Neural Comput & Applic (2018) 30:1845–1863
https://doi.org/10.1007/s00521-016-2816-4

ORIGINAL ARTICLE

Orthogonal Taguchi-based cat algorithm for solving task scheduling problem in cloud computing

Danlami Gabi[1,3] · **Abdul Samad Ismail**[1] · **Anazida Zainal**[1] · **Zalmiyah Zakaria**[1] · **Ajith Abraham**[2]

Received: 16 September 2016 / Accepted: 19 December 2016 / Published online: 26 December 2016

Abstract In cloud computing datacenter, task execution delay is a common phenomenal cause by task imbalance across virtual machines (VMs). In recent times, a number of artificial intelligence scheduling techniques are applied to reduced task execution delay. These techniques have contributed toward the need for an ideal solution. The objective of this study is to optimize task scheduling based on proposed orthogonal Taguchi-based cat swarm optimization (OTB-CSO) in order to reduce total task execution delay. In our proposed algorithm, Taguchi orthogonal approach was incorporated into tracing mode of CSO to scheduled tasks on VMs with minimum execution time. CloudSim tool was used to implement the proposed algorithm where the impact of the algorithm was checked with 5, 10 and 20 VMs besides input tasks and evaluated based on makespan and degree of imbalance metrics. Experimental results showed that for 20 VMs used, our proposed OTB-CSO was able to minimize makespan of the total tasks scheduled across VMs with 42.86, 34.57 and 2.58% improvement over minimum and maximum job first (Min–Max), particle swarm optimization with linear descending inertia weight (PSO-LDIW) and hybrid PSO with simulated annealing (HPSO-SA) and likewise returned degree of imbalance with 70.03, 62.83 and 35.68% improvement over existing algorithms. Results obtained showed that OTB-CSO is effective to optimize task scheduling and improve overall cloud computing performance through minimizing task execution delay while ensuring better system utilization.

Keywords Cloud computing · Taguchi · Task scheduling · Makespan · Optimization

✉ Danlami Gabi
gabsonley4life@yahoo.ca

Abdul Samad Ismail
abdsamad@utm.my

Anazida Zainal
anazida@gmail.com

Zalmiyah Zakaria
zalmiyah@utm.my

Ajith Abraham
ajith.abraham@ieee.org

1 Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia

2 Machine Intelligence Research Labs, Scientific Network for Innovation and Research Excellence, Auburn, WA 98071, USA

3 Department of Computer Science, Faculty of Science and Education, Kebbi State University of Science and Technology, Aliero, Kebbi State, Nigeria

1 Introduction

Cloud computing is an Internet-based computing, where different services (infrastructure, platform and software as a service) are provided to consumers' using computers and devices through a medium called the Internet. Software services are purchased using pay-per-use service [1–3] and IT infrastructures are rented over a short time period [2, 4, 5]. Systems in cloud computing are dynamically provisioned and presented through means of negotiation between service providers and the cloud consumers' based on unified computing resources [6, 7].

The underlying interest of service provider' is to make sure scheduled tasks meet their expected deadline, maintain better systems utilization and ensure revenue generation [8–12]. Task execution delay which normally affects quality of service (QoS) provisioning is a key issue confronting cloud computing [3, 13–15]. Although the existence of an enterprise prompted a competitive market paradigm on cloud computing environment given priority to revenue generation, dynamic solutions are then needed to schedule tasks on virtual machines (VMs) with shortest expected execution time in order to reduce total task execution delay.

Previous researchers have proposed task scheduling techniques and minimized task execution time, revenue loss and maximized system performance using meta-heuristics such as particle swarm optimization (PSO), genetic algorithm (GA), bee colony optimization (BCO), ant colony optimization (ACO) and global league championship [14, 16–26]. These techniques have contributed to further developments of ideal solutions. With changing cloud computing environment, VMs are limited to handle the volume of the task that often arrives datacenters. Methods applied by existing researchers still need to be addressed with some new methods. An ideal solution that can take cognizance of the dynamic changing nature of tasks for time-effective schedule across VMs becomes paramount.

In contrast to existing AI task scheduling techniques mentioned, cat swarm optimization (CSO) has proven faster than particle swarm optimization in terms of speed and convergence [27, 28]. Its search capability can further be improved to address task scheduling problem in cloud computing although the real version of CSO can increase complexity if directly apply to address task scheduling problem. Its global search (seeking mode) and local search (tracing mode) are carried out independently for each iteration and likewise its velocity and position update. This requires a very high computation time [29, 30]. As a result, a special mechanism is needed to increase its convergence speed and for selection of best virtual machine mapping that will return minimum exaction time.

In this study, we incorporated orthogonal Taguchi-based approach into local search (tracing mode) of CSO to increase its convergence speed and to schedule a task on VMs with minimum execution time in order to overcome task execution delay [2]. As a result, a dynamic task scheduling algorithm called orthogonal Taguchi-based cat swarm optimization (OTB-CSO) that minimized makespan and the degree of imbalance of total task schedule across virtual machines was proposed.

Contribution for the proposed work is summarized as follows:

- Development of *makespan* model for optimum task scheduling as an objective function;
- Hybridization of *Taguchi*-based approach with CSO for optimum task scheduling for cloud environment;
- Development of OTB-CSO algorithm to address the proposed scheduling model;
- Mathematical illustration of OTB-CSO-based approach for optimum task scheduling;
- Implementation of the proposed algorithm on CloudSim tool;
- Performance evaluation of existing works with proposed method based on makespan, the degree of imbalance and percentage improvement (%).

The rest of this article is structured as follows: Works related to task scheduling based on Taguchi approach are explained in Sect. 2. Section 3 discusses Taguchi orthogonal approach. Section 4 discusses problem formulation. Section 5 discusses cat swarm optimization. Section 6 discusses Taguchi optimization and proposed OTB-CSO algorithm. Experimental setup and performance metrics are discussed in Sect. 7. Section 8 discusses the results of the simulation. The conclusion is then presented in Sect. 9.

2 Related Taguchi-based scheduling works

Taguchi used orthogonal array matrix with a number of factors and their levels to form matrix-based experiments [31]. This method is a powerful optimization process. The orthogonal array adopted by Taguchi is useful for reducing run of an experiment and can be applied to solve complex problems [31]. As an effective tool for optimization, it provides an elastic, efficient and systematic way to optimize designs. A signal-to-noise (S/N) ratio is one of the fundamentals of Taguchi-based design [4]. Several researchers have to make used of Taguchi-based design in different fields such as engineering science, social sciences as well as in industries to solve the optimization problem. Few among existing researchers are discussed as follows:

In Abd et al. [32], a dynamic scheduling problem for robotic flexible assembly cells (RFACs) was presented. The researchers applied Taguchi method to minimize total tardiness and number of tardy jobs associated with their scheduling problem. Based on the use of Taguchi method, the researchers were able to reduce a number of simulations for scheduling RFACs at a minimum. Tsai et al. [11] proposed an improved differential evolutionary algorithm (IDEA) using Taguchi-based approach. The researchers focused on task scheduling and resource allocation for their optimization process. They discovered the application of Taguchi method enabled generation of potential offspring on macro-space, where the effectiveness of their proposed algorithm was compared with that of the differential evolutionary algorithm (DEA) and non-static genetic algorithm II (NSGA-II) and results outperformed better than aforementioned algorithms.

In Cavory et al. [33], machines associated with manufacturing production line are considered for optimum schedule. The researchers presented a genetic algorithm inspired by Taguchi optimization approach to increase system throughput. The result obtained was validated against real genetic algorithm which shows outstanding performance. In the part of Asefi et al. [34], a hybrid no-wait flexible flowshop scheduling algorithm that combined non-static genetic algorithm (NSGA-II) with variable neighborhood search (VNS) was presented. The researchers improved the convergence speed of genetic algorithm as they incorporated Taguchi method to minimize the makespan and mean tardiness of jobs. The result showed remarkable improvement as compared to existing algorithm.

Chang et al. [35] also improved the effectiveness of a genetic algorithm using Taguchi-based approach. The researchers used this method to minimize completion time of task on virtual machines through encoding of feasible solutions in a genetic algorithm. Results showed their proposed algorithm outperformed that of the real genetic algorithm (GA) and that of existing solutions. Caprihan et al. [36] studied the effect associated with scheduling rules based on the performance of a dynamic scheduling on flexible manufacturing systems. They applied Taguchi rule to minimize number of experiments. Likewise, in Tsai et al. [5], the Taguchi-based genetic algorithm (TBGA) that solved job shop scheduling problem was presented. They discovered Taguchi method to generate optimal offspring in a genetic algorithm which later produced better results.

Researchers such as Tsai et al. [31] designed an enhanced parallel CSO (EPCSO) to achieve better accuracy with less computational time. Based on existing works, this research makes use of the idea proposed in [5] for design of our proposed orthogonal Taguchi-based cat swarm optimization (OTB-CSO) algorithm in order to minimize makespan of virtual machines with the aim of reducing task execution delay in a dynamic cloud environment. Section 3 explains Taguchi orthogonal approach in detail.

3 Taguchi orthogonal approach

Researchers in the field of science, engineering and social sciences proposed methods that addressed given problem in an efficient manner. Most of these methods (game theory, artificial intelligence, Taguchi, etc.) are used to address complex problems. For instance, Taguchi method proposed by Dr. Genichi Taguchi [37] is, however, applicable over a wide range of areas, comprising of processes in raw materials manufacturing, subsystems and consumer markets. This method used matrix form, based on orthogonal array representation. As earlier highlighted, the reasoning ability of Taguchi optimization method is to study a large number of design variables with few experiments [31, 35]. Taguchi combined design variables and represents an orthogonal array in a matrix form. The concept adopted by Dr. Genichi Taguchi is used for addressing both single- and multi-objective optimization problems [11]. The proposed orthogonal array matrix experiment by Taguchi showed two-level orthogonal array (2OA) with "Z" factors, where "Z" is considered as design factors (variables), and each factor is said to be based on two levels. Taguchi formulated a general symbol for establishing an OA with two levels of Z factors using Eq. 1 [11, 35].

$$L_n\left(2^{n-1}\right), \tag{1}$$

where $n - 1 =$ symbolizing the columns numbers in two-level orthogonal array; $n = 2\ k$ number of experiments corresponding to the n rows, and columns, $2 =$ number of required level for each factor Z; and $k =$ a positive integer ($k > 1$).

