Symbiotic Organism Search Optimization based Task Scheduling in Cloud Computing Environment

Mohammed Abdullahi[1,4], Md Asri Ngadi[2], Shafi'i Muhammad Abdulhamid[3]

[1,2] Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, Malaysia.
[3]Department of Cyber Security Science, Federal University of Technology Minna, Nigeria.
[4]Department of Mathematics, Ahmadu Bello University Zaria, Nigeria
[1,4] abdullahilwafu@abu.edu.ng, [2]dr.asri@utm.my,[3] shafii.abdulhamid@futminna.edu.ng

Abstract

Efficient task scheduling is one of the major steps for effectively harnessing the potential of cloud computing. In cloud computing, a number of tasks may need to be scheduled on different virtual machines in order to minimize makespan and increase system utilization. Task scheduling problem is NP-complete, hence finding an exact solution is intractable especially for large task sizes. This paper presents a Discrete Symbiotic Organism Search (DSOS) algorithm for optimal scheduling of tasks on cloud resources. Symbiotic Organism Search (SOS) is a newly developed metaheuristic optimization technique for solving numerical optimization problems. SOS mimics the symbiotic relationships (mutualism, commensalism, and parasitism) exhibited by organisms in an ecosystem. Simulation results revealed that DSOS outperforms Particle Swarm Optimization (PSO) which is one of the most popular heuristic optimization techniques used for task scheduling problems. DSOS converges faster when the search gets larger which makes it suitable for large-scale scheduling problems. Analysis of the proposed method conducted using t-test showed that DSOS performance is significantly better than that of PSO particularly for large search space.

Keywords: cloud computing, task scheduling, makespan, symbiotic organism search, mutualism, commensalism, parasitism, ecosystem

1. Introduction

In cloud computing, resources such as processors, memory, storage, and applications are provisioned as services, thereby changing the way IT resources are designed and acquired[1]. The cloud computing paradigm had greatly reduced the financial cost of acquiring hardware and software for application deployment as well as maintenance cost. Because of the high scalability, users are not bothered with imprecise forecasting of service scale which will amount to resource wastage if overprovisioned, and revenue loss if under provisioned[2, 3]. Cloud computing resources are shared among cloud clients through the concept of virtualization. Virtualization allows many remote running environments to be safely combined on physical servers for optimum utilization of physical resources and energy[2]. Virtual Machine (VM) is a vital component of software

stacks in the cloud data center. Cloud data are located across servers which are interconnected through networked resources and accessed via virtual machines. Amazon Elastic Computing Cloud (Amazon EC2) [4] is an example of cloud platform that provides infrastructure services in the form of VMs. One of the cardinal objectives of cloud computing is maximization of revenue both on the part of the cloud provider and the user. Task scheduling has evolved as one the focus in cloud computing[5] since inefficient task scheduling can lead to revenue loss, performance degradation, and breach of Service Level Agreement (SLA). Therefore, efficient scheduling algorithms are required to minimize both computation-based metrics such as response time, system utilization, makespan, system throughput and network-based metrics such as network communication cost, traffic volume, round trip time, data communication cost[6]. These metrics are central to monitoring cloud activities in order to address issues like load balancing, energy efficiency, SLA and Quality of Service (QoS) guarantee, fault tolerance[6].

There are encouraging research results for efficient task scheduling in the cloud, but task scheduling problems are still NP-complete [7]. Most of the task scheduling algorithms used in cloud computing are rule based [8-10] because they are easy to implement. Rule based algorithms perform poorly when it comes to complex task scheduling problems [5]. The most common metaheuristic techniques applied to task scheduling problems in grid and cloud computing are Genetic Algorithm (GA) [11-15], Particle Swarm Optimization (PSO) [16-26], and Ant Colony Optimization (ACO) [27-30]. PSO converges faster and obtain better solution than GA and ACO due its exploratory capability for finding optimal solutions [16, 31]. Owing to the better performance of PSO over ACO and GA, variants and hybrid versions of PSO have been used for benchmarking the proposed algorithm.

In this paper, DSOS is proposed and used to schedule a bag of tasks in a cloud environment, hence network communication cost and data transfer cost are not main optimization issues when dealing with independent tasks. In the experimental setup, four data set categories, normal, left-skewed, right-skewed, and uniform distributions, were used to test the suitability of the proposed method. We used different forms of distributions to gain insight into the performance trend of the proposed method. Web applications such as web services are usually run for a long time and their CPU demand is variable. Moreover, High Performance Computing (HPC) applications have short life span and place a high demand on CPU. Furthermore, chosen statistical models for task sizes represents different scenarios of concurrently scheduling HPC and web applications. Uniform distribution depicts a situation where HPC and web applications are of the same magnitude. Left skewed distribution represents a situation where HPC applications to be scheduled are more than web applications and right skewed distribution represent the other way round. Uniform distribution represents a scenario where a single application type is scheduled.

The proposed algorithm can be applied to obtain optimal solutions to well defined problems on discrete space including, production planning, scheduling, inventory control, optimization of network synthesis and design problems etc. These problems arise in real-world situations like management, engineering, telecommunications etc.

The main contributions of the paper are:

- Clearer presentation of SOS procedures.
- Design and implementation of discrete version of SOS algorithm for scheduling of tasks in a cloud computing environment.
- Evaluation of the proposed method using makespan, response and degree of imbalance among VMs as performance metrics
- Statistical validation of the obtained results against that of PSO using significance test.

The organization of the remainder of the paper is as follows. Metaheuristic algorithms applied to task scheduling problems in the cloud and SOS are presented in Section 2. Section 3 describes the problem formulation. Design of proposed algorithm and its description is presented in Section 4. Results of simulation and its discussion are in Section 5. Section 6 presented a summary and conclusion of the paper.

2. Related Work

2.1. Metaheuristic Algorithms in cloud Scheduling

Metaheuristic methods [11, 13, 25, 27, 29, 32-37] have been applied to solve task assignment problems in order to reduce makespan and response time. The methods have proven to find an optimum mapping of tasks to resources which reduce the cost of computation, improve quality of service, and increase utilization of computing resources. ACO, PSO, GA, and their variants are the mostly commonly used nature inspired population based algorithms in the cloud. PSO outperforms GA and ACO in most situations [16, 31] and has faster execution time. PSO is simple to implement as compared to GA and ACO respectively. Workflow scheduling problems have been widely studied using PSO [25, 26, 38-40] with the aim of reducing communication cost and makespan. Scheduling of Independent tasks has also been studied in cloud using PSO [18, 33, 34, 41, 42] and it has proved to ensure minimal makespan. Improved and hybrid versions [5, 20, 25, 40-42] of PSO were also proposed for scheduling of tasks in the cloud, and they obtained better solution than those of ACO and GA.

2.1.1. Symbiotic Organism Search Algorithm

Symbiotic Organism Search (SOS) algorithm, a novel population-based metaheuristic algorithm, was presented in [43] for solving numerical optimization problems on a continuous real space. SOS mimics the symbiotic associations (mutualism, commensalism, and parasitism) among different species in an ecosystem. Mutualism simply means the relationship between different species where both individuals benefit from the association. Commensalism is the association of two different species where one benefits from the union and the other is not harmed while in parasitism relation one species benefits and other is harmed. Each member of the organism within an ecosystem is represented by a vector in the solution plane. Each organism in the search space is assigned a value which suggests the extent of adaptation to the sought objective. The Algorithm repeatedly uses a population of the possible solutions to converge to an optimal position where the global optimal solution lies. The algorithm used mutualism, commensalism, and parasitism mechanisms to update the positions of the solution vector in the search space.

SOS is a repetitive process for an optimization problem [44] given in definition 1.1. The procedure keeps a population of organisms that depict the candidate solutions of the studied problem. The relevant information concerning the decision variables and a fitness value is encapsulated into the organism as an indicator of its performance. Essentially, the trajectories of the organisms are modified using the phases of the symbiotic association.

Definition 1.1
Given a function $f: D \rightarrow \Re$ find $X' \in D: \forall X \in D \ f(X') \leq or \geq f(X). \ (\leq)(\geq)$ minimization (maximization).
where f is an objective function to be optimized and D represents the search space while the elements of D are the feasible solutions. X is a vector of optimization variables $X = \{x_1, x_2, x_3, ..., x_n\}$. An optimal solution is a feasible solution X' that optimize f.

2.1.2. Procedures of Symbiotic Organism Search

The steps of the Symbiotic Organism Search algorithm are given below:
Step 1: Ecosystem initialization
Initial population of the ecosystem is generated and other control variables such as ecosystem size, maximum number of iterations are specified. The positions of the organisms in the solution space are represented by real numbers.
Step 2: Selection of the organism with the best fitted objective function represented as x^{best}
Step 3: Mutualism phase
In *ith* iteration, an organism x_j is randomly selected from the ecosystem to interact with an organism x_i for mutual benefit with $i \neq j$ according to (1) and (2) respectively.

$$x_i^{'} = x_i + r^{'}\left(x^{best} - \left(\left.(x_i + x_j)\right/2\right) f_1\right) \quad (1)$$

$$x_j^{'} = x_j + r^{"}\left(x^{best} - \left(\left.(x_i + x_j)\right/2\right) f_2\right) \quad (2)$$

where $r^{'}$ and $r^{"}$ are uniformly generated random numbers between 0 and 1. f_1 and f_2 are the mutual benefit factors resulting from the association between the organisms in the relationship. The two organisms may benefit equally from the mutual relationship or one may benefit more than the other. Values of f_1 and f_2 are determined randomly as either 1 or 2. 1 denotes partial benefit while 2 indicates full benefit. The term $\left(\left.(x_i + x_j)\right/2\right)$ defines the extent of adaptation of the two specie to ecosystem. The fitness function

$f(x_i^{'})$ and $f(x_j^{'})$ are evaluated, x_i is updated to $x_i^{'}$ using (1) only if $f(x_i^{'})$ is greater than $f(x_i)$ and x_j is updated to $x_j^{'}$ using (2) only if $f(x_j^{'})$ is greater than $f(x_j)$. x^{best} is the organism with highest fitness function so far indicating highest degree of adaptation for survival.

