5	6	5	4	28.33

Table 14. The new matrix of positions M and vector of Fitness ($t = 1$)

The simulation environment is a "Dell inspiron" PC, equipped with an Intel(R) Core (TM) i7-3632QM CPU 2.20 GHz, 6 GB RAM, 1 TB hard drive, and it uses a Windows 10 operating system.

The experiment environment is as follows: we set the number of iterations to 1 000 ($t_{max} = 1\,000$), a single data center containing 30 hots machines, and 50 virtual machines. The overall memory of the host machine is 16.384 MB.

To evaluate the performance of our proposed approach, the experiments below are based on the following metrics:

1. the waiting time of the optimal solution (WTOS),
2. the maximum time required for the execution of a batch of tasks (makespan),
3. the throughput (AVGThroughput) which represents the average number of tasks executed per second and per iteration,
4. the average utilization resource ratio (AVGUR), and
5. the hypervolume indicator (HV).

The experimental results obtained are compared with the works closest to our approach, namely: BGA, HGWO-ABC, GWO and ACO. In the following experiments, we assume that we have a maximum set of 1 000 tasks.

Experiment 1: In this experiment, we will evaluate the waiting time of the optimal solution of our approach. Then compare the results obtained with the four approaches: BGA, HGWO-ABC, GWO and ACO.

In this experiment, we vary the number of iterations from 100 to 1 000 with a step of 100, and we set the number of tasks to 500. Then we observe the waiting time necessary to obtain the best solution. Through Figure 5, we can notice that our approach approximates the optimal solution at the end of iteration 670, while the two approaches BGA and HGWO-ABC stabilize respectively from iteration 780 and 900. The optimal solution of two last approaches namely ACO and GWO is reached during the last iteration (1 000).

From the above, we can see that our approach gives better results in terms of waiting time to reach the optimal solution as well as in terms of the number of iterations. This offers a very considerable economic gain.

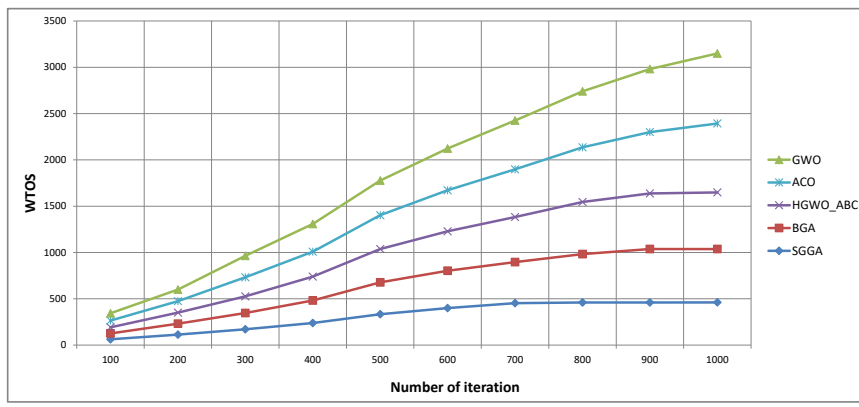


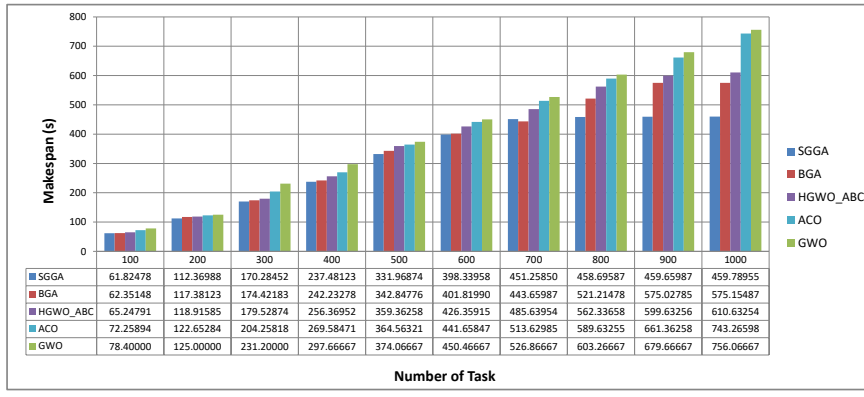
Figure 5. The results of the time to reach the optimal solution

Experiment 2: In this experiment, we will evaluate the makespan of our approach, then the obtained results are compared with those of the other approaches.