The design matrix table of Taguchi (e.g., Table 1) showed columns array values mutually orthogonal. According to Taguchi, for any column pairs, combinations of all factors at each level occur at an equal number of times. In Table 1, there are seven parameters A, B, C, D, E, F and G, consider at two levels, called an "L_8" design, such that the number "8" indicates eight rows expected to be tested. According to Taguchi, when six factors are to be allocated, and either of which possessed two orthogonal levels "$L_8(2^6)$," the only needed columns for run of the experiments are six; hence, $L_8(2^7)$ orthogonal is then consider sufficient since it contains seven columns. "L_8" is an indication that eight experiments are to be carried out by studying six variables at 2 levels where the value "6" represents the dimension. Excluding the application of Taguchi method to run matrix experiment with $L_8(2^7)$, there are possible 128 experiments evaluations, but with Taguchi, only "8" experiment is expected to achieve optimal results.

Table 1 $L_8(2^7)$ orthogonal array [49]

Experiment number	Factors						
	A	B	C	D	E	F	G
Column numbers							
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

The need to determine which level is optimal for each of the factors in the orthogonal table [5, 37] is one of the goals in carrying out a matrix experiments. The signal-to-noise ratio (S/N) proposed by Taguchi is mean square deviation that represents its objective function. Taguchi definition based on signal-to-noise ratio is $\mathrm{S/N} = -10\log\frac{1}{n}\left(\sum y^2\right)$ as smaller-the-better characteristics, and $\mathrm{S/N} = -10\log\frac{1}{n}\left(\sum\frac{1}{y^2}\right)$ as larger the better, where the integer "n" is the number associated with the experiments on OA $L_n(2^{n-1})$ [11]. Assuming two sets of solutions with seven parameters exist, where optimal combination of their values is expected according to Table 1, the orthogonal array elements normally indicate which parameter value is needed for next experiments based on specified conditions [31]. For instance, factors in matrix of Table 1 indicate that the factor "1" should be considered for first set of experiments while "2" for second experiments.

Assuming Table 1 matrix is to experiment for task scheduling, horizontal numbers in the orthogonal array (1,2,3,4,5,6,7) represent tasks instances, while factors "1" and "2" at the orthogonal array table represent the position of VMs, each with expected time to compute for each available task instances. Table 2 shows an indication of $n \times m$ matrix for expected time to compute (ETC) for tasks $T = \{T_0, T_1, T_2, T_3, T_4, T_5, T_6\}$ with eight virtual machines $V = \{V_0, V_2, V_3, V_4, V_5, V_6, V_7\}$. The aim of this study is to incorporate Taguchi approach into tracing mode (local search) of CSO for task mapping on VMs with minimum ETC in order to minimize makespan of total tasks scheduled. The problem formulation associated with the task scheduling optimization adopted in this study is provided in Sect. 4.

4 Problem formulation

Cloud computing is comprised of datacenters, and each datacenter is associated with virtual machines (VMs) arranged in parallel order. Suppose there exist a set of cloudlets (tasks) that emanated from cloud users' and are channeled to cloud datacenter, awaiting to be scheduled by cloud broker (CB) to respective VMs with minimum execution time. Assume all VMs as heterogeneous (having different processing speeds in million instructions per second) and cloudlet assignment strategy is based on first-come first-serve (FCFS), where pre-emption is not allowed. Our goal is to dynamically assign these cloudlets to VMs by applying best cloudlet assignment strategy using proposed OTB-CSO method in order to minimize the maximum completion time (makespan). However, expected time to compute (ETC) matrix experiment [13] as illustrated in Table 2 will be used to make schedule decision, where each cloudlet is will be associated with required VM and each VM will contain expected a time of computation (ETC) of each cloudlet. To ensure schedule decision, the value of ETC matrix for all virtual machines is generated using uniform distribution in ratio of million instructions per second (MIPS) to the length of cloudlet in million instruction (MI) [13]. The model formulation to this problem is, however, presented in Sect. 4.1.

Table 2 Example of ETC matrix

	T_0	T_1	T_2	T_3	T_4	T_5	T_6
V_0	T_0/V_0	T_1/V_0	T_2/V_0	T_3/V_0	T_4/V_0	T_5/V_0	T_6/V_0
V_1	T_0/V_1	T_1/V_1	T_2/V_1	T_3/V_1	T_4/V_1	T_5/V_1	T_6/V_1
V_2	T_0/V_2	T_1/V_2	T_2/V_2	T_3/V_2	T_4/V_2	T_5/V_2	T_6/V_2
V_3	T_0/V_3	T_1/V_3	T_2/V_3	T_3/V_3	T_4/V_3	T_5/V_3	T_6/V_3
V_4	T_0/V_4	T_1/V_4	T_2/V_4	T_3/V_4	T_4/V_4	T_5/V_4	T_6/V_4
V_5	T_0/V_5	T_1/V_5	T_2/V_5	T_3/V_5	T_4/V_5	T_5/V_5	T_6/V_5
V_6	T_0/V_6	T_1/V_6	T_2/V_6	T_3/V_6	T_4/V_6	T_5/V_6	T_6/V_6
V_7	T_0/V_7	T_1/V_7	T_2/V_7	T_3/V_7	T_4/V_7	T_5/V_7	T_6/V_7

4.1 Mathematical model of the scheduling goal

The objective function for our proposed scheduling problem was formulated based on [21] and [38] model as follows: Let $\mathrm{TL}(i) = \{T_1, T_2,\ldots, T_n\}$ denote set of the cloudlets that are independent on each and $V(j) = \{V_1, V_2, \ldots, V_m\}$ symbolize the position of virtual machines (VMs). Suppose a cloudlet TL(i) is scheduled on a VM $V(j)$, execution time exec(i, j) of a cloudlet executed on one VM $V(j)$ is calculated using Eq. 2.

$$\mathrm{exec}(i,j) = \frac{\mathrm{TL}(i)}{\mathrm{npe}\ (\mathrm{j}) \times V_{\mathrm{mips}}(j)}, \quad \forall i \in \mathrm{Task}, i = \{1\}, j \in \mathrm{Vm}, j = \{1\} \tag{2}$$

where exec(i, j) is the execution time of running a single cloudlet on one VM; $V_{\mathrm{mips}}(j)$ is the length of a cloudlet in million instruction (MI); $V_{\mathrm{mips}}(j)$ is the VM speeds in million instructions per second (MIPS); and npe(j) is the number of processing elements. When more than one VMs are involved to rum set of cloudlets, the total execution time Texec(i, j) of all cloudlets executed on all VMs is calculated using Eq. 3.

$$\mathrm{Texec}(i,j) = \sum\left(\frac{\mathrm{TL}(i)}{\mathrm{npe}(j) \times V_{\mathrm{mips}}(j)}\right), \quad \forall i = \{1,2,\ldots,n\}, j = \{1,2,\ldots,m\} \tag{3}$$

Our aim is to minimize total cloudlets execution time on all VMs; therefore, the makespan model expected to be minimized is shown in Eq. 4.

$$\text{makespan} = \min\left\{\max \sum\left(\frac{TL(i)}{\text{npe}(j) \times V_{\text{mips}}(j)}\right)\right\}, \quad \forall i = \{1, 2, \ldots, n\}, j = \{1, 2, \ldots, m\} \tag{4}$$

Equation 4 above is the fitness function of our current scheduling problem that achieved an optimal solution which minimizes makespan of total cloudlet scheduled on all VMs.

5 Cat swarm optimization

In solving complex computation problems, researchers established the collective behavior of animals for the purpose of attaining a common goal have a significant resemblance in terms of finding an optimal solution. With interesting observation on these animals, different algorithms were proposed, most especially those belonging to the family of algorithms that commonly refer to as swarm intelligence [39, 40]. These algorithms have gained popularity among many researchers in addressing complex problems. Some of these algorithms include artificial bee colony (ABC), which is realized by observing the behavior of bees when foraging for food [41], ant colony optimization (ACO) and realized based on ant behavior as they leave pheromone trails while working in search of food [42]. Cat swarm optimization is one among swarm optimization technique added to the family of swarm intelligence [28]. The interesting behavior of cat enabled them to observe that the cat has resting behavior and also chasing behavior. These behaviors have been modeled into seeking mode (correspond to resting mode) and the tracing mode (which is the seeking stage). One of the attributes associated with the modeling is the mixed ratio (MR), a controllable factor that distinguished cat in either tracing or seeking mode. As cat progresses closer to solutions (prey), best results are updated at the memory and the process continues till all cats achieve best solution (fitness) [28].

5.1 Seeking and tracing mode of cat

5.1.1 Seeking mode

The seeking mode corresponds to global search process of CSO. It modeled cat behavior as per resting, looking around, at the same time deciding next position to move to [42]. Four attributes are associated with this mode. The seeking memory pool (SMP) indicates memory size sort by the cat; seeking range of selected dimension (SRD) is used for setting the mutation ratio for selecting cat dimensions; counts of dimension to change (CDC) is used to disclose how many dimensions according to cat number varied; and finally, the self-position considering (SPC) represents a Boolean variable that unveils whether position the cat is presently standing can be chosen as the candidates' position to move into [28]. Algorithm 1 shows the procedure for the seeking mode [42].

Algorithm 1: CSO seeking mode

```
1. Generate Y (where Y = SMP) copies of k-th cat, i.e. $Z_{qd}$ where $(1 \le q \le Y)$ and $(1 \le d \le D)$ where D is the overall dimension.
2. Change at random the dimension of cats as per CDC by applying mutation operator to $Z_{qd}$.
3. Determine all changed cats' fitness values.
4. Discover most suitable cats (non-dominant) based on their fitness values.
5. Replace the position of the k-th cat after picking a candidate at random from Y.
```

5.1.2 Tracing mode

The tracing mode is the local search of CSO [42]. It is, however, represented as follows:

1. Compute and update cat kth velocity using new velocity in Eq. 5:

$$V_{K,d} = V_{K,d} + \left(c_1 \times r_1 \times \left(\text{Xbest}_d - X_{K,d}\right)\right) \quad d = 1, 2, \ldots, M \tag{5}$$

   where c is the constant value of acceleration and r is the uniformly distribution random number in the range of [0,1]. For each iteration, Eq. 5 is used to update the velocity.