Step 4: Commensalism phase

In *ith* iteration, an organism x_j is randomly selected from the population to interact with x_i where $i \neq j$. In this phase, x_i benefits from x_j but x_j neither gains nor loses from the interaction. The interaction is modelled according to (3).

$$x_i^{'} = x_i + r^{'}\left(x^{best} - x_j\right) \qquad (3)$$

where $r^{'}$ is a uniformly generated random numbers between -1 and 1. The fitness function $f(x_i^{'})$ is evaluated and x_i is updated to $x_i^{'}$ using (3) only if $f(x_i^{'})$ is greater than $f(x_i)$.

Step 5: Parasitism phase

In *ith* iteration, a parasite vector x^p is created by mutating x_i using a randomly generated number in the range of the decision variables under consideration and an organism x_j with $i \neq j$ is selected randomly from the population to serve as host to x^p. If the fitness value $f(x^p)$ is greater than $f(x_j)$, then x^p will replace x_j, otherwise x^p is discarded.

Steps 2 through 5 are repeated until stopping criterion is reached.

Step 6: Stopping criterion

The pseudocode of Symbiotic Organism Search is presented as Algorithm 1.

Algorithm 1 Symbiotic Organism Search Algorithm

Create and Initialize the population of organisms in ecosystem $X = \{x_1, x_2, x_3, ..., x_N\}$
Set up stopping criteria
$iteration\_number \leftarrow 0$
$x^{best} \leftarrow 0$
Do
    $iteration\_number \leftarrow iteration\_number + 1$
    $i \leftarrow 0$
        Do
            $i \leftarrow i + 1$
            For $j = 1$ to N
                If $f(x_j) > f(x^{best})$ **Then** // $f(x)$ is the fitness function

$x^{best} \leftarrow x_j$

End if

End for

//mutualism phase

Randomly select x_j with $i \neq j$

$f_1 \leftarrow 1 or 2$

$f_2 \leftarrow 1 or 2$

$r' \leftarrow rand(0,1)$

$r'' \leftarrow rand(0,1)$

$x_i' \leftarrow x_i + r'\left(x^{best} - \left(\left(x_i + x_j\right)/2\right) f_1\right)$

$x_j' \leftarrow x_j + r''\left(x^{best} - \left(\left(x_i + x_j\right)/2\right) f_2\right)$

If $f(x_i') > f(x_i)$ **Then**

$x_i \leftarrow x_i'$

End if

If $f(x_j') > f(x_j)$ **Then**

$x_j \leftarrow x_j'$

End if

//commensalism phase

Randomly select x_j with $i \neq j$

$r' \leftarrow rand(-1,1)$

$x_i' \leftarrow x_i + r'\left(x^{best} - x_j\right)$

If $f(x_i') > f(x_i)$ **Then**

$x_i \leftarrow x_i'$

End if

//parasitism phase

Randomly select x_j with $i \neq j$

Create a parasite vector x^p from x_i using random number

If $f(x^p) > f(x_j)$ **Then**

$x_j \leftarrow x^p$

End if

While $i <= N$

While stopping condition is not true

3. Problem Description

When tasks to be scheduled are received by Cloud Broker (CB), Cloud Information Service (CIS) is queried to identify the services required to execute the received tasks from the user and then schedule the tasks on the discovered services. For instance, tasks $\{T_1, T_2, T_3, ..., T_n\}$ may be submitted to CB in a given time interval. The processing elements (Virtual Machines) are heterogeneous having varied processing speeds and memory, indicating that a task executed on different Virtual Machines (VMs) will result in varying execution cost. Suppose Virtual Machines $\{V_1, V_2, V_3, ..., V_m\}$ are available when the tasks are received by CB. The tasks are scheduled on the available VMs and execution of the tasks are done on the basis of First-Come First-Serve.

Our aim is to schedule tasks on VMs in order to achieve higher utilization of VMs with minimal makespan. As a result, Expected Time to Compute (ETC) of the tasks to be scheduled on each VM will be used by the proposed method to make schedule decision. ETC values are determined using the ratio of million instructions per second (MIPS) of a VM to the length of the task [45, 46] as illustrated in Table 1. ETC values are usually represented in matrix form [45, 46], where the number of tasks to be scheduled are represented by the rows of the matrix and number of available VMs are represented by the columns of the matrix. Each row of ETC matrix represents execution times of a given task for each VM, while each column represents execution times of each task on a given VM.

Our objective is to minimize the makespan by finding the best group of tasks to be executed on VMs. Let $C_{ij} (i \in \{1, 2, 3, ..., m\}, j \in \{1, 2, 3, ..., n\})$ be the execution time of executing jth task on *ith* VM where m is the number of VMs is and n is the number of tasks. The fitness value of each organism can be determined using (4), which determines the strength of the level of adaptation of the organism to the ecosystem.

$$fitness = \max\{C_{ij}, \forall all\ tasks\ j\ mapped\ to\ VM\ i\}\ i \in \{1, 2, 3, ..., m\} \quad (4)$$

Table 1
An example of an ETC matrix

	T_1	T_2	T_3	T_4
V_1	T_1/V_1	T_2/V_1	T_3/V_1	T_4/V_1
V_2	T_1/V_2	T_2/V_2	T_3/V_2	T_4/V_2
V_3	T_1/V_3	T_2/V_3	T_3/V_3	T_4/V_3

Table 2
An example of a task schedule

T_1	T_2	T_3	T_4
1	1	3	2

4. Discrete Symbiotic Organisms Search Algorithm

SOS was proposed and applied to solve continuous problems. The *Discrete Symbiotic Organism Search* (DSOS) is presented and applied to solve task scheduling problems in

cloud computing environment, with task scheduling formulated as a discrete optimization problem. The pseudocode and organogram of the proposed algorithm are presented in Algorithm 2 and Figure 1 respectively. The continuous version of SOS can be used to solve optimization problems for which the optimization variables $x_i,\ i=1,2,3,...n$ are continuous, $x_i \in \Re,\ i=1,2,3,...,n$ while DSOS can be applicable to optimization problems for which the optimization variables $x_i,\ i=1,2,3,...n$ are members of a countable set. SOS operates entirely on continuous optimization variables while DSOS operates on discrete optimization variables. In DSOS, the movement and position of organisms in the continuous space are mapped into developed discrete functions as shown in (7) through (12).

DSOS consists of three phases: initialization phase, repetition phase, and termination. The initialization phase generates the initial population of organisms. Each organism consists of D elements indicating candidate solutions and a fitness function to determine the extent of optimality of solutions. Therefore, each organism corresponds to a choice for task schedule encoded in a vector of dimension $1\times n$ as illustrated in Table 2, with n being the number of tasks. The elements of the vector are natural numbers in the range $[1,m]$, where m is the number of VMs. Suppose x_k is the position of the kth organism in the solution space; $x_k(j)$ signifies the virtual machine where task j is assigned by scheduler in the organism. The iterative phase mimics the mutualism, commensalism, and parasitism kinds of association to update the positions of the organisms. In mutualism and commensalism stages, x^{best} forms part of the update variables which act as the memory of the procedure. x^{best} is the best point an organism and its neighbors have visited so far. (5) through (8) are used to create modified positions of the selected organisms at mutualism phase.

$$s_1(p) \leftarrow x_i + r_1\left(x^{best} - \left(\frac{(x_i + x_j)}{2}\right) f_1\right) \tag{5}$$

$$s_2(p) \leftarrow x_j + r_2\left(x^{best} - \left(\frac{(x_i + x_j)}{2}\right) f_2\right) \tag{6}$$

$$x_i^{'}(q) \leftarrow \left\lceil s_1(p) \right\rceil \bmod m + 1 \tag{7}$$

$$x_j^{'}(q) \leftarrow \left\lceil s_2(p) \right\rceil \bmod m + 1 \tag{8}$$

$$\forall p \in \{1,2,3,...,n\}\ \forall q \in \{1,2,3,...,m\}$$

where $x_i^{'}$ and $x_j^{'}$ are the modified positions of the ith and jth members of the ecosystem with $i \neq j$, x_j is the randomly selected organism in ith iteration for $i \neq j, f_1, f_2 \in \{1,2\}$ are randomly determined and they represent the benefit factor from the mutual relationship, $r_1, r_2 \in rand(0,1)$ are uniformly generated random numbers in the specified

interval and $\lceil \rceil$ is a ceiling function. In commensalism phase, the modified position of the organism $x_i^{'}$ is obtained using (9) and (10)

$$s_3(p) \leftarrow r_3\left(x^{best} - x_j\right) \tag{9}$$

$$x_i^{'}(q) \leftarrow \lceil s_3(p) \rceil \bmod m + 1 \tag{10}$$

$$\forall p \in \{1,2,3,...,n\}\ \forall q \in \{1,2,3,...,m\}\ and\ j \neq i$$

where r_3 is a uniformly generated random number between 0 and 1.