In this experimentation, we vary the number of tasks from 100 to 1 000 with a packet of 100, and we set the number of iterations to 1 000. Then we observe the maximum time required for the execution of a batch of tasks.

Figure 6 shows that in the first two packages, all the approaches give almost close values in terms of the makespan. However, from the 7^{th} package the difference between the different approach is clearly visible.

Experiment 3: After evaluating the first metric of the fitness function in the previous experiment, we evaluate its second metric which is the average resource utilization ratio (AVG_{UR}). This metric is between 0 and 1. This metric will be analyzed regarding to the number of tasks, then will compare the obtained results with the other approaches.

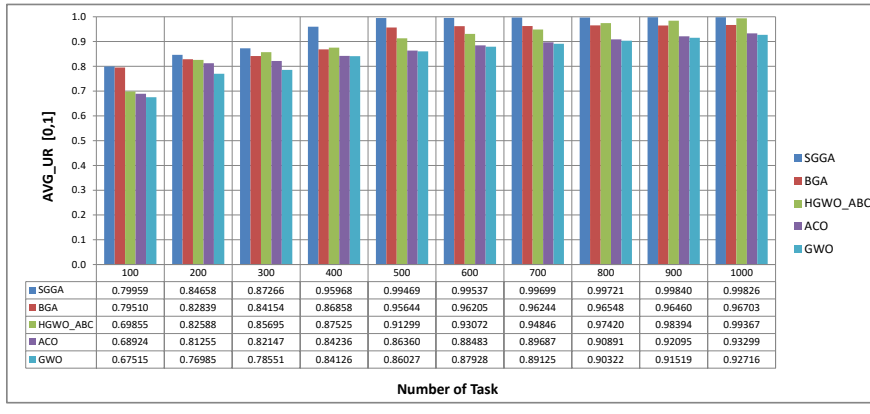


	100	200	300	400	500	600	700	800	900	1000
SGGA	61.82478	112.36988	170.28452	237.48123	331.96874	398.33958	451.25850	458.69587	459.65987	459.78955
BGA	62.35148	117.38123	174.42183	242.23278	342.84776	401.81990	443.65987	521.21478	575.02785	575.15487
HGWO_ABC	65.24791	118.91585	179.52874	256.36952	359.36258	426.35915	485.63954	562.33658	599.63256	610.63254
ACO	72.25894	122.65284	204.25818	269.58471	364.56321	441.65847	513.62985	589.63255	661.36258	743.26598
GWO	78.40000	125.00000	231.20000	297.66667	374.06667	450.46667	526.86667	603.26667	679.66667	756.06667

Figure 6. The results of the makespan compared to the number of tasks

In this experimentation, we vary the number of tasks from 100 to 1 000 with a packet of 100, and we set the number of iterations to 1 000. Then we observe the average rate of resource utilization regarding to the number of tasks. Figure 7 shows that, for the first packet of 100 tasks, our approach and the BGA approach give a higher average resource utilization (AVG_{UR}) than the other approaches. From the 8^{th} packet, the HGOW-ABC approach exceeds that of BGA with a rate of 2.86 %.

Our approach exceeds BGA, HGWO-ABC, ACO and GWO in terms of resource utilization by the order of 3.13 %, 0.46 %, 6.54 % and 7.12 %, respectively. This explains that SGGA gives better performance in terms of average resource utilization compared to other approaches.



	100	200	300	400	500	600	700	800	900	1000
SGGA	0.79959	0.84658	0.87266	0.95968	0.99469	0.99537	0.99699	0.99721	0.99840	0.99826
BGA	0.79510	0.82839	0.84154	0.86858	0.95644	0.96205	0.96244	0.96548	0.96460	0.96703
HGWO_ABC	0.69855	0.82588	0.85695	0.87525	0.91299	0.93072	0.94846	0.97420	0.98394	0.99367
ACO	0.68924	0.81255	0.82147	0.84236	0.86360	0.88483	0.89687	0.90891	0.92095	0.93299
GWO	0.67515	0.76985	0.78551	0.84126	0.86027	0.87928	0.89125	0.90322	0.91519	0.92716

Figure 7. The results of resource utilization rates against the number of tasks

Experiment 4: The fourth metric is evaluated in this experiment. It is the average number of tasks executed in a unit of time (AVG-Throughput). This parameter automatically depends on the makespan. This evaluation is also compared with the same four approaches.