2. Add new velocity by computing the current (new) position of the kth cat using Eq. 6:

$$X_{k,d} = X_{k,d} + V_{k,d} \tag{6}$$

3. Calculate the fitness values of all cats.
4. Update and return best cats with the best fitness.

5.2 Proposed CSO tracing mode (local search)

The Taguchi orthogonal array is great potential for guiding metaheuristic algorithms to obtain good solution when apply to solve NP-hard problem. Our objective is to minimize makespan of total tasks scheduled across virtual machines (VMs) and to reduce the degree of imbalance of all VMs in a dynamic cloud computing environment. As a result, an optimum task scheduling algorithm based on Taguchi is proposed [14, 25]. In real CSO, its most control variable is the mixed ratio (MR) and defines in the range of [0,1]. This value is use to show a number of a cat in either seeking or tracing mode. For instance, if MR is set to 1%, amounts of cats that allow into tracing mode will be 10 and 90% of cats that are to move into seeking mode. This showed a number of cats that moved into seeking mode (global search mode) will always exceed that of the tracing mode (local search mode). In this process, mutation operation of CSO at local search is bound to affect performance which may end up not achieving near-optimal solution [11]. On the other hand, the global search (seeking mode) and local search (tracing mode) of CSO are carried out independently for each iteration. This causes its velocity and position update to be likewise done independently with a very high computation time [30]. These served as a drawback to this algorithm. In order to overcome this drawback, local search of CSO is incorporated with orthogonal array (OA) of Taguchi method to improve computation time by reducing time consumption of the algorithm to run a large number of tasks with few experiments [35].

The tracing mode operation of CSO is restructured as follows:

1. Generate two velocity sets:

$$\mathrm{Vo}_{k,d}(t) = \begin{cases} \mathrm{Vset}_{1K,d}(t) = V_{K,d}(t-1) + (c_1 \times r_1 \times (\mathrm{Xgbest}_d(t-1) - X_{K,d}(t-1))) \\ \mathrm{Vset}_{2K,d}(t) = V_{K,d}(t-1) + (c_1 \times r_1 \times (\mathrm{Xlbest}_d(t-1) - X_{K,d}(t-1))) \end{cases} \tag{7}$$

Such that:

$$\mathrm{Vo}_{k,d}(t) = \begin{cases} \mathrm{Vset}_{1k,d}(t), & \text{if orthogonal array element is ``1''} \\ \mathrm{Vset}_{2k,d}(t), & \text{otherwise} \end{cases} \tag{8}$$

where $\mathrm{Vo}_{k,d}$ represents two candidates' velocity sets; d is dimension of the solution space; Xgbest_d represents the global best position of the cat; Xlbest_d is the local best position of the cat; r_1 represents uniform random number in the range of [0,1]; c_1 is a constant value of acceleration; $X_{K,d}$ represents position of the cat; and t is the number of iteration.

We note that, based on the representation of Taguchi orthogonal matrix in Table 1, elements "1" and "2" hold expected time to compute (ETC) of each virtual machine on task instances to execute at each level of the experiment. However, for the task size, the orthogonal array size is often determined based on $L_n(2^{n-1})$ matrix. From the generated two velocities, one is chosen to update the original velocity $V_{k,d}$, and each time there is a run of the experiment according to Eq. 9:

$$V_{k,d} \begin{cases} \max v, & \text{if} \left[V_{k,d} + Vo_{k,d}(t) \right] \quad > \text{maximum velocity}, \\ & V_{k,d} + Vo_{k,d}, (t) \quad \text{otherwise} \end{cases} \tag{9}$$

2. Add new velocity by computing current (new) position of kth cat using Eq. 10:

$$X_{k,d} = X_{k,d} + V_{k,d} \tag{10}$$

3. Calculate each cat fitness value.
4. Sum the fitness values according to generated velocities, compare and select the final velocity to formulate the latest velocity.

6 Taguchi optimization algorithm

The orthogonal array of Taguchi approach is a good optimization method. In running a matrix experiment for task scheduling based on ETC matrix, Taguchi method can help minimize execution time of a task. According to Dr. Genichi Taguchi [37], instead of running a large number of an experiment that required a very large computation time, it divides the experiment into two or more orthogonals, depending on the size of the tasks and run according to Algorithm 2. This method was used in our proposed algorithm to increase the speed of convergence of CSO to run several heterogeneous task instances on VMs. The detail pseudocode of Taguchi method for our matrix experiment is shown in Algorithm 2 [5]:

Algorithm 2: Taguchi Optimization Algorithm

```
1. Select two-level orthogonal Taguchi array matrix such that, $L_n(2^{n-1})$ $\forall n \geq N+1$ and $N$ represent task
2. numbers
3. Generate two sets of velocities $Vset_{1_{K,d}}(t)$ and $Vset_{2_{K,d}}$ (t) according to **Equation (7)**
4. Determine dimension of scheduling problem which corresponds to task number N.
5. Calculate the fitness values of the, $n$ experiments in accordance with the orthogonal array $L_n(2^{n-1})$.
6. The above algorithm is applied at tracing mode of cat swarm optimization (CSO) for minimization of
7. makespan.
```

6.1 OTB-CSO based algorithm

In this subsection, we outlined our proposed OTB-CSO method that solved the task scheduling model presented in Sect. 4.1. The essence is to find an optimal scheduling pattern that will guarantee minimum makespan of all tasks scheduled on VMs. Algorithm 3 shows our proposed OTB-CSO, and Fig. 1 shows the flowchart of the algorithm.

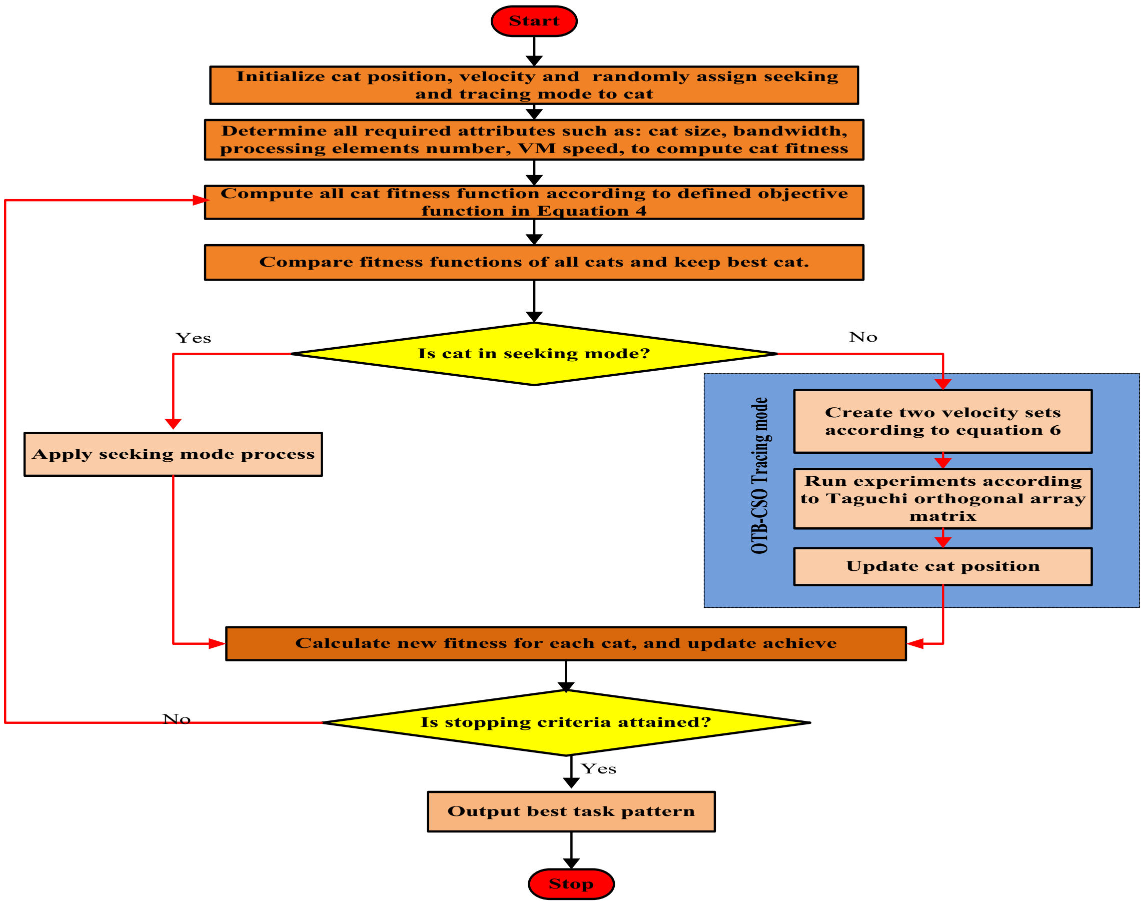
Start
Initialize cat position, velocity and randomly assign seeking and tracing mode to cat
Determine all required attributes such as: cat size, bandwidth, processing elements number, VM speed, to compute cat fitness
Compute all cat fitness function according to defined objective function in Equation 4
Compare fitness functions of all cats and keep best cat.
Yes
Is cat in seeking mode?
No
Apply seeking mode process
OTB-CSO Tracing mode
Create two velocity sets according to equation 6
Run experiments according to Taguchi orthogonal array matrix
Update cat position
Calculate new fitness for each cat, and update achieve
No
Is stopping criteria attained?
Yes
Output best task pattern
Stop