In parasitism phase, the parasite vector x^p is created using (11) and (12).

$$s_4(p) \leftarrow r_4 x_i \tag{11}$$

$$x^p(q) \leftarrow \lceil s_4(p) \rceil \bmod m + 1 \tag{12}$$

$$\forall p \in \{1,2,3,...,n\}\ \forall q \in \{1,2,3,...,m\}\ and\ j \neq i$$

where r_4 is a uniformly generated random number between 0 and 1 .

Algorithm 2 Discrete Symbiotic Organism Search Procedure

Create and Initialize the population of organisms in ecosystem $X = \{x_1, x_2, x_3, ..., x_N\}$

Set up stopping criteria

$iteration\_number \leftarrow 0$

$x^{best} \leftarrow 0$

Do

$iteration\_number \leftarrow iteration\_number + 1$

$i \leftarrow 0$

Do

$i \leftarrow i + 1$

For $j = 1$ to N

If $f(x_j) > f(x^{best})$ **Then** // $f(x)$ is the fitness function

$x^{best} \leftarrow x_j$

End if

End for

//mutualism phase

Randomly select x_j with $i \neq j$

Update $x_i^{'}$ and $x_j^{'}$ according to equation (7) and (8)

If $f(x_i^{'}) > f(x_i)$ **Then**

$x_i \leftarrow x_i^{'}$

End if

If $f(x_j^{'}) > f(x_j)$ **Then**

$x_j \leftarrow x_j^{'}$

End if

//commensalism phase

Randomly select x_j with $i \neq j$

Update $x_i^{'}$ according to equation (10)

If $f(x_i^{'}) > f(x_i)$ **Then**

$x_i \leftarrow x_i^{'}$

End if

//parasitism phase

Randomly select x_j with $i \neq j$

Create a parasite vector x^p according to equation(12)

If $f(x^p) > f(x_j)$ **Then**

$x_j \leftarrow x^p$

End if

While $i <= N$

While stopping condition is not true

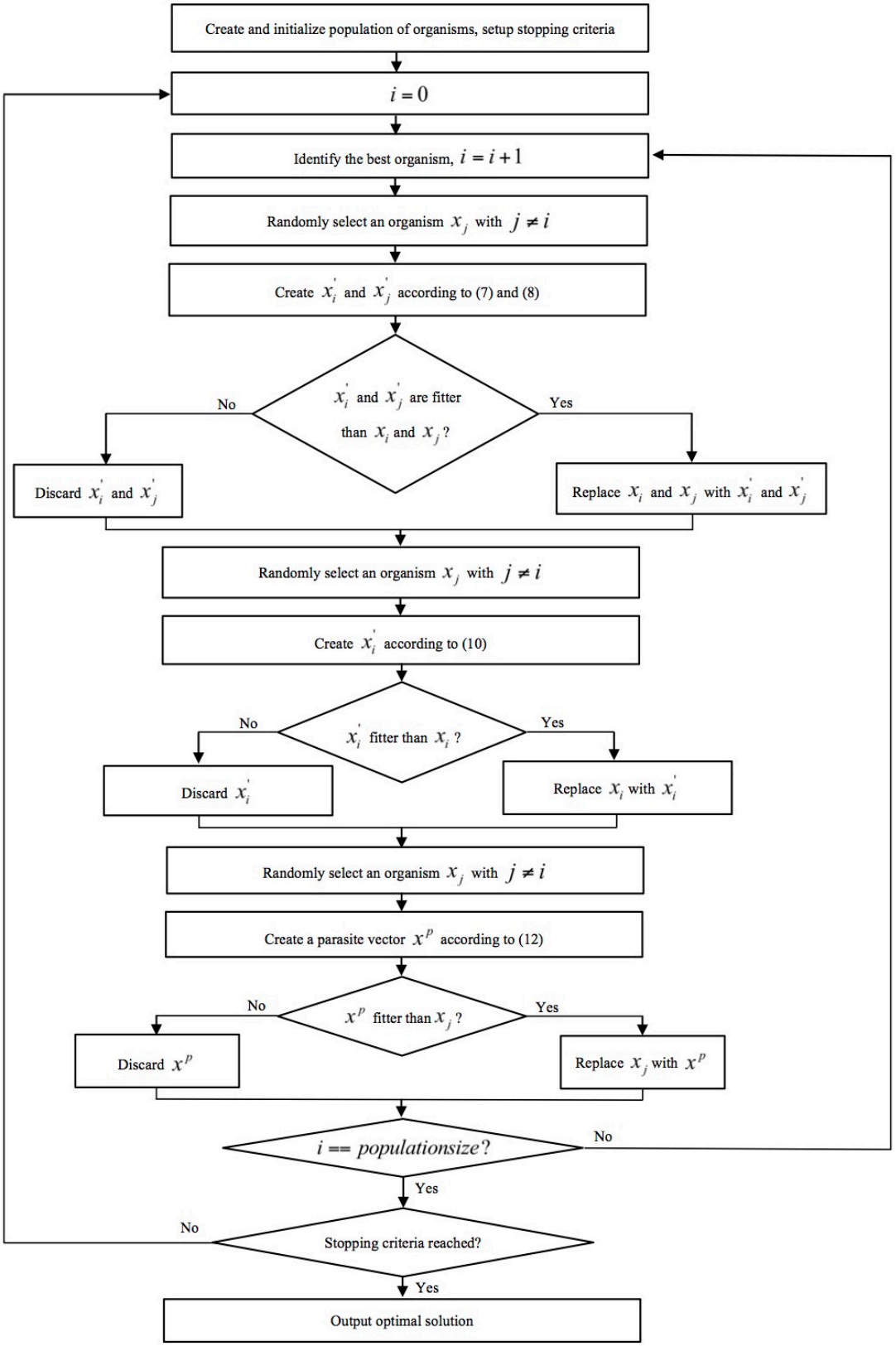

Fig. 1: Organogram of the DSOS

5. Simulation and Results

In order to test the performance of the proposed method, it was implemented in CloudSim[47] which is a toolkit for simulating Cloud computing scenarios. Two datacenters were created each containing two hosts respectively. Each host has 20 GB ram, 1 TB storage, 10 GB/s bandwidth and time-shared VM scheduling algorithm. One host is a dual-core machine while the other is a quad-core machine each with X86 architecture, Linux operating system, Xen virtual machine monitor(VMM), and cumulative processing power of 1000000 MIPS. 20 VMs were created, each with image size of 10 GB, 0.5 GB memory, 1 GB/s bandwidth and 1 processing element. The processing power of the VMs ranges from 1000 to 10000 MIPS respectively. Time-shared cloudlet scheduler and Xen VMM were used for all the VMs. Task sizes (cloudlet length) were generated from normal, left-skewed, right-skewed, and uniform distribution. Uniform distribution depicts more medium size tasks, and fewer small and large size tasks. Left-skewed represents a few small size tasks and more large size tasks while right-skewed is the opposite. Uniform distribution depicts an equal number of large, medium, and small size tasks. For each distribution, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 instances were generated. The larger instances will enable us to gain insight into the scalability of performance of the algorithms with large problem sizes. The parameter settings for SAPSO and DSOS are presented in Table 3. The choice of values for inertia weight and constriction factors (c1, c2) was based on [48].

Table 3
Parameter settings for PSO and SOS

Algorithm	Parameter	Value
PSO	Particle Size	100
	Self-recognition coefficient c1	2
	Social effect c2	2
	Static Inertia weight	0.9
	Variable Inertia weight	0.9-0.4
	Number of iterations	1000
SOS	Number of organisms	100
	Number of iterations	1000

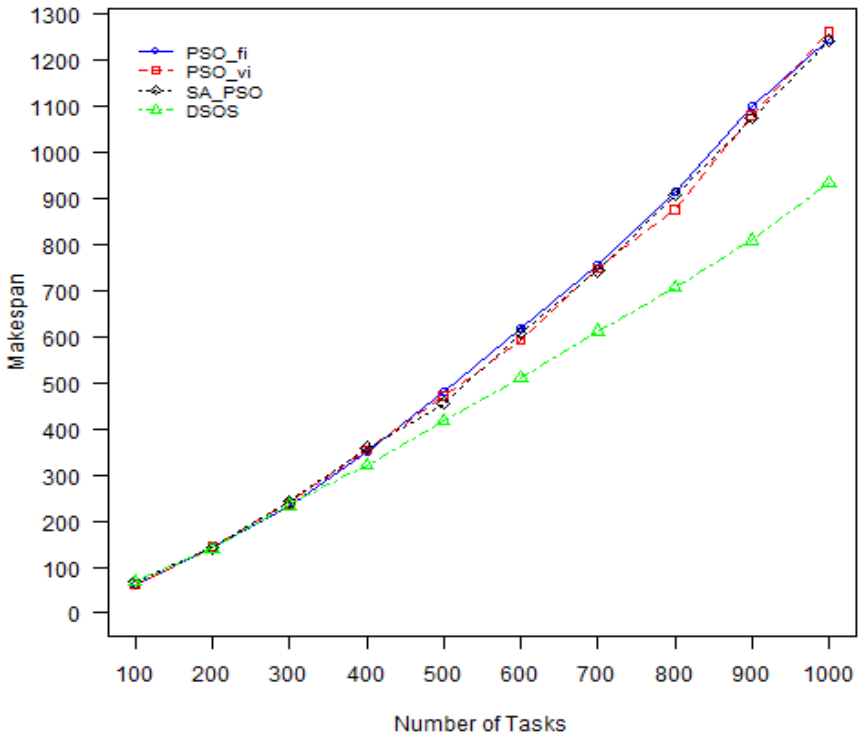

Fig. 2: Makespan (normal dist.)