In this experimentation, we set the number of iterations and the number of tasks to 1 000. At the end of the experiment, we calculate the average number of tasks executed per second which is the number of tasks divided by the global execution time. Figure 8 shows that our approach gives a gain of 6.46 %, 16.11 %, 18.81 % and 25.09 % compared to the BGA, HGWO-ABC, ACO and GWO approaches, respectively, which confirms our theoretical assumptions.

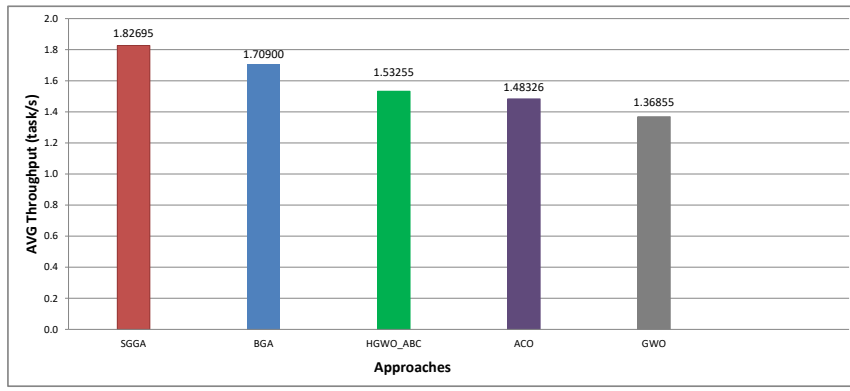


Figure 8. The average throughput comparison

Experiment 5: In this experiment, the SGGA is compared with the other approaches according to the hypervolume indicator, which is the most used metric to compare the performance of scalable multi-objective algorithms [8].

HV is a unary metric that calculates the volume of the area bounded by the set of solutions and a reference point, where a higher value indicates a better result [5].

Figure 9 shows that our approach SGGA and BGA have the best HV indicator with a slight superiority of our approach of around 1.70 %. Compared to other techniques, we find that our approach outperforms other approaches whose gain rate is 13.64 %, 22.87 % and 17.84 % compared to HGWO-ABC, ACO and GWO approaches, respectively.

The HV values confirm that our SGGA approach gives a much more efficient set of solutions than the other approaches.

As a conclusion of the realized experiments, our SGGA approach based on the combination of genetic algorithm and grasshopper optimization algorithm, gives an optimal solution for dynamic load balancing in the CloudIoT.