Fig. 1 Flowchart of the proposed OTB-CSO method

Algorithm 3: OTB-CSO Algorithm

```
1.  Start
2.  Initialize associated cats' parameters; MR, mixed ratio; SMP; SRD; CDC; SPC,
3.  Generate an empty non-dominant archive of (n × m) size of uniform random number [0,1].
4.  Initialize position of cats, the velocity of cats and cat flag.
5.  Determine all require attributes such as virtual machine number, number of processing elements,
6.  processing power to calculate cats' fitness functions.
7.  Compute all cats according to defined objective(Fitness) functions in Equation (4)
8.  Compare fitness functions of all cats, keep position with best fitness value.
9.  Do
10. increment_iteration_number ← $t + 1$
11.  If (flag← 1)
12. Apply seeking mode process as follows:
13. Generate y (where y=SMP) copies of k-th cat i.e. $Z_{qd}$ where (1≤q≤Y) and (1≤d≤D) where D is the
14. overall dimension.
15. Change at random dimension of cats as per CDC using + or – mutation operation to $Z_{qd}$.
16. Determine the fitness of changed cats.
17. Discover and replace cat best position after picking a candidate at random from Y.
18. Else
19. Apply Taguchi optimization as follows:
20. Select two-level orthogonal Taguchi array matrix such that, $L_n(2^{n-1})$ $\forall n \geq N + 1$, where N
21. represent task number.
22. Generate two sets of velocities according to Equation (7) and determine the dimension of the
23. scheduling problem that corresponds to task number N.
24. Calculate the fitness values of the n experiments according to orthogonal array $L_n(2^{n-1})$.
25. Endif
26. Choose current best member as $Xl_{best}$ and corresponding best position as $X_{pbest}$
27. If ($Xl_{best} > X_{gbest}$)
28. $Xl_{best} = X_{gbest}$
29. $X_{pbest} = G_{best-id}$ // current best position becomes the global best position
30. Compute and update the new velocity and current position according to (Equation (9) and (10))
31. If (termination condition reached)
32. Output task sequence with the best task scheduling pattern.
33. Else
34. Go to step 6.
35. Endif
36. Endif
37. End.
```

6.2 Taguchi experimental illustration for task scheduling

The two-level orthogonal array in Table 1 with column numbers (1,2,3,4,5,6,7) are assumed for task instances. The element at each level of the orthogonal array represents index value that holds expected execution time of each task. Assuming expected time to compute (ETC) matrix representation of L_8 array in Table 2 is considered for task schedule with seven tasks to experiment at eight levels, our objective is to find the best sequence of task instances that can reduce makespan and degree of imbalance of total tasks scheduled on VMs. An ETC matrix shown in Table 2 is similar to orthogonal Taguchi array in Table 1, where values "1" and "2" indicate element for the first run and second run of the experiment. By applying Taguchi method at tracing mode of CSO, task instances are mapped on VMs based on the orthogonal method. Assuming seven (7) tasks (T_0, T_1, T_2, T_3, T_4, T_5, T_6) are scheduled on VMs, an $L_8(2^7)$ orthogonal is better to consider for the experiment since its dimension is 7 and can run at level 8. In Table 1, to run the third row, corresponding elements of the orthogonal are: 1,2,2,1,1,2,2. The elements of the third run according to Taguchi are used to generate two set of tasks such that, T_0, T_3, T_4 are considered for the first set of the run and T_1, T_2, T_5, T_6 to for second set of tasks consider for the set of the second run.

To demonstrate the above scenario on our proposed algorithm, assume sequence numbers "1" and "2" in Table 1 are expected execution time of each task, and our goal is to make sure the tracing mode of CSO utilized Taguchi-based approach to minimize the fitness function indicated in Eq. 11 [31].

$$f(x) = \sum_{d=1}^{m} \frac{1}{x_d} \tag{11}$$

$\forall d \in M,\ d = \{1, 2,\ldots, M\}$, where M is the dimension which contained number of tasks to be schedule. We

Table 3 Position and original velocity

Current cat position	Considered factor						
	A	B	C	D	E	F	G
X	1	1	1	1	1	1	1
$V_{k,d}\ (t)$	2	2	2	2	2	2	2

Table 4 Generated velocity sets for the experiment

Cat velocities	Considered factor						
	A	B	C	D	E	F	G
$\text{Vset}_{1k,d}\ (t)$	0	1	3	2	4	2	3
$\text{Vset}_{2k,d}\ (t)$	3	2	4	3	0	1	2

assume previous run of the experiment with current position and velocity of the cat as indicated in Tables 3 and 4. Based on orthogonal presentation in Table 1, when the algorithm is run, current position and velocity of cat are updated by the best velocity among the two velocities generated by Taguchi method, and the values obtained are indicated in Table 5. At each level of the experiment, the fitness value obtained in Table 5 is based on Eq. 11. Whenever there is a run of the algorithm, the OTB-CSO updates it positions and a new fitness is generated. Hence, obtained fitness values from the updated position of the cat are 1.46, 1.70, 1.47, 1.54, 1.49, 1.29, 1.48 and 1.24 as shown in Table 5. To find the total fitness for each generated velocity sets, the algorithm scans through the first column "A" in matrix Table 1, which has the value 1,1,1,1,2,2,2,2 (as shown in Fig. 2). As earlier stated, for the first run of the experiment, it takes an index value of "1" which corresponds to the first velocity set $\text{Vset}_{1K,d}$ of the orthogonal and the index value "2" for second run of the experiment which represents that of the second velocity set $\text{Vset}_{2K,d}$. Therefore, OTB-CSO finds total fitness value for the first "1" set of experiment which corresponds to the first velocity set in column "A," by summing the obtained values 1.46, 1.70, 1.47, 1.54 while that of second velocity set using the obtained values 1.49, 1.29, 1.48, 1.24 as indicated in Table 6. The process continues for the remaining "B" to "G."

Each column (B, C, D, E, F) based on their index values is mapped according to Fig. 2, and their total fitness is obtained as shown in Table 6 since our objective is to find a sequence of the tasks that will minimize the total makespan. The two accumulated fitness obtained by the two velocity sets on each column of the orthogonal, each fitness on each column are compared and the velocity that generates fitness with a minimum amount of time is selected for the next run of the experiment. The comparison is taken and selected candidate velocity which is now

Table 5 Fitness values for $L_8(2^7)$

Experiment number	Updated position of cat							Fitness
	A	B	C	D	E	F	G	
1	$1+2+0=3$	$1+2+1=4$	$1+2+3=6$	$1+2+2=5$	$1+2+4=7$	$1+2+2=5$	$1+2+3=6$	1.46
2	$1+2+0=3$	$1+2+1=4$	$1+2+3=6$	$1+2+3=6$	$1+2+0=3$	$1+2+1=4$	$1+2+2=5$	1.70
3	3	5	7	5	7	4	5	1.47
4	3	5	7	6	3	5	6	1.54
5	6	4	7	5	3	5	5	1.49
6	6	4	7	6	7	4	6	1.29
7	6	5	6	5	3	4	6	1.48
8	6	5	6	6	7	5	5	1.24

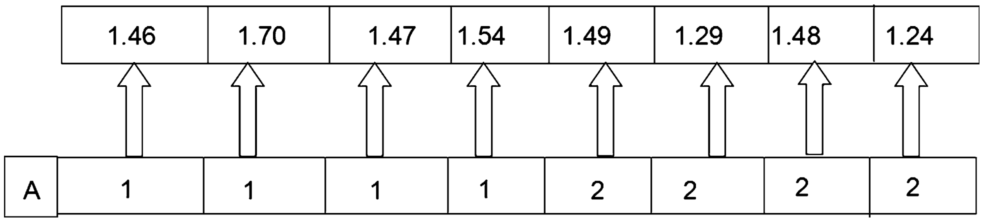

Fig. 2 First column mapping for accumulated fitness of two velocity sets

Table 6 Accumulated fitness values for the two velocities based on $L_8(2^7)$ orthogonal

Velocities	Considered factors						
	A	B	C	D	E	F	G
Total fitness $\text{Vset}_{1k,d}$ (t)	6.17	5.94	5.88	5.90	5.46	5.73	5.77
Total fitness $\text{Vset}_{2k,d}$ (t)	5.50	5.53	5.79	5.77	6.21	5.94	5.90
Selected candidates	Vset_2	Vset_2	Vset_2	Vset_2	Vset_1	Vset_1	Vset_1

Table 7 Final velocity for the next experiment

Final velocities	Considered factors						
	A	B	C	D	E	F	G
$V_{k,d}$	3	2	4	3	4	2	3

Table 8 Computer and utility software specification

OS	Windows specifications
Processor	Intel® Core™ i5-5200U CPU @ 3.60 GHz 3.60 GHz
System type	Window 10 (64-bit), ×64-based processor
Memory	4 GB DDR3L RAM
Hard disk	1000 GB (1 TB) SATA-3G HDD
Software	Eclipse-java-luna-SR2-win32-x86-64
Simulation tool	CloudSim 3.0.3 [43]

the new velocity is shown in Table 6. The values 3,2,4,3,4,2,3 in Table 7 based on the selected velocity sets are obtained using Table 4.

7 Experimental setup

A variety of environments have been used to evaluate recent task scheduling algorithms. This environment differs from one to another in terms of hardware specifications and software utilities; hence, result obtained when running an algorithm on two different environments may vary. For the sake of this article, we experiment with the following computer specification and utility software shown in Table 8.

The CloudSim is regarded best simulation tool for experimenting cloud computing scenario, modeling of real cloud computing environment and evaluating resource provisioning algorithm [44]. However, CloudSim package was extended to implement our proposed algorithm as used by several other researchers like in [14, 18 and 45]. We ensured VMs- and tasks (cloudlets)-associated properties are heterogeneous and are uniformly generated with varying speed in million instructions per seconds (MIPS) and length in million instructions (MI). The choice of properties for both virtual machines, host and tasks used for the experiment is based on [13, 46] as shown in Table 9. The selection of value for inertia weight and that of construction factors ($c1$, $c2$) is based on [47] as shown in Table 10.

7.1 Performance metric

Cloud computing task scheduling metrics are used for evaluating the performance of task scheduling algorithms. These metrics are categorized based on either computational based (e.g., execution time, throughput, makespan) or network based (e.g., computational cost, roundtrip cost) [13, 16, 48]. This study considered makespan, degree of imbalance and performance improvement rate (PIR %) to evaluate the performance of the proposed OTB-CSO.

7.1.1 Makespan time

Makespan is the maximum total execution time of all tasks executed on all VMs [8, 13]. This article minimized makespan of total task schedule on VMs as defined in Eq. 4.