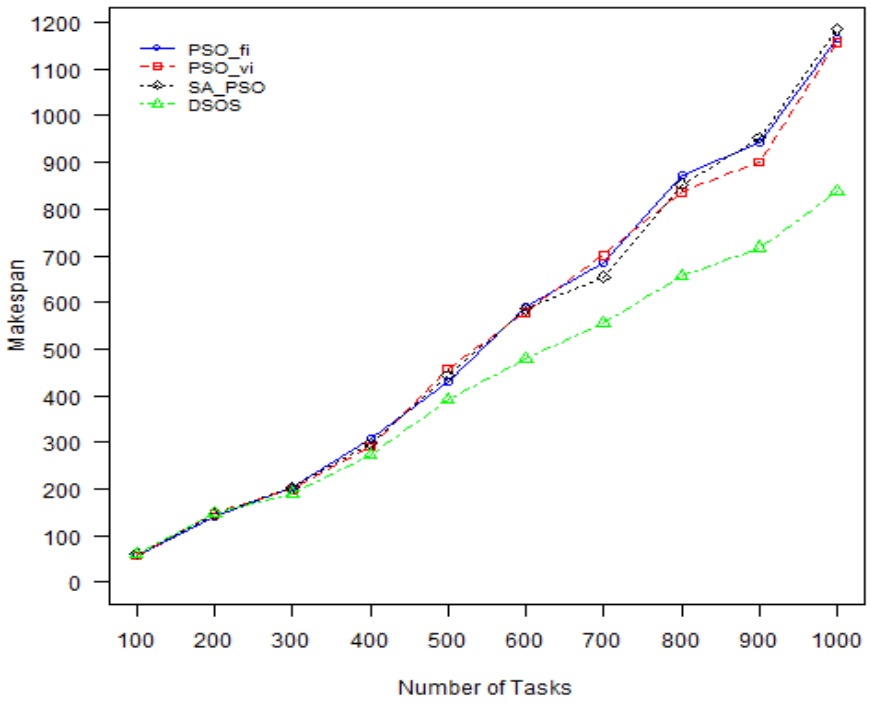

Fig. 3: Makespan (left skewed)

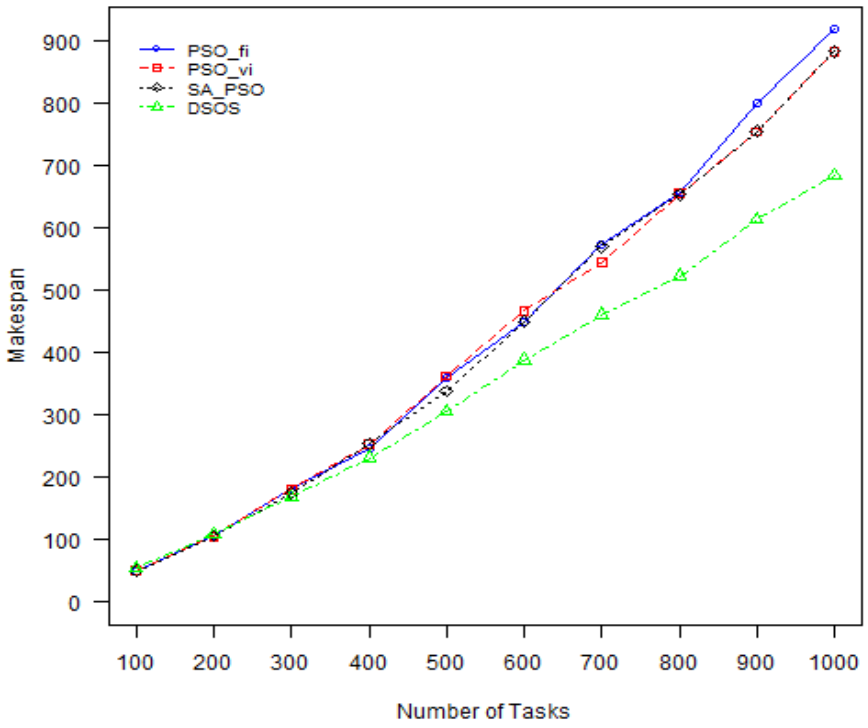

Fig. 4: Makespan (right skewed)

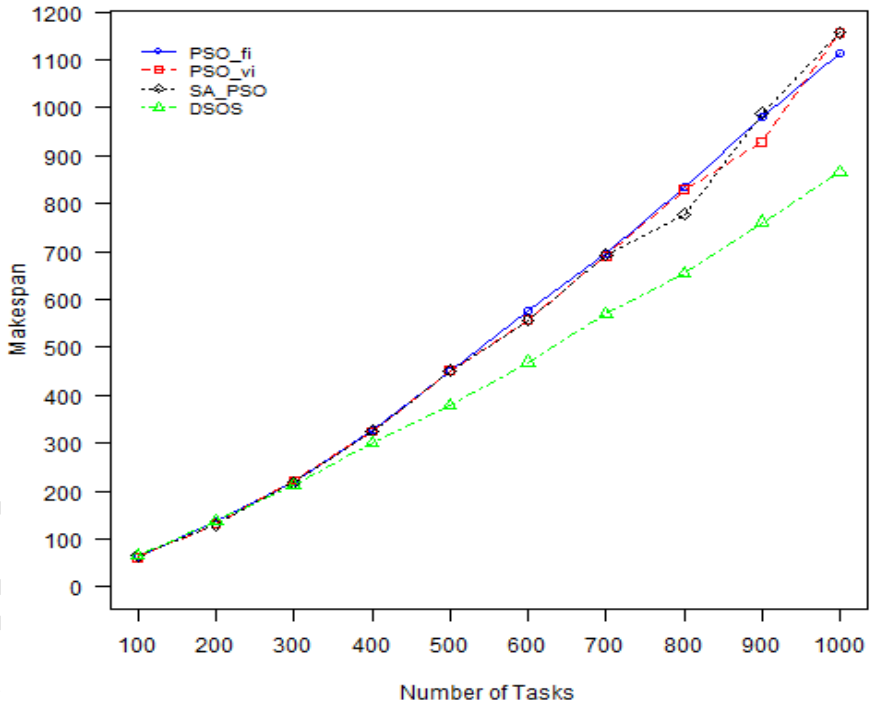

Fig. 5: Makespan (uniform dist.)

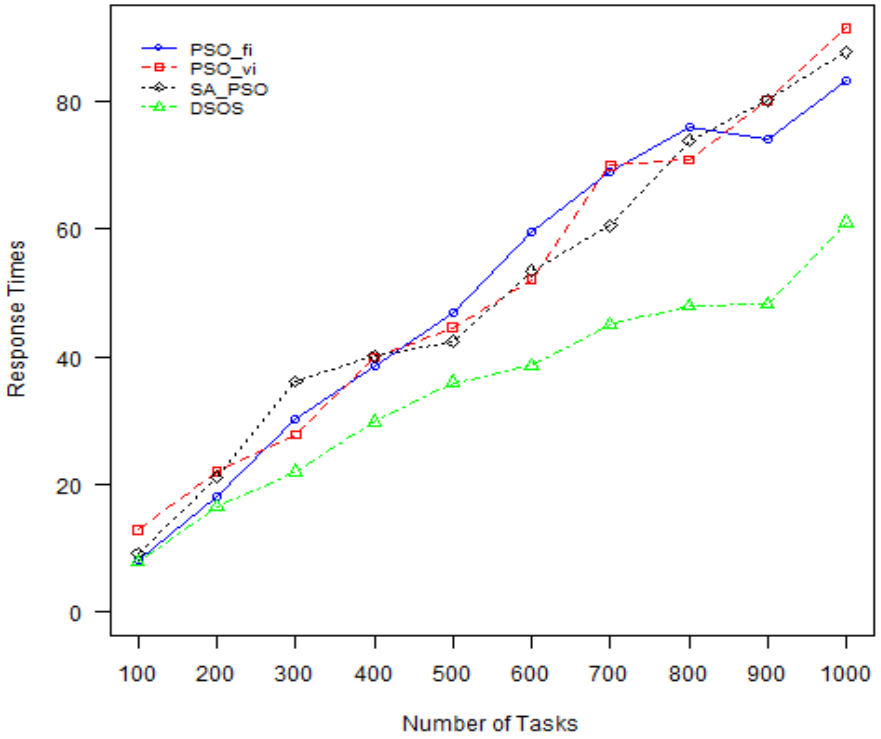

Fig. 6: Response time (normal dist.)