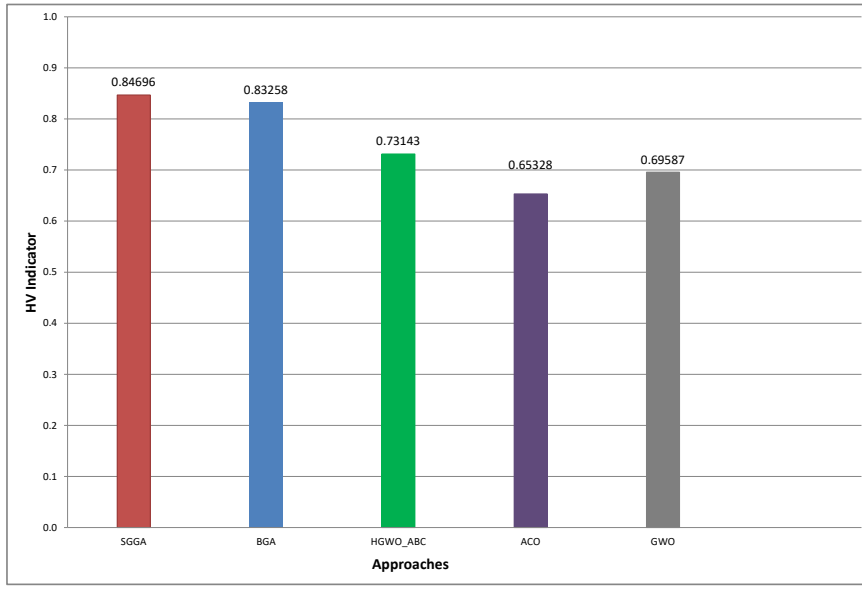


Figure 9. The hypervolume indicator comparison

6 CONCLUSION

The CloudIoT paradigm enables intelligent resource utilization in an equitable manner. In this paradigm, IoTs send their tasks to Cloud Computing for processing or storage, mapping them to the various hardware and software resources represented by virtual machines. When distributing the data to be processed to the virtual machines (VMs), some will be loaded while others will be less loaded or inactive. Load balancing is a mechanism that manages the allocation of VMs to tasks sent by IoTs. It thus allows the optimization of makespan, throughput and the rentable use of resources.

To achieve dynamic load balancing in the CloudIoT, a task scheduler (SGGA) based on the combination between genetic algorithm and grasshopper optimization algorithm has been proposed. The two operators of GA (selection and crossover), are developed to avoid redundancy in the choice of a chromosome several times. Then the mutation operator is replaced by a component based on the grasshopper optimization algorithm. Careful experiments are carried out on CloudSim, to demonstrate that SGGA is more efficient compared to more recent works (BGA, HGOW-ABC, GWO and ACO) in terms of time to reach the optimal solution, makespan, throughput, the average resource utilization ratio and the hypervolume indicator.

In future work, we aim to expand the parameters of load balancing so that SGGA can improve energy consumption and cost, and we implement our approach

on a real system such as a smart city. We aim, also, to evaluate it with the new CEC functions such as Ackley, Rosenbrock, Michalewicz, Dixon and Price function.

REFERENCES

[1] Alguliyev, R. M.—Imamverdiyev, Y. N.—Abdullayeva, F. J.: PSO-Based Load Balancing Method in Cloud Computing. Automatic Control and Computer Sciences, Vol. 53, 2019, No. 1, pp. 45–55, doi: 10.3103/S0146411619010024.

[2] Almezeini, N.—Hafez, A.: Task Scheduling in Cloud Computing Using Lion Optimization Algorithm. International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 8, 2017, No. 11, pp. 77–83, doi: 10.14569/IJACSA.2017.081110.

[3] Arulkumar, V.—Bhalaji, N.: Load Balancing in Cloud Computing Using Water Wave Algorithm. Concurrency and Computation: Practice and Experience, Vol. 34, 2019, No. 8, Art. No. e5492, doi: 10.1002/cpe.5492.

[4] Botta, A.—de Donato, W.—Persico, V.—Pescapé, A.: Integration of Cloud Computing and Internet of Things: A Survey. Future Generation Computer Systems, Vol. 56, 2016, pp. 684–700, doi: 10.1016/j.future.2015.09.021.

[5] Boucetti, R.—Hioual, O.—Hemam, S. M.: An Approach Based on Genetic Algorithms and Neural Networks for QoS-Aware IoT Services Composition. Journal of King Saud University – Computer and Information Sciences, Vol. 34, 2022, No. 8, Part B, pp. 5619–5632, doi: 10.1016/j.jksuci.2022.02.012.

[6] Calheiros, R. N.—Ranjan, R.—Beloglazov, A.—De Rose, C. A. F.—Buyya, R.: CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. Software: Practice and Experience, Vol. 41, 2011, No. 1, pp. 23–50, doi: 10.1002/spe.995.

[7] Ehsanimoghadam, P.—Effatparvar, M.: Load Balancing Based on Bee Colony Algorithm with Partitioning of Public Clouds. International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 9, 2018, No. 4, pp. 450–455, doi: 10.14569/IJACSA.2018.090462.

[8] Guerreiro, A. P.—Fonseca, C. M.—Paquete, L.: The Hypervolume Indicator: Computational Problems and Algorithms. ACM Computing Surveys, Vol. 54, 2021, No. 6, Art. No. 119, doi: 10.1145/3453474.

[9] Gulbaz, R.—Siddiqui, A. B.—Anjum, N.—Alotaibi, A. A.—Althobaiti, T.—Ramzan, N.: Balancer Genetic Algorithm – A Novel Task Scheduling Optimization Approach in Cloud Computing. Applied Sciences, Vol. 11, 2021, No. 14, Art. No. 6244, doi: 10.3390/app11146244.