7.1.2 Degree of imbalance

The degree of imbalance (DI) is used to define extent at which task is distributed among VMs. It is also used to investigate unbalanced load across VMs. A small value for the degree of imbalance (DI) unveils how tasks on a system are balanced, either the better the performance. Hence, our

Table 9 Experimental setting

No. of runs of the simulation	10
Datacenter	
No. of datacenter	2
No. of host in a datacenter	1
Host RAM	2 GB
Storage	ITB
Bandwidths	10 GB/s
Accumulated host processing power	1,000,000 MIPS
Cloudlets	
Lengths	100–1000 MIs
No. of cloudlets	10–100
VMs	
VMs used	20,10,5
VMs monitor	Xen
Ram	0.5 GB
Storage	10 GB
Bandwidth	1 GB/s
VMs processing power	1000–10,000 MIPS
Processing element	1–2
VM policy	Time-shared

Table 10 Parameter setting for PSO and CSO

Algorithm	Parameter	Value
PSO	Particle size	100
	Self-recognition coefficients ($c1$, $c2$)	2
	Uniform random number ($R1$)	[0,1]
	Maximum iteration	1000
	Inertia weight (W)	0.9–0.4
	Mixed ration	2%
CSO	Count dimension to change	5%

aim is to ensure load balancing across virtual machine through a minimizing degree of imbalance is achieved. It is denoted using Eq. 12 [8, 13, 49].

$$\mathrm{DI} = \frac{T_{\max} - T_{\min}}{T_{\mathrm{avg}}} \tag{12}$$

where $T_{\max}$, $T_{\min}$ and T_{avg} are the maximum, minimum and the average total execution times among all VMs, respectively. This values are obtained when Eq. 2 is implemented for $\forall i \in \mathrm{Task}, i = \{1, 2, \ldots, n\}$ and $j \in \ \mathrm{VM}, j = \{1, 2, \ldots, m\}$.

7.1.3 Performance improvement rate (PIR%)

The reduction in makespan for the proposed OTB-CSO algorithm over existing schemes is determined using Eq. 13. The improvement rate will help discover efficiency of the algorithms in reducing makespan and degree of imbalance when compare to existing algorithms [13, 48].

$$\mathrm{PIR}(\%) = \left(\frac{(\mathrm{makespan}(\mathrm{other\,scheme}) - \mathrm{makespan}(\mathrm{OTB-CSO}))}{\mathrm{makespan}(\mathrm{other\,scheme})}\right) \times 100 \tag{13}$$

8 Results and discussion

8.1 Results

Ten (10) independent runs of the simulation are carried out on Min–Max [50], PSO-LDIW [14], HPSO-SA [45] and OTB-CSO algorithm for the same size of input cloudlets (tasks). In order to determine the impact of VMs on proposed OTB-CSO algorithm besides input tasks, the proposed method was evaluated based on 20, 10 and 5 VM instances, where best, worst and average makespan are obtained and tabulated in Tables 11, 12 and 13. Tables 14, 15 and 16 show the value of the degree of imbalance obtained by the four algorithms using 20, 10 and 5 VM instances. Likewise, performance improvement rate [PIR (%)] on makespan and degree of imbalance is presented in Tables 17 and 18.

The results of the simulation are also compared with that of real CSO and enhanced parallel cat swarm optimization (EPCSO) from the literature as shown in Table 19.

8.2 Discussion

For each problem size, ten independent simulation runs are executed based on input tasks 10–100 on Min–Max, PSO-LDIW, HPSO-SA and OTB-CSO algorithms. The makespan, degree of imbalance and percentage improvement (%) achieved by the four scheduling algorithms are tabulated in Tables 11, 12, 13, 14, 15, 16, 17 and 18. Table 19 also shows results of comparison with real CSO and EPCSO in terms of execution time. Figures 3, 4 and 5 show a graph of the average makespan achieved by the four scheduling algorithms, while Figs. 6, 7 and 8 show a graph of the average degree of imbalance. Table 17 unveils the percentage improvement achieved by OTB-CSO algorithm over three other algorithms. The result obtained showed that for 20 VMs used, OTB-CSO was able to minimize makespan by achieving 42.86, 34.52 and 2.45% reduction over Min–Max, PSO-LDIW and HPSO-SA algorithms. However, for 10 VMs used, it minimized the makespan by achieving 39.87, 27.23 and 15.03% reduction over Min–Max, PSO-LDIW and HPSO-SA algorithms. Likewise, for 5 VMs used so far, it was able to minimize the makespan of

Table 11 Comparison of makespan obtained with 20 VMs

Task	Min–Max			PSO-LDIW			HPSO-SA			OTB-CSO		
	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
10	14.77	47.07	30.66	14.23	32.90	20.80	6.60	21.52	14.90	7.62	21.49	13.53
20	31.17	59.74	46.68	23.04	37.20	35.01	12.59	48.39	34.26	10.01	38.94	29.15
30	48.07	72.02	65.32	42.18	82.51	51.04	26.54	60.57	39.79	32.76	40.76	37.57
40	79.35	147.54	99.41	51.09	104.95	89.66	49.66	66.86	54.70	45.55	59.76	54.04
50	104.57	264.13	149.74	111.29	212.71	148.73	65.76	111.53	95.16	62.45	101.14	88.96
60	186.75	275.98	224.15	176.53	366.20	222.56	97.79	164.09	132.97	89.51	165.51	129.06
70	249.68	414.22	319.13	146.84	435.72	307.02	138.45	241.01	176.21	109.71	183.81	174.98
80	386.22	686.22	447.44	158.80	477.95	331.60	186.96	322.03	203.18	154.84	252.37	201.69
90	452.25	831.66	495.27	252.16	519.89	367.36	245.29	443.03	264.53	213.74	289.62	252.48
100	530.29	998.66	691.39	384.23	942.97	669.88	367.54	573.86	489.27	302.54	605.31	486.57

Table 12 Comparison of makespan obtained with 10 VMs

Task	Min–Max			PSO-LDIW			HPSO-SA			OTB-CSO		
	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
10	18.39	46.12	35.16	6.88	109.45	30.78	6.99	49.05	25.81	3.96	11.64	10.35
20	49.20	181.39	117.90	40.01	156.25	78.79	22.72	131.01	53.19	16.01	76.40	49.52
30	79.01	228.89	180.37	53.47	178.31	107.10	49.59	114.12	74.61	36.01	134.56	69.91
40	193.75	411.16	292.35	83.30	375.12	196.53	113.00	367.91	186.16	85.49	218.91	138.62
50	246.92	555.20	387.62	91.72	625.64	297.65	119.69	536.23	248.60	111.95	312.21	231.63
60	312.21	685.31	481.96	289.94	750.73	402.87	216.94	600.68	379.09	138.61	416.26	280.82
70	358.13	762.33	632.76	319.00	808.21	593.72	296.40	699.18	401.94	168.25	427.42	336.98
80	485.75	876.51	732.71	407.97	962.66	648.79	368.27	993.26	587.01	202.20	667.46	483.72
90	503.44	925.49	798.53	408.68	1079.99	701.35	477.77	1049.37	623.80	390.22	717.33	578.06
100	628.76	1453.88	955.72	559.06	1193.01	755.86	509.05	1086.56	685.56	490.20	741.29	595.21

Table 13 Comparison of makespan obtained with 5 VMs

Task	Min–Max			PSO-LDIW			HPSO-SA			OTB-CSO		
	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
10	12.77	32.28	25.55	15.85	45.21	32.18	8.25	44.41	28.70	11.47	43.12	23.14
20	36.95	246.82	140.82	30.22	530.49	155.88	30.91	214.22	102.84	42.93	163.65	107.13
30	177.06	395.15	290.05	119.33	537.62	245.55	46.14	562.16	254.96	74.21	173.76	120.55
40	179.99	459.46	497.68	177.35	645.69	464.95	145.31	699.70	332.35	124.25	448.59	268.27
50	347.17	1624.31	1107.56	439.73	1032.70	712.75	279.85	824.51	439.34	245.03	513.86	429.71
60	740.93	1757.52	1471.07	475.02	1624.55	1129.88	392.78	965.82	626.53	263.10	660.36	567.72
70	767.42	3972.42	1881.75	748.75	4817.92	1790.58	485.96	999.63	841.91	443.98	1492.49	798.82
80	828.75	4958.18	2610.02	816.39	5150.34	2260.56	551.09	1383.88	1009.14	634.27	1501.80	895.60
90	955.96	5649.55	3005.79	874.36	5272.27	2593.67	774.83	1525.99	1117.32	758.91	1683.33	1100.22
100	1680.51	7670.15	3761.25	972.00	6063.49	2891.74	848.09	4525.59	1812.64	802.45	1987.81	1231.64

the total tasks scheduled on VMs by achieving 62.53, 54.85 and 15.58% reduction over Min–Max, PSO-LDIW and HPSO-SA algorithms.