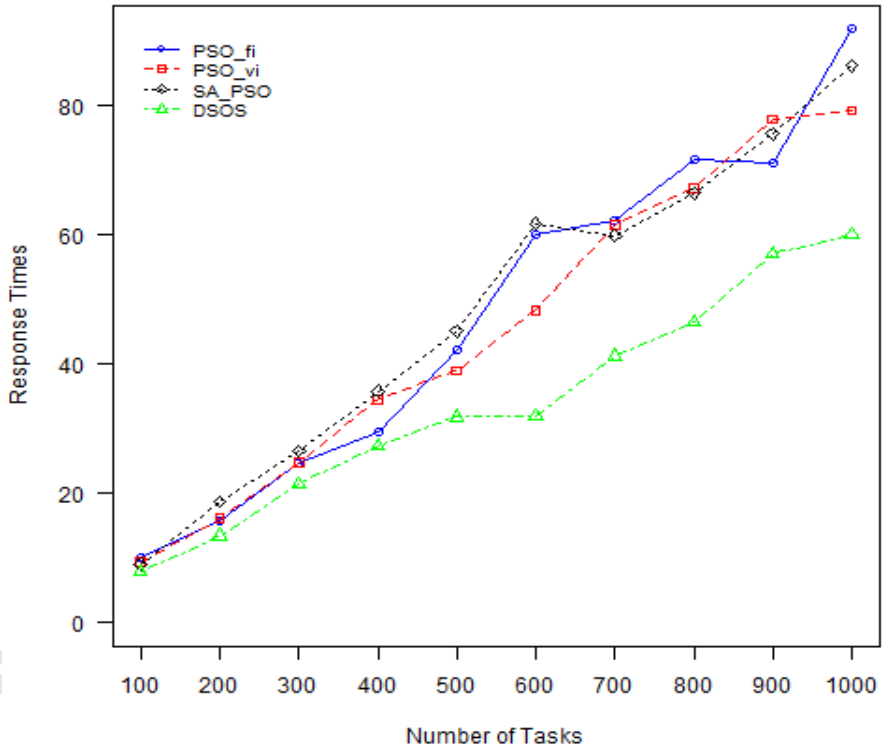

Fig. 7: Response time (left skewed)

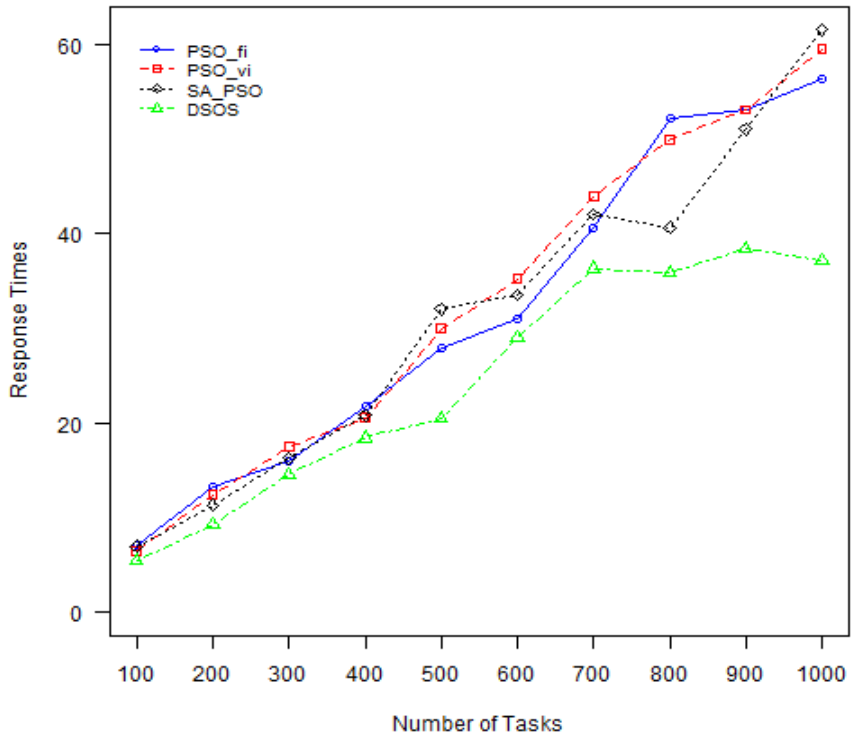

Fig. 8: Response time (right skewed)

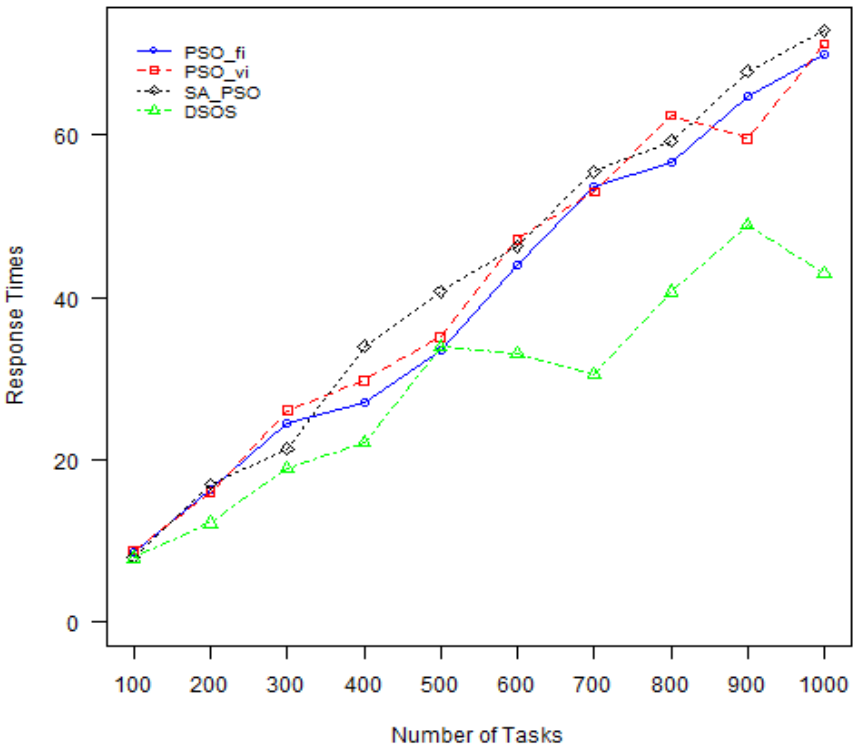

Fig. 9: Response time (uniform dist.)

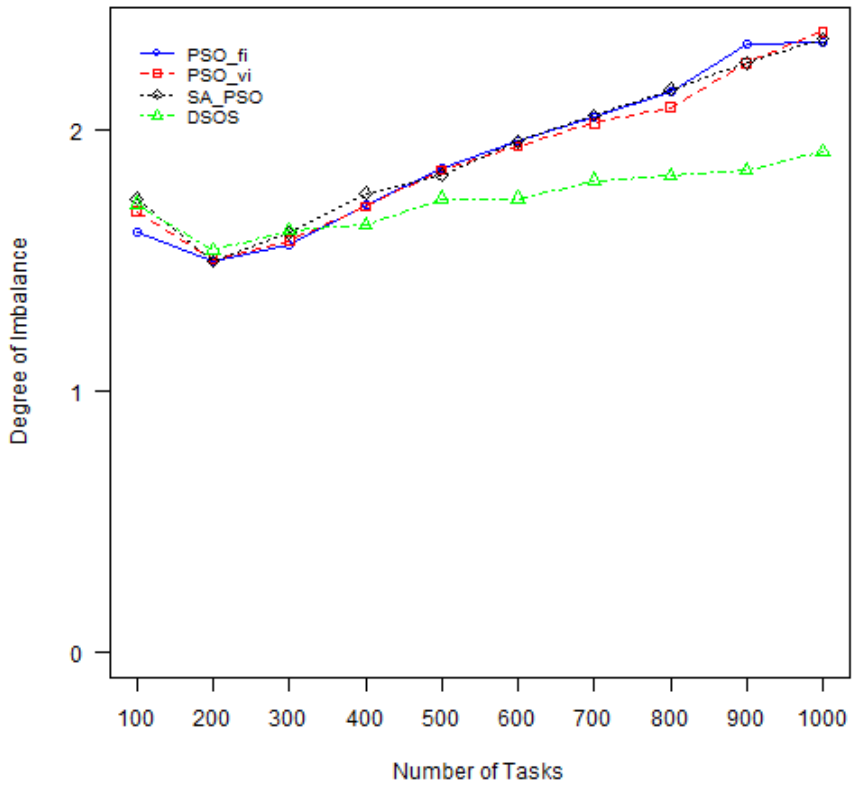

Fig. 10: Degree of imbalance (normal dist.)