[10] Li, G.—Wu, Z.: Ant Colony Optimization Task Scheduling Algorithm for SWIM Based on Load Balancing. Future Internet, Vol. 11, 2019, No. 4, Art. No. 90, doi: 10.3390/fi11040090.

[11] Liu, Y.—Xiao, F.: Intelligent Monitoring System of Residential Environment Based on Cloud Computing and Internet of Things. IEEE Access, Vol. 9, 2021, pp. 58378–58389, doi: 10.1109/ACCESS.2021.3070344.

[12] Muthsamy, G.—Chandran, S. R.: Task Scheduling Using Artificial Bee Foraging Optimization for Load Balancing in Cloud Data Centers. Computer Applications in Engineering Education, Vol. 28, 2020, No. 4, pp. 769–778, doi: 10.1002/cae.22236.

[13] Natesan, G.—Chokkalingam, A.: An Improved Grey Wolf Optimization Algorithm Based Task Scheduling in Cloud Computing Environment. The International Arab Journal of Information Technology, Vol. 17, 2020, No. 1, pp. 73–81, doi: 10.34028/iajit/17/1/9.

[14] Ouhame, S.—Hadi, Y.—Arifullah, A.: A Hybrid Grey Wolf Optimizer and Artificial Bee Colony Algorithm Used for Improvement in Resource Allocation System for Cloud Technology. International Journal of Online and Biomedical Engineering (iJOE), Vol. 16, 2020, No. 14, pp. 4–17, doi: 10.3991/ijoe.v16i14.16623.

[15] Ragmani, A.—Elomri, A.—Abghour, N.—Moussaid, K.—Rida, M.: An Improved Hybrid Fuzzy-Ant Colony Algorithm Applied to Load Balancing in Cloud Computing Environment. Procedia Computer Science, Vol. 151, 2019, pp. 519–526, doi: 10.1016/j.procs.2019.04.070.

[16] Saremi, S.—Mirjalili, S.—Lewis, A.: Grasshopper Optimisation Algorithm: Theory and Application. Advances in Engineering Software, Vol. 105, 2017, pp. 30–47, doi: 10.1016/j.advengsoft.2017.01.004.

[17] Shradha, J.—Jayshree, J.—Chandraprbha, K.: Internet of Things Integrates with Cloud Computing. IRJCS: International Research Journal of Computer Science, Vol. 6, 2019, No. 1, pp. 1–3, doi: 10.26562/IRJCS.2019.JACS10082.

[18] Hachimi, H.: Hybridations d'Algorithmes Métaheuristiques en Optimisation Globale et Leurs Applications. Ph.D. Thesis. INSA de Rouen, France, École Mohammadia d'Ingénieurs, Université Mohammed V Agdal, Rabat, Morocco, 2013. `https://tel.archives-ouvertes.fr/tel-00905604` (in French).

[19] Atlam, H. F.—Alenezi, A.—Alharthi, A.—Walters, R. J.—Wills, G. B.: Integration of Cloud Computing with Internet of Things: Challenges and Open Issues. 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2017, pp. 670–675, doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.105.

[20] Benabbes, S.—Hemam, S. M.: An Approach Based on (Tasks-VMs) Classification and MCDA for Dynamic Load Balancing in the CloudIoT. In: Hatti, M. (Ed.): Smart Energy Empowerment in Smart and Resilient Cities (ICAIRES 2019). Springer, Cham, Lecture Notes in Networks and Systems, Vol. 102, 2019, pp. 387–396, doi: 10.1007/978-3-030-37207-1_41.

[21] Gohil, B. N.—Patel, D. R.: An Improved Grey Wolf Optimizer (iGWO) for Load Balancing in Cloud Computing Environment. In: Hu, T., Wang, F., Li, H., Wang, Q. (Eds.): Algorithms and Architectures for Parallel Processing (ICA3PP 2018). Springer, Cham, Lecture Notes in Computer Science, Vol. 11338, 2018, pp. 3–9, doi: 10.1007/978-3-030-05234-8_1.

[22] Kaur, G.—Sachdeva, R.: Virtual Machine Migration Approach in Cloud Computing Using Genetic Algorithm. In: Goar, V., Kuri, M., Kumar, R., Senjyu, T. (Eds.): Advances in Information Communication Technology and Computing (AICTC 2019).