One of the main objectives of this article is to minimize the degree of imbalance. The degree of imbalance help discovers whether proposed algorithm is able to balance

Table 14 Degree of imbalance (DI) with 20 VMs

Task	Min–Max			PSO-LDIW			HPSO-SA			OTB-CSO		
	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
10	2.11	16.01	4.53	0.19	7.01	2.20	0.01	4.16	1.16	0.11	3.0	1.17
20	4.34	28.87	6.67	0.11	16.47	5.18	0.03	5.06	3.52	0.21	2.06	2.70
30	5.67	32.13	12.27	0.26	20.13	7.62	0.10	7.30	4.68	0.31	4.55	3.47
40	7.13	35.73	11.70	0.11	34.73	15.02	0.11	23.88	9.29	0.51	10.62	6.02
50	4.18	68.83	25.84	0.11	42.83	19.94	0.21	32.40	22.93	0.67	13.11	6.96
60	5.11	89.47	48.21	0.11	59.53	33.24	0.51	41.08	13.02	1.01	18.82	13.79
70	7.11	118.73	66.76	0.11	128.73	57.64	0.71	71.79	30.24	2.11	28.79	16.47
80	5.89	213.67	87.37	0.11	183.39	74.49	0.91	96.53	35.07	3.11	43.14	36.04
90	9.10	368.54	95.64	0.10	245.54	87.11	1.11	110.83	41.56	3.39	25.19	19.00
100	9.78	676.18	134.56	0.11	95.59	48.18	2.11	123.61	68.50	4.11	42.96	42.29

Table 15 Degree of imbalance (DI) with 10 VMs

Task	Min–Max			PSO-LDIW			HPSO-SA			OTB-CSO		
	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
10	6.67	27.66	9.37	0.36	11.66	3.37	0.51	9.17	2.42	0.27	10.41	5.63
20	5.28	33.67	14.75	0.11	12.02	4.84	0.35	6.74	4.85	0.11	7.94	3.59
30	4.27	38.48	19.12	0.01	4.48	5.12	0.12	21.26	8.75	0.22	15.15	14.38
40	5.29	45.73	21.57	0.29	37.73	18.14	0.23	23.8	15.60	0.10	48.79	25.53
50	3.01	50.89	26.76	0.01	45.99	23.31	0.24	44.22	22.28	0.22	12.73	16.08
60	6.19	56.38	29.71	0.10	49.38	34.47	0.26	69.09	38.86	0.10	25.93	21.53
70	5.32	357.08	159.59	0.32	57.08	79.59	0.11	129.56	46.39	0.11	15.32	29.62
80	9.67	187.29	73.46	0.11	98.29	54.46	0.11	144.30	43.61	0.39	59.88	37.57
90	4.60	273.39	62.91	0.8	160.39	59.91	0.22	119.59	54.83	0.23	48.97	46.68
100	6.43	789.05	192.56	0.11	293.05	78.78	0.22	167.90	67.52	0.11	52.61	49.94

Table 16 Degree of imbalance (DI) with 5 VMs

Task	Min–Max			PSO-LDIW			HPSO-SA			OTB-CSO		
	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
10	3.67	46.78	29.36	0.11	6.44	3.23	0.25	5.14	4.21	0.32	6.58	6.22
20	13.28	72.67	34.75	0.34	9.88	13.27	0.75	13.39	15.90	0.11	23.44	43.37
30	12.17	283.65	129.43	0.32	44.95	34.46	1.09	20.69	19.82	0.01	38.33	91.81
40	17.32	345.73	134.71	0.56	60.47	111.24	0.15	16.10	22.55	0.35	17.90	38.32
50	15.21	546.89	243.54	0.11	45.82	90.12	0.23	67.41	71.49	0.10	8.83	41.80
60	16.19	587.24	328.72	0.21	73.27	130.37	0.12	117.05	81.19	0.41	87.28	58.94
70	27.90	657.08	487.59	0.21	88.09	369.53	3.42	367.76	218.29	0.11	132.96	98.40
80	11.25	496.29	294.67	0.20	539.61	375.15	0.11	402.83	286.71	0.23	179.14	184.22
90	5.60	673.39	467.56	0.33	739.75	334.37	0.11	588.43	264.05	0.01	106.66	187.47
100	10.43	889.05	492.38	0.23	817.91	431.27	0.34	714.09	329.62	0.20	119.12	120.70

tasks across VM and return minimum execution time. The smaller degree of imbalance, better the performance of the algorithm. Percentage improvement achieved based on the degree of imbalance is shown in Table 18. For 20 VMs used, OTB-CSO was able to balance tasks based on the degree of imbalance on VMs with an improvement of 70.03, 62.03 and 35.68% over Min–Max, PSO-LDIW and HPSO-SA. For 10 VMs, improvement of 58.91, 30.78 and

Table 17 OTB-CSO performance improvement (%) based on makespan

	Min–Max	PSO-LDIW	HPSO-SA	OTB-CSO
With 20 virtual machines				
Total average makespan	2569.19	2243.66	1504.97	1468.03
PIR (%) over I-Min–Max		12.67	41.42	42.86
PIR (%) over PSO-LDIW			32.92	34.57
PIR (%) over HPSO-SA				2.45
With 10 virtual machines				
Total average makespan	4615.08	3813.44	3265.77	2774.82
PIR (%) over I-Min–Max		17.37	29.24	39.87
PIR (%) over PSO-LDIW			14.36	27.23
PIR (%) over HPSO-SA				15.03
With 5 virtual machines				
Total average makespan	14,791.54	12,277.74	6565.73	5542.80
PIR (%) over I-Min–Max		16.99	55.61	62.53
PIR (%) over PSO-LDIW			46.52	54.85
PIR (%) over HPSO-SA				15.58

Table 18 OTB-CSO performance improvement (%) based on degree of imbalance (DI)

	Min–Max	PSO-LDIW	HPSO-SA	OTB-CSO
With 20 virtual machines				
Total average DI	493.55	398.03	229.97	147.91
PIR (%) over I-Min–Max		19.35	53.40	70.03
PIR (%) over PSO-LDIW			42.22	62.83
PIR (%) over HPSO-SA				35.68
With 10 virtual machines				
Total average DI	609.80	361.99	305.11	250.55
PIR (%) over I-Min–Max		40.63	49.97	58.91
PIR (%) over PSO-LDIW			15.71	30.78
PIR (%) over HPSO-SA				17.88
With 5 virtual machines				
Total average DI	2642.71	1893.01	1313.82	871.25
PIR (%) over I-Min–Max		28.37	50.29	67.03
PIR (%) over PSO-LDIW			30.60	53.98
PIR (%) over HPSO-SA				33.69

17.88% over Min–Max, PSO-LDIW and HPSO-SA algorithms was achieved. Likewise, for 5 VMs used it balances task across VMs by achieving 67.03, 53.98 and 33.69% over Min–Max, PSO-LDIW and *HPSO-SA* algorithms.

The outlined performance of OTB-CSO over three existing algorithms was attributed to the incorporation of Taguchi orthogonal approach at CSO tracing mode. It was discovered search process within tracing mode of the CSO traversed the best solution regions; hence, position vector becomes capable of ensuring better convergence while eliminating solution that is non-best and ensuring a more solution that is active is introduced. This eventually pushes away the searching processes from local optima [13]. The orthogonal array of Taguchi also helps in guiding the metaheuristic to obtain good solution. It was discovered for 100 cats spread across solution area, and the tracing mode phase was able to empower search process without capitalizing on only best solution regions that may end up trapping the search toward a certain region. However, our proposed OTB-CSO tracing mode has actually utilized an orthogonal array of Taguchi method to return search results more suitable by enabling improved searching efficiency[31]. This followed the best mapping of the task across VMs that reduced makespan and degree of imbalance. Hence, the performance of OTB-CSO is believed to be attributed to the effectiveness of the tracing mode. Although results in Tables 11, 12, 13, 14, 15, 16, 17 and 18 show that of 20, 10 and 5 VMs used, it was used to discover whether there exists any impact on the algorithm apart from input tasks. Obtained results have significantly

Table 19 Results of performance comparison of CSO, EPCSO [31] and OTB-CSO with 20 VMs based on average execution time

Tasks	CSO	EPCSO	OTB-CSO
10	28.66	16.45	13.53
20	37.09	28.53	29.15
30	58.62	42.71	37.57
40	73.34	59.46	54.04
50	135.61	96.43	88.96
60	196.24	134.72	129.06
70	253.89	198.98	174.98
80	345.11	223.45	201.69
90	416.13	259.52	252.48
100	614.34	496.02	486.57

shown that VM numbers do not show any impact based on the performance of our proposed algorithm, rather proposes algorithm outperformed better in both makespan and degree of imbalance. This showed proposed OTB-CSO is well balanced and enhanced with effectiveness in optimizing task scheduling as compared to existing three algorithms.

9 Conclusion

The performance of cloud computing datacenter toward providing better computing service that meets task deadline can be measured based on makespan and degree of imbalance. Models were proposed by existing researchers,