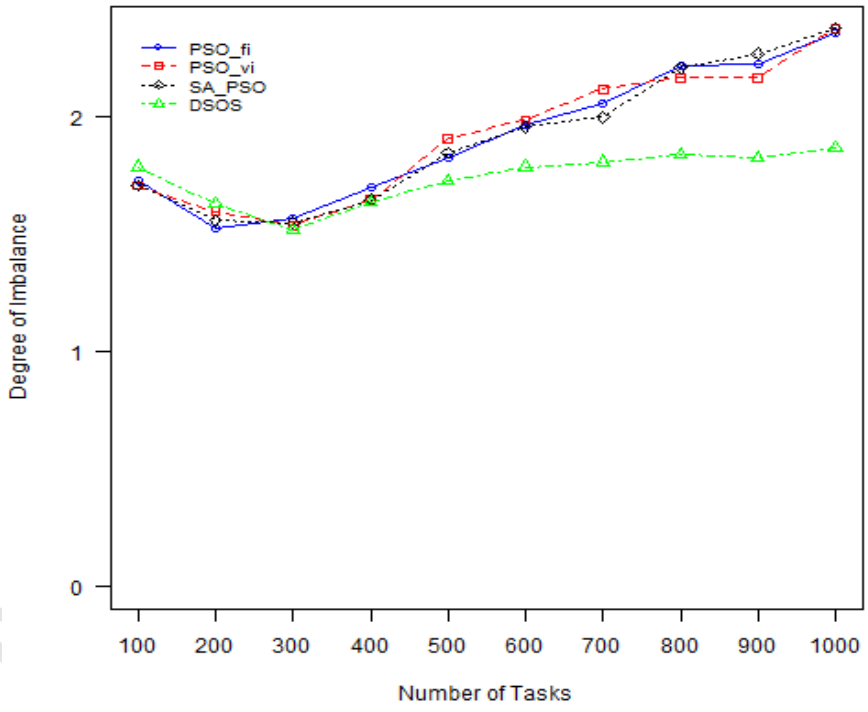

Fig. 11: Degree of imbalance (left skewed)

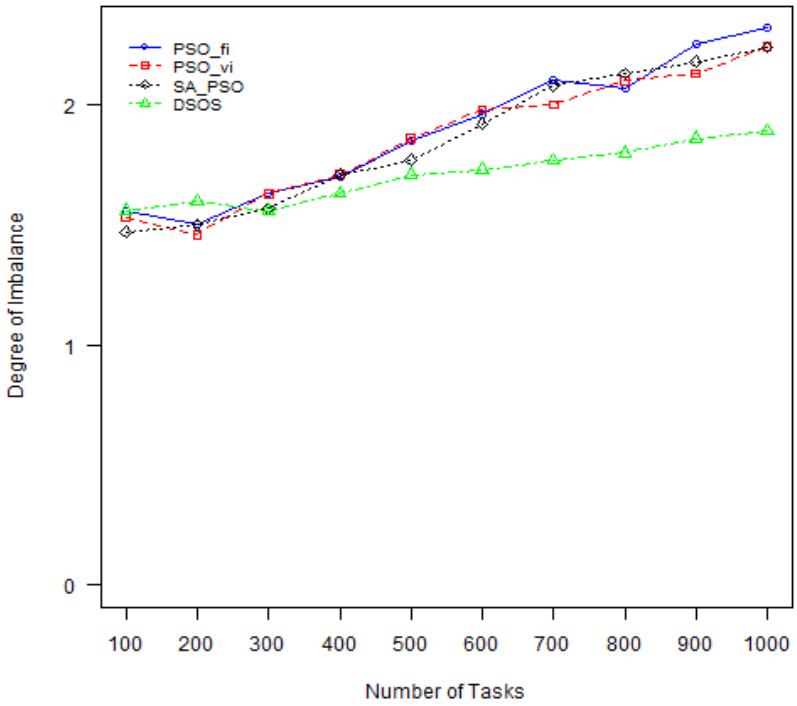

Fig. 12: Degree of imbalance (right skewed)

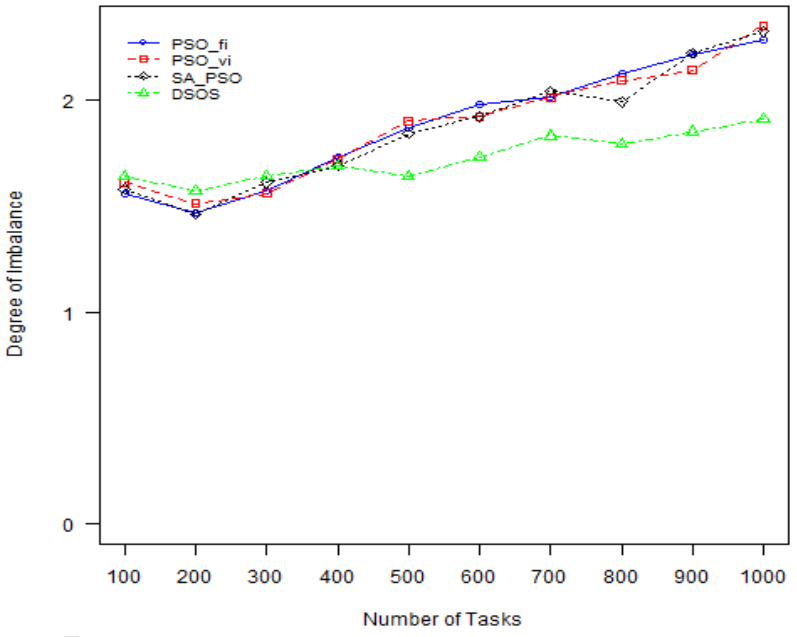

Fig. 13: Degree of imbalance (uniform dist.)

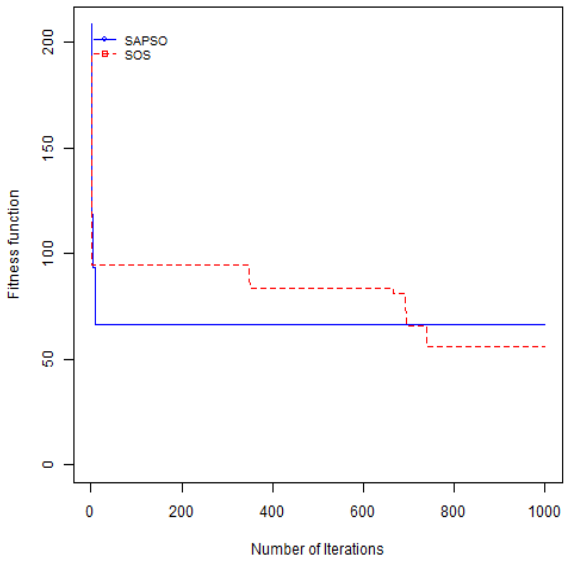

Fig. 14: Convergence graph (100 tasks)

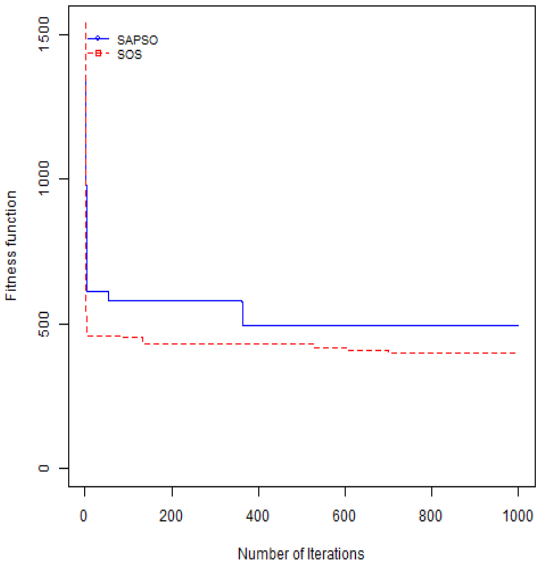

Fig. 15: Convergence graph (500 tasks)

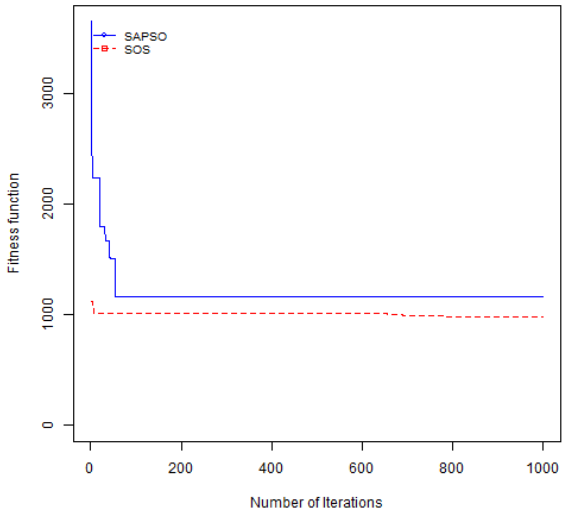

Fig. 16: Convergence graph (1000 tasks)

Table 4
Comparison of makespan obtained by SAPSO and DSOS for data instances generated using normal distribution.

Task size	SAPSO			DSOS			Improvement			t-test
	Best	Worst	Avg	Best	Worst	Avg	Best (%)	Worst (%)	Avg (%)	Calculate t-value
100	56.92	75.59	67.37	58.52	86.30	68.87	-2.81	-14.17	-2.23	-0.479
200	120.35	160.66	142.35	126.64	151.52	139.41	-5.23	5.69	2.07	0.538
300	217.29	269.49	241.31	218.10	250.17	233.86	-0.37	7.17	3.09	1.114
400	328.34	400.17	359.87	313.28	331.90	320.65	4.59	17.06	10.90	4.909
500	415.05	501.15	456.53	391.55	434.03	416.92	5.66	13.39	8.68	3.658
600	574.68	646.56	607.96	468.22	537.55	510.02	18.53	16.86	16.11	8.846
700	674.65	874.20	743.28	576.83	651.63	612.30	14.50	25.46	17.62	5.617
800	860.65	963.29	908.47	654.91	763.11	708.04	23.91	20.78	22.06	14.039
900	1007.28	1225.58	1074.82	751.48	868.11	809.73	25.40	29.17	24.66	10.716
1000	1140.45	1357.25	1242.98	863.72	1015.56	934.05	24.26	25.18	24.85	11.91

Table 5
Comparison of makespan obtained by SAPSO and DSOS for data instances generated using normal distribution.