Springer, Singapore, Lecture Notes in Networks and Systems, Vol. 135, 2019, pp. 195–204, doi: 10.1007/978-981-15-5421-6_20.

[23] Patel, D.—Patra, M. K.—Sahoo, B.: GWO Based Task Allocation for Load Balancing in Containerized Cloud. 2020 International Conference on Inventive Computation Technologies (ICICT 2020), 2020, pp. 655–659, doi: 10.1109/ICICT48043.2020.9112525.

[24] Makasarwala, H. A.—Hazari, P.: Using Genetic Algorithm for Load Balancing in Cloud Computing. 2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI 2016), 2016, pp. 1–6, doi: 10.1109/ECAI.2016.7861166.

[25] Shafahi, Z.—Yari, A.: An Efficient Task Scheduling in Cloud Computing Based on ACO Algorithm. 2021 12th International Conference on Information and Knowledge Technology (IKT 2021), 2021, pp. 72–77, doi: 10.1109/IKT54664.2021.9685674.

[26] Shen, L.—Li, J.—Wu, Y.—Tang, Z.—Wang, Y.: Optimization of Artificial Bee Colony Algorithm Based Load Balancing in Smart Grid Cloud. 2019 IEEE Innovative Smart Grid Technologies – Asia (ISGT-Asia 2019), 2019, pp. 1131–1134, doi: 10.1109/ISGT-Asia.2019.8881232.

[27] Agushaka, J. O.—Ezugwu, A. E.—Abualigah, L.: Dwarf Mongoose Optimization Algorithm. Computer Methods in Applied Mechanics and Engineering, Vol. 391, 2022, Art. No. 114570, doi: 10.1016/j.cma.2022.114570.

[28] Talatahari, S.—Azizi, M.—Tolouei, M.—Talatahari, B.—Sareh, P.: Crystal Structure Algorithm (CryStAl): A Metaheuristic Optimization Method. IEEE Access, Vol. 9, 2021, pp. 71244–71261, doi: 10.1109/ACCESS.2021.3079161.

[29] Fahim, Y.—Ben Lahmar, E.—Labriji, E.—Eddaoui, A.: Une Nouvelle conception d'Equilibrage de Charge dans le Cloud Computing. 4eme Journée sur les Technologies d'Information et de Modélisation (TIM'16), 2016, pp. 1–6. `https://www.researchgate.net/publication/318686170` (in French).

[30] Sadeeq, H. T.—Abdulazeez, A. M.: Giant Trevally Optimizer (GTO): A Novel Metaheuristic Algorithm for Global Optimization and Challenging Engineering Problems. IEEE Access, Vol. 10, 2022, pp. 121615–121640, doi: 10.1109/ACCESS.2022.3223388.

[31] Zhang, Y.—Jin, Z: Group Teaching Optimization Algorithm: A Novel Metaheuristic Method for Solving Global Optimization Problems. Expert Systems with Applications, Vol. 148, 2020, Art. No. 113246, doi: 10.1016/j.eswa.2020.113246.

[32] Cheraghalipour, A.—Hajiaghaei-Keshteli, M.—Paydar, M. M.: Tree Growth Algorithm (TGA): A Novel Approach for Solving Optimization Problems. Engineering Applications of Artificial Intelligence, Vol. 72, 2018, pp. 393–414, doi: 10.1016/j.engappai.2018.04.021.

Sofiane Benabbes is a Ph.D. student on computer science, option: security and web technology. His research in the field of dynamic load balancing in the CloudIoT, IoT and cloud computing at the ICOSI Laboratory at the University of Abbes Laghrour Khenchela, Algeria.

Sofiane Mounine Hemam received his B.Sc. in computer science from the Mentouri University of Constantine, Algeria in 1996, and his M.Sc. in computer science from the Larbi Tebessi University of Tebessa, Algeria in 2005. Currently, he is working as Full Professor at the Department of Mathematics and Computer Science at Abbes Laghrour University of Khenchela, Algeria since 2005. He supervised many Ph.D., Master and License students. Between October 2008 and December 2013, he worked on his Ph.D. in computer science. He has published a number of articles in international journals and conferences. His research interests include database, P2P networks, cloud computing and distributed applications, load balancing, fault tolerance, artificial intelligence.