Glowworm swarm optimisation based task scheduling for cloud computing

Alboaneen, Dabiah Ahmed; Tianfield, Huaglory; Zhang, Yan

Published in:
Proceedings of the 2nd International Conference on Internet of things and Cloud Computing

DOI:
10.1145/3018896.3036395

Publication date:
2017

Document Version
Peer reviewed version

Glowworm Swarm Optimisation Based Task Scheduling for Cloud Computing

Dabiah Ahmed Alboaneen
Department of Computer, Communications and Interactive Systems
Glasgow Caledonian University
Glasgow, UK
dabiah.alboaneen@gcu.ac.uk

Huaglory Tianfield
Department of Computer, Communications and Interactive Systems
Glasgow Caledonian University
Glasgow, UK
h.tianfield@gcu.ac.uk

Yan Zhang
Department of Computer, Communications and Interactive Systems
Glasgow Caledonian University
Glasgow, UK
yan.zhang@gcu.ac.uk

ABSTRACT

Task scheduling is a non-deterministic polynomial-time hard (NP-hard) optimisation problem, thus applying metaheuristics is important. In this paper, we employ glowworm swarm optimisation (GSO) to solve the task scheduling problem in cloud computing to minimise the total execution cost of tasks while keeping the total completion time within the deadline. Simulation results show that GSO based task scheduling (GSOTS) algorithm outperforms shortest task first (STF), largest task first (LTF) and particle swarm optimisation (PSO) algorithms in reducing the total completion time and the cost of executing tasks.

CCS Concepts

•Computer systems organization → Cloud computing;

Keywords

Cloud computing; resource management; glowworm swarm optimisation (GSO); task scheduling

1. INTRODUCTION

Task scheduling is crucial in cloud computing, which is a process that maps users' tasks to suitable resources in the form of virtual machines (VMs) to execute.

There are two types of task scheduling in cloud computing, i.e., (i) Static task scheduling where all tasks arrive at the same time and they are known a priori to scheduling. In this case, tasks are assigned to VMs in a static way. (ii) Dynamic task scheduling where all the tasks are scheduled instantly once they arrive.

When users' tasks need to be scheduled, users usually sign a service level agreement (SLA) with cloud providers. In this SLA, the quality of service (QoS) requirements of the task scheduling should be clearly defined such as, the deadline of each task, task scheduling budget and service security. Each cloud user has to decide which and how many VMs need to be provisioned before actually requesting and paying for them. Moreover, as cloud resources are pay-per-use, the cost of task execution has to be taken into account. So, task scheduling directly affects the performance of cloud computing.

Task scheduling in cloud computing can be modelled as a bin-packing problem and is a non-deterministic polynomial-time hard (NP-hard) problem [1]. This problem becomes more challenging with the increase of the complexity of cloud computing. Generally, it is difficult to develop algorithms for producing optimal solutions within a short time. Recently, using metaheuristic algorithms such as genetic algorithm (GA) and particle swarm optimisation (PSO) to deal with cloud resource scheduling has received increasing attention due to the ability of the algorithms to provide near-optimal solutions within a reasonable time [2].

To address the aforementioned problem, this paper proposes a solution by employing glowworm swarm optimisation (GSO) for improving the execution cost of scheduled independent tasks. The execution cost of a task considers completion time and the price of using the VM. Completion time includes waiting time and execution time of the task and it should be within the user defined deadline to meet the user requirements. To evaluate the performance of the proposed GSO based task scheduling (GSOTS) algorithm, we undertake a comparative simulation study of the proposed algorithm with shortest task first (STF), largest task first (LTF) and PSO algorithms.

The remainder of this paper is arranged as follows. Section 2 gives the task scheduling problem formulation. The GSOTS algorithm is put forward in Section 4. Simulations setup and the results are presented in Section 5. Section 6 discusses the work related to the scheduling algorithms in a cloud environment. Finally, Section 7 draws the conclusion and points out the future work.