Task size	SAPSO			DSOS			Improvement			t-test
	Best	Worst	Avg	Best	Worst	Avg	Best (%)	Worst (%)	Avg (%)	Calculate t-value
100	49.50	88.39	62.14	50.55	73.14	61.63	-2.12	17.25	0.83	0.127
200	124.13	160.14	144.91	128.95	157.31	146.71	-3.88	1.77	-1.24	-0.31
300	189.49	220.06	202.70	179.50	208.10	191.17	5.27	5.43	5.69	2.532
400	274.10	321.22	296.72	234.99	284.77	268.45	14.27	11.35	9.53	3.943
500	407.25	469.88	442.37	355.38	450.46	390.93	12.74	4.13	11.63	4.653
600	554.92	646.33	585.76	447.64	452.37	479.36	19.33	30.01	18.16	8.226
700	621.63	678.24	653.88	519.87	579.81	554.16	16.37	14.51	15.25	12.388
800	767.26	987.08	851.99	692.59	621.44	656.93	9.73	37.04	22.90	9.399
900	862.02	1016.98	952.30	665.89	752.85	716.98	22.75	25.97	24.71	13.442
1000	1127.74	1245.06	1185.46	769.52	888.99	836.29	31.76	28.60	29.45	18.998

Table 6
Comparison of makespan obtained by SAPSO and DSOS for data instances generated using normal distribution.

Task size	SAPSO			DSOS			Improvement			t-test
	Best	Worst	Avg	Best	Worst	Avg	Best (%)	Worst (%)	Avg (%)	calculate t-value
100	42.85	56.71	50.74	51.66	60.21	55.81	-20.56	-6.17	-9.99	-3.106
200	85.64	118.67	106.82	102.07	122.75	109.01	-19.18	-3.44	-2.05	-0.574
300	154.58	205.79	175.48	156.74	180.00	168.46	-1.40	12.53	4.00	1.176
400	235.96	283.93	253.93	209.56	243.20	232.03	11.19	14.35	8.62	3.635
500	300.42	384.22	338.74	286.86	334.59	306.20	4.51	12.92	9.61	3.372
600	413.89	482.06	450.72	367.46	407.87	388.89	11.22	15.39	13.72	6.509
700	510.38	638.15	570.55	427.65	488.48	461.83	16.21	23.45	19.06	7.895
800	583.46	693.98	654.16	492.08	577.81	523.90	15.66	16.74	19.91	8.85
900	642.47	827.08	755.51	554.29	655.03	614.74	13.73	20.80	18.63	7.445
1000	806.24	940.52	883.69	646.60	733.27	684.83	19.80	22.04	22.50	11.369

Table 7
Comparison of makespan obtained by SAPSO and DSOS for data instances generated using normal distribution.

Task size	SAPSO			DSOS			Improvement			t-test
	Best	Worst	Avg	Best	Worst	Avg	Best (%)	Worst (%)	Avg (%)	Calculate t-value
100	66.66	72.15	64.45	59.80	73.00	64.83	10.29	-1.18	-0.59	-0.156
200	111.56	145.93	129.96	124.64	150.32	138.55	-11.72	-3.01	-6.61	-1.862
300	196.92	243.14	217.70	185.75	229.66	212.40	5.67	5.54	2.44	0.83
400	289.48	353.06	326.02	292.16	307.18	301.28	-0.93	12.99	7.59	3.464
500	426.06	507.54	451.06	353.41	408.19	378.59	17.05	19.57	16.07	8.006
600	502.74	617.33	556.55	440.88	509.89	468.49	12.30	17.40	15.82	6.16
700	623.93	782.27	693.54	510.26	628.74	570.48	18.22	19.63	17.74	5.707
800	648.59	791.52	778.22	606.72	702.22	654.08	6.46	11.28	15.95	3.982
900	893.18	1044.98	988.48	686.35	801.27	760.97	23.16	23.32	23.02	11.641
1000	1062.00	1259.73	1156.24	819.43	907.57	864.78	22.84	27.96	25.21	13.337

Figure 2 to Figure 5 shows the average makespan for executing task instances 10 times, using PSO with fixed inertia, PSO with variable inertia, SAPSO, and DSOS. The figures indicated a minimization of makespan using DSOS in most scenarios, particularly from task instances of 300 upward. The percentage improvement of DSOS over SAPSO for different distribution of data instances are summarized in Table 4 to Table 7, showing that the degree of performance of DSOS over SAPSO increases as search space increases. Figures 6 to Figure 9 showed the response times obtained by PSO versions and DSOS,

with the figures showcasing that DSOS has minimal response time for different distributions of tasks sizes. DSOS also gives a better degree of imbalance among VMs for large problem instances as can be observed in Figure 10 to Figure 13.

The convergence graphs showing improvements in the quality of solutions for makespan obtained by SAPSO and DSOS using data instances 100, 500, and 1000 are presented in Figure 14 to Figure 16. As can be observed, both methods showed improvement in quality of solution at the beginning of the search but DSOS demonstrated the ability of improving its quality of solution at a later stage of the search process. The quality of solutions obtained by DSOS is better than that of SAPSO especially when the problem size is larger.

One sided t test was carried out to examine whether the makespan obtained by DSOS is significantly less than that of SAPSO for all task instances using the same stopping criteria. The alternative hypothesis (H_a) was set as the statement of the test while the null hypothesis (H_0) was set as the complementary statement. Acceptable Type I and Type II errors of 1% were used to conduct a test with critical value obtained from statistical table as 2.554. The statistical analysis of performance of DSOS and SAPSO under different data instances are presented in Tables 2 to Table 5 which indicate that for data instances 400 through 1000 the calculated t-value is greater than 2.554(critical t-value) which means that there is a significant difference between the performance of DSOS and SAPSO for these data instances. This leads to acceptance of H_a for data instances 400 through 1000. For the data instances 100 through 300, calculated t-value is less than critical t-value which indicates that the difference between the performances is insignificant. Therefore, H_0 is accepted for data instances 100 through 300. This means that for large task instances (400 and above) makespan obtained by SOS is significantly less than that of SAPSO using the same stopping criteria at 99% confidence level. It can be concluded that DSOS outperforms SAPSO when the search space is larger.

The performance improvement by DSOS is believed to be attributed to mutual benefit and parasite vector mechanisms which are unique to SOS. The mutual benefit factor mechanism in mutualism phase gives the search process exploitative power by enabling it to traverse the best solution regions. The parasite vector technique in parasitism phase is capable of preventing premature convergence by eliminating inactive solutions and introducing a more active solution which pushes away search processes from local optima. The parasitism phase empowers search process with the explorative ability by not concentrating only on the best solution regions which could likely trap the search in a certain search region as demonstrated in Figure 14 to Figure 16. The method is able to improve its quality even at a later stage of search process which means that DSOS has a higher probability of obtaining near-optimal solution than SAPSO.

6. Summary and Conclusion

In this paper, we designed a discrete version of Symbiotic Organism Search metaheuristic algorithm. The continuous version of the algorithm was inspired by symbiotic relationships exhibited by organisms in a habitat. The algorithm mimics the mutualism, commensalism and parasitic relationship to improve the quality of a given

objective function. The method was implemented to schedule independent tasks using the CloudSim tool kit. Makespan, response time, and degree of imbalance among VMs were measured and DSOS was found to be better than SAPSO. The performance of DSOS over SAPSO increases with increase in search space. The average makespan minimization by DSOS was 3.8% to 25.5% less than that of SAPSO for 300 through 1000 instances of tasks respectively. The results of statistical test conducted revealed that DSOS outperforms SAPSO especially when the search space is larger. The mutual benefit factor mechanism in mutualism phase enables DSOS to explore new regions of the search for better solutions. The parasite vector technique in parasitism phase is capable of preventing premature convergence by adding perturbation to the ecosystem. The benefit factor and parasite vector mechanism are unique to SOS which are considered as its advantage. The mechanisms play a vital role in the exploration and exploitation in the search process. SOS has fewer parameters and is easier to implement which is considered an advantage in addition to the explorative and exploitative ability.

Application of DSOS to other discrete optimization problems is a potential future research. Also, multi-objective version of SOS can be designed for the cloud environment, taking into consideration other factors when scheduling tasks. Study of workflow scheduling using SOS is another future investigation.

Acknowledgments

This work is supported by UTM/RUG/04H11 RMC Universiti Teknologi Malaysia and PRGS/1/2014/ICT03/UTM/02/1 and FRGS/1/2014/ICT03/UTM/02/1 by Ministry of Higher Education (MOHE). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