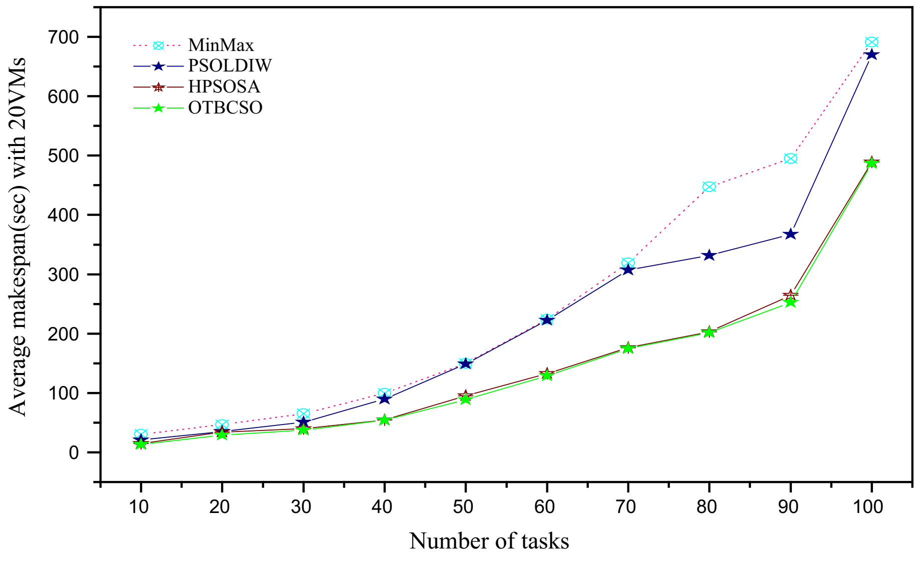

Fig. 3 Comparison between algorithms based on makespan with 20 VMs

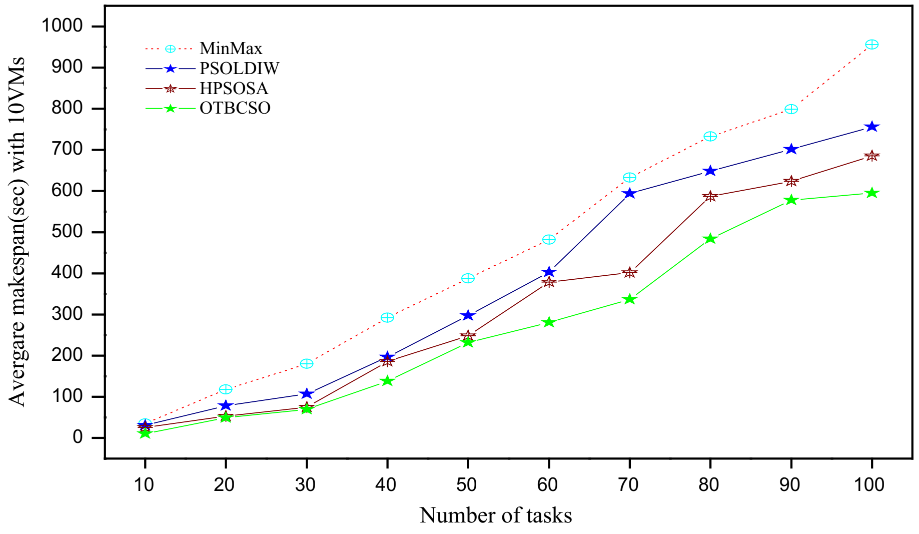

Fig. 4 Comparison between algorithms based on makespan with 10 VMs

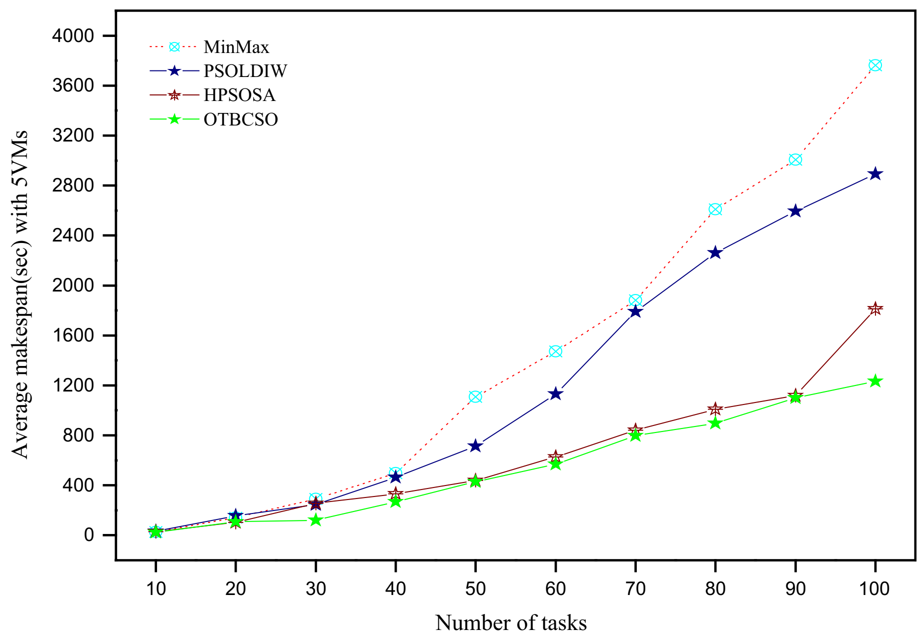

Fig. 5 Comparison between algorithms based on makespan with 5 VMs

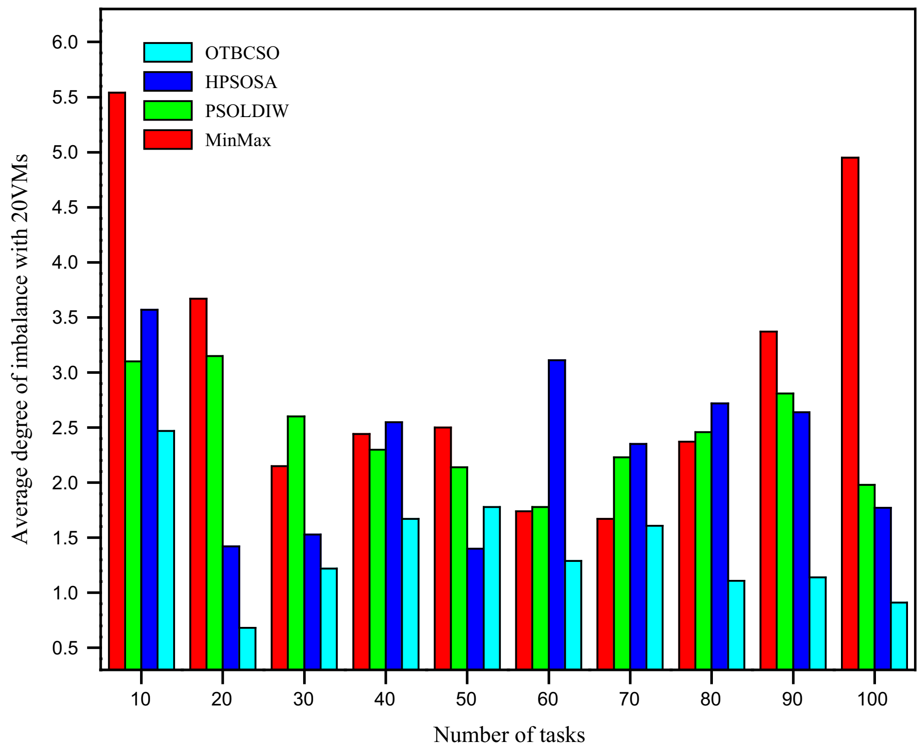

Fig. 6 Comparison between algorithms based on degree of imbalance with 20 VMs

and scheduling methods were applied to minimize the models. In this paper, we presented an OTB-CSO algorithm for optimum task scheduling in a dynamic cloud environment based on Taguchi approach. The purpose is to minimize makespan and degree of imbalance of the total task scheduled on VMs based on proposed model. Implementation of the algorithm was carried out on CloudSim tool with three groups of heterogeneous virtual machine (VM) instances (20, 10, 5 VMs). Performance evaluation was carried out based on makespan, degree of imbalance and percentage (%) improvements rate. Results obtained by OTB-CSO showed outstanding performance. The percentage improvements (%) recorded OTB-CSO algorithm outperformed Min–Max, PSO-LDIW and HPSO-SA in both makespan and degree of imbalance. The OTB-CSO was later compared with real version of CSO and EPCSO

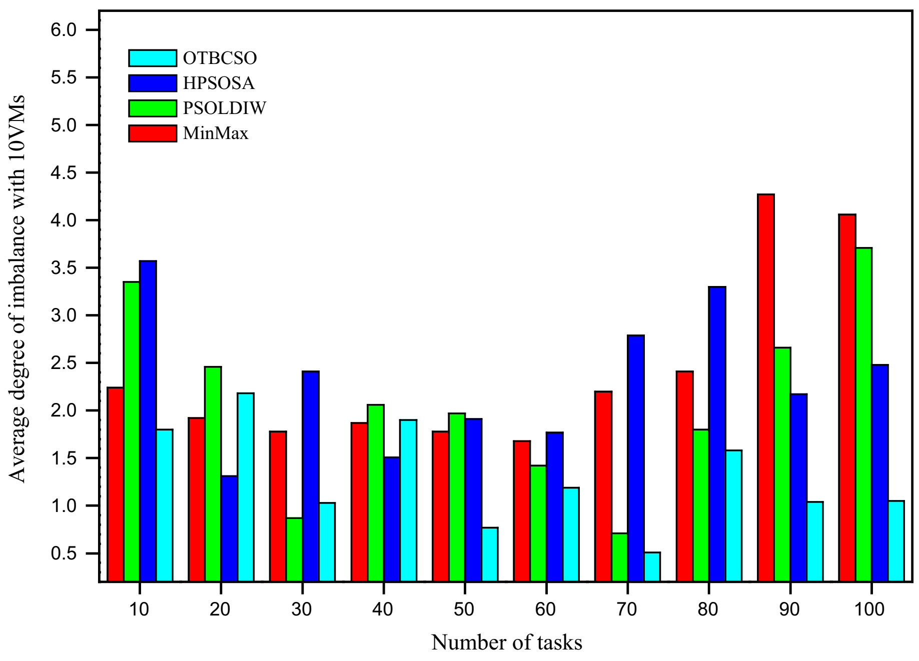

Fig. 7 Comparison between algorithms based on degree of imbalance with 10 VMs

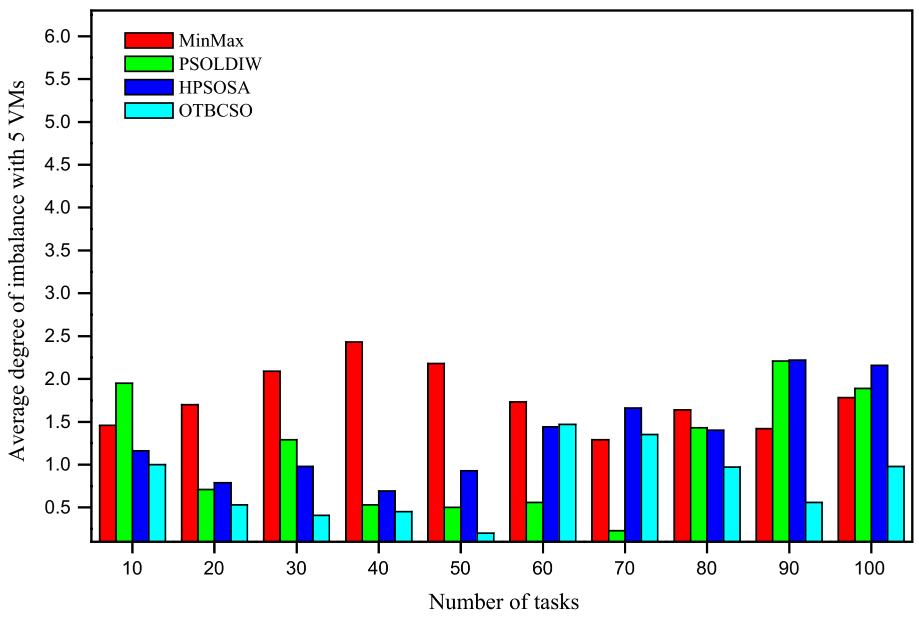

Fig. 8 Comparison between algorithms based on degree of imbalance with 5 VMs

algorithm. OTB-CSO employed local search ability of Taguchi optimization method to improve convergence speed by achieving solutions that are optimum as it returned minimum makespan and degree of imbalance was discovered. A more study of other computation-based and network-based parameters is required and also the integration of more advanced concepts such as virtual machine migration, energy consumption, multi-objective optimization and to optimize further the algorithm in order to scale with larger workloads.