7. References

1. Durao, F., et al., *A systematic review on cloud computing.* The Journal of Supercomputing, 2014. **68**: p. 1321-1346.
2. Avram, M.G., *Advantages and Challenges of Adopting Cloud Computing from an Enterprise Perspective.* Procedia Technology, 2014. **12**: p. 529-534.
3. Assunção, M.D., A. Costanzo, and R. Buyya, *A cost-benefit analysis of using cloud computing to extend the capacity of clusters.* Cluster Computing, 2010. **13**(3): p. 335-347.
4. Amazon EC2. Available from: https://aws.amazon.com/ec2/.
5. Tsai, C.-W., et al., *A Hyper-Heuristic Scheduling Algorithm for Cloud.* IEEE Transactions on Cloud Computing, 2014. **2**: p. 236-250.
6. Aceto, G., et al., *Cloud monitoring: A survey.* Computer Networks, 2013. **57**(9): p. 2093-2115.
7. M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*. 1979, New York: W.H. Freeman and Company.
8. Gao Ming and H. Li, *An Improved Algorithm Based on Max-Min for Cloud Task Scheduling.* Recent Advances in CSIE 2011, LNEE, 2012. **125**: p. 217–223.
9. Upendra Bhoi and P.N. Ramanuj, *Enhanced Max-min Task Scheduling Algorithm in Cloud Computing.* International Journal of Application or Innovation in Engineering and Management, 2013. **2**(4): p. 259-264.
10. Ehsan ullah Munir, Jian Zhong li, and S. Shi, *QOS sufferage Heuristic for Independent Task Scheduling in Grid.* Journal of Information Technology, 2007. **6**(8): p. 1166-1170.
11. Wu, L., Y.J. Wang, and C.K. Yan, *Performance comparison of energy-aware task scheduling with GA and CRO algorithms in cloud environment*, in *Applied Mechanics and Materials*. 2014. p. 204-208.
12. Zhao, C., et al., *Independent Tasks Scheduling Based on Genetic Algorithm in Cloud Computing*. 2009 5th International Conference on Wireless Communications, Networking and Mobile Computing, Vols 1-8. 2009. 5548-5551.
13. Fei Taoa and L.Z. Ying Fengb, T.W. Liaoc, *CLPS-GA: A case library and Pareto solution-based hybrid genetic algorithm for energy-aware cloud service scheduling.* Applied Soft Computing, 2014. **19**: p. 264–279.
14. Zhu, Y. and P. Liu, *Multi-dimensional constrained cloud computing task scheduling mechanism based on genetic algorithm.* International Journal of Online Engineering, 2013. **9**(SPECIALISSUE.6): p. 15-18.
15. Zheng, S.M., et al., *Grid task scheduling genetic algorithm based on cloud model.* Dianzi Keji Daxue Xuebao/Journal of the University of Electronic Science and Technology of China, 2012. **41**(6): p. 911-915.
16. Wang, M., W. Zeng, and Ieee, *A comparison of four popular heuristics for task scheduling problem in computational grid*. The 6th International Conference on Wireless Communications Networking and Mobile Computing. 2010.

17. Liu, Z. and X. Wang, *A PSO-Based Algorithm for Load Balancing in Virtual Machines of Cloud Computing Environment*, in *Advances in Swarm Intelligence, Icsi 2012, Pt I*, Y. Tan, Y. Shi, and Z. Ji, Editors. 2012. p. 142-147.
18. Wang, J., F. Li, and L.Q. Zhang, *Apply PSO into cloud storage task scheduling with QoS preference awareness.* Tongxin Xuebao/Journal on Communications, 2014. **35**(3): p. 231-238.
19. Guo, L.Z., et al., *Particle swarm optimization embedded in variable neighborhood search for task scheduling in cloud computing.* Journal of Donghua University (English Edition), 2013. **30**(2): p. 145-152.
20. Ramezani, F., J. Lu, and F. Hussain, *Task scheduling optimization in cloud computing applying multi-objective particle swarm optimization*, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2013. p. 237-251.
21. Ramezani, F., J. Lu, and F.K. Hussain, *Task-based system load balancing in cloud computing using particle swarm optimization.* International Journal of Parallel Programming, 2014. **42**(5): p. 739-754.
22. Popov, V., *Particle swarm optimization technique for task-resource scheduling for robotic clouds*, *Applied Mechanics and Materials*. 2014. p. 243-246.
23. Netjinda, N., B. Sirinaovakul, and T. Achalakul, *Cost optimal scheduling in IaaS for dependent workload with particle swarm optimization.* Journal of Supercomputing, 2014. **68**(3): p. 1579-1603.
24. Netjinda, N., B. Sirinaovakul, and T. Achalakul. *Cost optimization in cloud provisioning using particle swarm optimization*. The *9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, ECTI-CON 2012*. 2012.
25. Chitra, S., et al., *Local minima jump PSO for workflow scheduling in cloud computing environments*, *Lecture Notes in Electrical Engineering*. 2014. p. 1225-1234.
26. Bilgaiyan, S., S. Sagnika, and M. Das. *Workflow scheduling in cloud computing environment using Cat Swarm Optimization*. *Souvenir of the 2014 IEEE International Advance Computing Conference, IACC 2014*. 2014.
27. Xue, S., et al., *An ACO-LB algorithm for task scheduling in the cloud environment.* Journal of Software, 2014. **9**(2): p. 466-473.
28. Xue, S., J. Zhang, and X. Xu, *An improved algorithm based on ACO for cloud service PDTs scheduling.* Advances in Information Sciences and Service Sciences, 2012. **4**(18): p. 340-348.
29. Tong, Z., et al., *H2ACO: An optimization approach to scheduling tasks with availability constraint in heterogeneous systems.* Journal of Internet Technology, 2014. **15**(1): p. 115-124.
30. Sun, W., et al. *PACO: A period ACO based scheduling algorithm in cloud computing*. *Proceedings of International Conference on Cloud Computing and Big Data, CLOUDCOM-ASIA 2013*. 2013.
31. Hongbo Liua, Ajith Abraham, and A.E. Hassaniend, *Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm.* Future Generation Computer Systems, 2010. **26**: p. 1336–1343.

32. Shaminder Kaur and A. Verma, *An Efficient Approach to Genetic Algorithm for Task Scheduling in Cloud Computing Environment.* International Journal Information Technology and Computer Science, 2012. **10**: p. 74-79.
33. Yang, Z., et al., *Optimized task scheduling and resource allocation in cloud computing using PSO based fitness function.* Information Technology Journal, 2013. **12**(23): p. 7090-7095.
34. Zhan, S. and H. Huo, *Improved PSO-based task scheduling algorithm in cloud computing.* Journal of Information and Computational Science, 2012. **9**(13): p. 3821-3829.
35. Zhong, S. and Z. He, *The Scheduling Algorithm of Grid Task Based on PSO and Cloud Model*, *Advanced Measurement and Test, Parts 1 and 2*, Y.W. Wu, Editor. 2010. p. 1487-1492.
36. Zuo, X., G. Zhang, and W. Tan, *Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud.* IEEE Transactions on Automation Science and Engineering, 2014. **11**(2): p. 564-573.
37. Ge, Y. and G. Wei. *GA-based task scheduler for the cloud computing systems*. *Proceedings of International Conference on Web Information Systems and Mining, WISM 2010*. 2010.
38. Pandey, S., et al., *A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments.* The 24th IEEE International Conference on Advanced Information Networking and Applications (Aina), 2011: p. 400-407.
39. Suraj Pandey, et al., *A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments*, The *24th IEEE International Conference on Advanced Information Networking and Applications (AINA),* 2010: Perth, WA.
40. Verma, A. and S. Kaushal. *Bi-Criteria Priority based Particle Swarm Optimization workflow scheduling algorithm for cloud*. *Recent Advances in Engineering and Computational Sciences, RAECS 2014*. 2014.
41. Wang, J., F. Li, and A. Chen, *An improved PSO based task scheduling algorithm for cloud storage system.* Advances in Information Sciences and Service Sciences, 2012. **4**(18): p. 465-471.
42. Wang, J., F. Li, and L. Zhang, *QoS preferenceawareness task scheduling based on PSO and AHP methods.* International Journal of Control and Automation, 2014. **7**(4): p. 137-152.
43. Min-Yuan Cheng and D. Prayogo, *Symbiotic Organisms Search: A new metaheuristic optimization algorithm.* Computers and Structures, 2014. **139**: p. 98-112.
44. Khorram, E. and H. Zarei, *Multi-objective optimization problems with Fuzzy relation equation constraints regarding max-average composition.* Mathematical and Computer Modelling, 2009. **49**(5–6): p. 856-867.
45. Loo, S.M. and B.E. Wells, *Task scheduling in a finite-resource, reconfigurable hard ware/software codesign environment.* INFORMS Journal on Computing, 2006. **18**(2): p. 151-172.
46. Demiroz, B. and H.R. Topcuoglu, *Static task scheduling with a unified objective on time and resource domains.* Computer Journal, 2006. **49**(6): p. 731-743.

47. Calheiros, R.N., et al., *CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms.* Software - Practice and Experience, 2011. **41**(1): p. 23-50.
48. Eberhart, R.C. and Y. Shi. *Comparing inertia weights and constriction factors in particle swarm optimization*. *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC*. 2000.

Highlights

- Discrete Symbiotic Organism Search algorithm for task scheduling is proposed.
- The proposed has better ability to exploit best solution regions than PSO.
- The proposed method has global ability in terms of exploring optimal solution points.
- The proposed algorithm performs significantly better than PSO for large search spaces.

Abdullahi Mohammed is currently pursuing his PhD in Computer Science at Universiti Teknologi Malaysia. He obtained his B.Tech in Mathematics with Computer in Federal University of Technology, Minna Nigeria. He is a lecturer in Ahmadu Bello University, Zaria Nigeria where he received his MSc in Computer Science. His research interests include algorithm design for distributed systems, cloud computing and energy efficient computation. He is a member of IEEE and ACM.

Shafi'i Muhammad ABDULHAMID received his M.Sc. degree in Computer Science from Bayero University Kano, Nigeria and B.Tech. degree in Mathematics/Computer Science from the Federal University of Technology Minna, Nigeria. His current research interest is scheduling in Grid and Cloud Computing. He is a member of IEEE and a member of Nigerian Computer Society (NCS). Presently he is a PhD candidate at the Faculty of Computing, Universiti Teknologi Malaysia.

Md Asri Bin Ngadi received PhD in Computer Science from Aston University, Birmingham UK and B.Sc in Computer Science from Universiti Teknologi Malaysia. His research interests is related to Wireless Computer, Cloud Computing, and Network Security. Currently he is appointed as Associate Professor at Universiti Teknologi Malaysia and a member of IEEE and ACM.