Acknowledgement The first author will like to acknowledge Nigerian Tertiary Education Trust Fund (Tetfund) for their financial support in carrying out this research.

References

1. Bey KB, Benhammadi F, Benaissa R (2015) Balancing heuristic for independent task scheduling in cloud computing. In: Proceedings of the 2015 12th International Symposium on Programming and Systems (ISPS), IEEE, pp 1–6

2. Leena VA, Ajeena BAS, Rajasree MS (2016) Genetic algorithm based bi-objective task scheduling in hybrid cloud platform. Int J Comput Theory Eng 8(1):7–13
3. Raza HM, Adenola FA, Nafarieh A, Robertson W (2015) The slow adoption of cloud computing and IT workforce. Proc Comput Sci 52(2015):1114–1119
4. Durao F, Carvalho SFJ, Fonseka A, Garcia CV (2014) Systematic review on cloud computing. J Supercomput 68:1321–1346
5. Tsai J-T, Liu T-K, Ho W-H, Chou J-H (2008) An improved genetic algorithm for job-shop scheduling problems using Taguchi-based crossover. Int J Adv Manuf Technol 38:987–994
6. Banerjee S, Adhikari M, Kar S, Biswas U (2015) Development and analysis of a new cloudlet allocation strategy for QoS improvement in cloud. Arab J Sci Eng 40(5):1409–1425
7. Domanal GS, Reddy GRM (2014) Optimal load balancing in cloud computing by efficient utilization of virtual machines. In: Proceedings of the Sixth International Conference on Communication Systems and Networking (COMSNETS), IEEE, pp 1–4
8. Dhinesh BLD, Krishna PV (2013) Honey bee behavior inspired load balancing of tasks in cloud computing environments. J Appl Soft Comput 13(5):2292–2303
9. Ramezani F, Lu J, Hussain FK (2014) Task-based system load balancing in cloud computing using particle swarm optimization. Int J Parallel Prog 42:739–754
10. Shobana G, Geetha M, Suganthe RC (2014) Nature inspired preemptive task scheduling for load balancing in cloud datacenter. In: Proceedings of the International Conference on Information Communication and Embedded Systems (ICICES), IEEE, pp 1–6
11. Tsai J-T, Fang J-C, Chou J-H (2013) Optimized tasks scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. Comput Oper Res 40(2013):3045–3055
12. Madni SHH, Latiff MSA, Coulibaly Y, Abdulhamid SM (2016) Resource scheduling for infrastructure as a service (IaaS) in cloud computing: challenges and opportunities. J Netw Comput Appl 68:173–200
13. Abdullahi M, Ngadi MA, Abdulhamid SM (2016) Symbiotic organism search optimization based task scheduling in cloud computing environment. Future Gener Comput Syst 56(2016):640–650
14. Jung S-M, Kim N-U, Chung T-M (2013) Applying scheduling algorithms with QoS in the cloud computing. In: Proceedings of the International Conference on Information Science and Applications (ICISA), IEEE, pp 1–2
15. Tsai C-W, Huang W-C, Chiang M-H, Chiang M-C, Yang C-S (2014) A hyper-heuristic scheduling algorithm for cloud. IEEE Trans Cloud Comput 2(2):236–250
16. Abdullahi M, Ngadi MS (2016) Hybrid symbiotic organisms search optimization algorithm for scheduling of tasks on cloud computing environment. PLoS ONE 11(6):e0158229. doi:10.1371/journal.pone.0158229
17. Awad AI, EL-Hefnawy NA, Abdel_kader HM (2015) Dynamic multi-objective task scheduling in cloud computing based on modified particle swarm optimization. Adv Comput Sci Int J 4(5):110–117
18. Jena RK (2015) Multi-objective task scheduling in cloud environment using nested PSO framework. Proc Comput Sci J 57(2015):1219–1227
19. Liu C-Y, Zou C-M, Wu P (2014) A task scheduling algorithm based on genetic algorithm and ant colony optimization in cloud computing. In: Proceedings of the 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES), IEEE, pp 68–72
20. Netjinda N, Sirinaovakul B, Achalakul T (2012) Cost optimization in cloud provisioning using particle swarm optimization. In: Proceedings of the 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), IEEE, pp 1–4
21. Ramezani F, Lu J, Taheri J, Hussain FK (2015) Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments. World Wide Web 18(6):1737–1757
22. Singh S, Kalra M (2014) Scheduling of independent tasks in cloud computing using modified genetic algorithm. In: Proceedings of the Sixth International Conference on Computational Intelligence and Communication Networks (CICN), IEEE, pp 565–569
23. Tawfeek AM, El-Sisi A, Keshk EA, Torkey AF (2013) An ant algorithm for cloud task scheduling. In: Proceedings of the International Workshop on Cloud Computing and Information Security (CCIS 2013), IEEE, pp 64–69
24. Wang J, Li F, Zhang L (2014) QoS preference awareness task scheduling based on PSO and AHP methods. Int J Control Autom 7(4):137–152
25. Wu Z, Ni Z, Gu L, Liu X (2010) A revised discrete particle swarm optimization for cloud workflow scheduling. In: Proceedings of the International Conference on Computational Intelligence and Security (CIS), IEEE, pp 184–188
26. Abdulhamid SM, Abd Latiff MS, Abdul-Salaam G, Madni SHH (2016) Secure scientific applications scheduling technique for cloud computing environment using global league championship algorithm. PLoS ONE 11(7):e0158102
27. Ashwin TS, Domanal SG, Guddeti RMR (2014) A novel bio-inspired load balancing of virtual machines in cloud environment. In: Proceedings of the IEEE International Conference on Cloud Computing in Emerging Networks (CCEM), IEEE, pp 1–4
28. Chu S-C, Tsai P-W (2007) Computational intelligence based on the behavior of cats. Int J Innov Comput Inf Control 3(2007):163–173
29. Bansal N, Maurya A, Kumar T, Singh M, Bansal S (2015) Cost performance of QoS-driven task scheduling in cloud computing. Proc Comput Sci J 57(2015):126–130
30. Pradhan PM, Panda G (2012) Solving multi-objective problems using cat swarm optimization. Int J Expert Syst Appl 39(2012):2956–2964
31. Tsai P-W, Pan J-S, Chen S-M, Lio B-Y (2012) Enhanced parallel cat swarm optimization based on Taguchi method. Expert Syst Appl 39(2012):6309–6319
32. Abd K, Abhary K, Marian R (2013) Simulation modelling and analysis of scheduling in robotic flexible assembly cells using Taguchi method. Int J Prod Res 52(9):2654–2666
33. Cavory G, Dupas R, Goncalves G (2001) A genetic approach to the scheduling of preventive maintenance tasks on a single product manufacturing production line. Int J Prod Econ 74(2001):135–146
34. Asefi H, Jolai F, Rabiee M, Araghi MET (2014) A hybrid NSGA-II and VNS for solving a bi-objective no-wait flexible flowshop scheduling problem. Int J Adv Manuf Technol 75(2014):1017–1033
35. Chang H-C, Chen Y-P, Liu T-K, Chou J-H (2015) Solving the flexible job shop scheduling problem with makespan optimization by using a hybrid Taguchi-genetic algorithm. IEEE J Mag 3:1740–1754
36. Caprilhan R, Kumar A, Stecke KE (2013) Evaluation of the impact of information delays on flexible manufacturing systems performance in dynamic scheduling environments. Int J Adv Manuf Technol 67(1):311–338
37. Taguchi G, Chowdhury S, Taguchi S (2000) Robust engineering. McGraw-Hill, New York
38. Bilgaiyan S, Sagnika S, Das M (2015) A multi-objective cat swarm optimization algorithm for workflow scheduling in cloud computing environment. Int J Soft Comput 10(1):37–45

39. Kalaiselvan G, Lavanya A, Natrajan V (2011) Enhancing the performance of watermarking based on cat swarm optimization method. In: Proceedings of the IEEE-International Conference on Recent Trends in Information Technology (ICRTIT), IEEE, pp 1081–1086
40. Pappula L, Ghosh D (2014) Linear antenna array synthesis using cat swarm optimization. Int J Electr Commun 68:540–549
41. Al-Salamah M (2015) Constrained binary artificial bee colony to minimize the makespan for single machine batch processing with non-identical job sizes. Appl Soft Comput 29(2015):379–385
42. Shojaee R, Faragardi RH, Alaee S, Yazdani N (2012) A new cat swarm optimization based algorithm for reliability-oriented task allocation in distributed systems. In: Symposium on Sixth International Telecommunications (IST), IEEE, pp 861–866
43. Xu R, Chen H, Li X (2012) Makespan minimization on single batch-processing machine via ant colony optimization. Comput Oper Res 39(2012):582–593
44. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R (2010) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp 41(1):23–50
45. Garey MR, Johnson DSA (2016) Guide to the theory of NP-completeness. WH Freemann, New York
46. Al-Olimat HS, Alam M, Green R, Lee KJ (2015) Cloudlet scheduling with particle swarm optimization. In: Fifth International Conference on Communication Systems and Network Technologies (CSNT), IEEE, pp 991–995
47. Eberhart RC, Shi Y (2000) Comparing inertia weights and constriction factors in particle swarm optimization. In: Proceedings of the IEEE Conference on Evolutionary Computation, ICEC, IEEE, pp 84–88
48. Abdulhamid SM, Abd Latiff MS, Madni SHH (2016) Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm. Neural Comput Appl. doi:10.1007/s00521-016-2448-8
49. El-Sisi AB, Tawfeek MA, Keshk AE, Torkey FA (2014) Intelligent method for cloud scheduling based on particle swarm optimization algorithm. In: Proceedings of the International Arab Conference on Information Technology (Acit2014), IEEE, pp 39–44
50. Zhou Z, Zhigang H (2014) Task scheduling algorithm based on greedy strategy in cloud computing. Open Cybern Syst J 8:111–114