2. PROBLEM FORMULATION

The general system is depicted in Figure 1. Assume that a cloud platform is supported by physical machines (PMs) $PM_1, .., PM_P$, each of which hosts a set of VMs via the

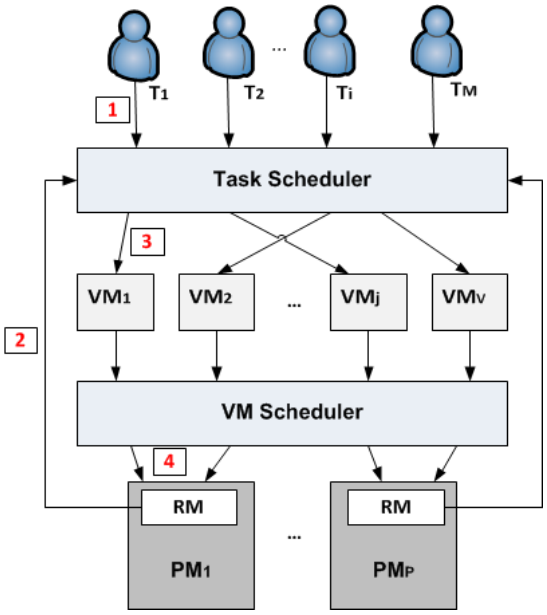


Figure 1. Task and VM scheduling in cloud computing

corresponding virtual machine monitor (VMM) and each PM runs resource manager (RM) which is responsible for monitoring the resource utilisation of PMs and collecting run-time statistics of each PM, including PM availability, resource utilisation and VM status.

There are M independent tasks to be scheduled. User requests (tasks) are submitted to the computing system (1). The task scheduler collects the feedback information from the RMs to maintain an overall view of the system resource utilisation as marked by information (2). Based on the information of user requests received from the service provider and the feedback received from RMs, the task scheduler initiates the task scheduling algorithm to schedule tasks on VMs (3) and VM scheduler aims to allocate VMs on PMs as marked by information (4).

In our task scheduling algorithm, the task scheduler first calculates the completion time to execute the task which is based on the execution time and waiting time of task in VM queuing. The execution time is calculated based on the number of instructions in each task which is received from the user side and the VM capacity which is received from the RM side. If the completion time of executing task is within the user defined deadline, then the task can be executed on the VM. After that, the task scheduler calculates the execution cost of task in each available VM. To minimise the execution cost of the task, task scheduler decides which VM is suitable to be provisioned to each task to meet the requirements of each task. Finally, the task scheduler will return the result of execution to user when all tasks are done.

Assumptions

(i) Each task is allowed to be processed on any given available VM that meets the constraints.

(ii) The execution time of each task is VM-dependent.

(iii) Each task must be completed without interruption once started (non-preemptable).

(iv) Each VM can be provisioned to more than one task.

Table 1. Amazon EC2 instance types and prices

VM type	MIPS	Pe	Price ($/Hour)
Type1	500	4	$0.34
Type2	1000	7	$0.5
Type3	1500	20	$0.6

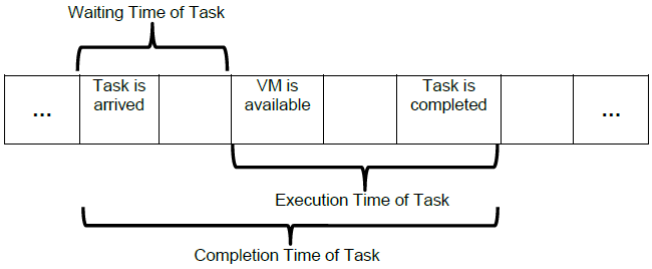


Figure 2. Task scheduling lifetime

Inputs:

- Set of tasks is defined as $T = \{T_1, T_2, ..., T_i, , ..., T_M\}$, where $i \in [1, M]$ and M is the total number of tasks. Each task i is described as $T_i(inst, deadline, arv)$, where $inst$, $deadline$ and arv represent the number of task instructions, task deadline and task arrival time, respectively.

- Set of VMs is defined as $VM = \{VM_1, .., VM_j, .., VM_V\}$, where $j \in [1, V]$ and V is the total number of VMs. Each VM_j is described as $VM_j(C, Price)$, where C is the VM capacity which is the processing speed and expressed in terms of million instructions per second (MIPS). $Price$ is the amount spent for using a VM in an hour.

Output: Selected VM for each task.

Objective: The aim of the GSOTS algorithm is to schedule tasks to appropriate VMs in order to minimise the execution cost (EC) as below.

$$f = minimise[EC_1, EC_2, ..., EC_i, EC_M] \quad (1)$$

Execution Cost: The execution cost (EC) of task i is defined as multiplication of the price of VM_j and the completion time of task i, that is,

$$EC_i = Price_j * CT_{ij} \quad (2)$$

where $Price_j$ is the price of VM_j (Table 1 shows Amazon EC2 pricing model[1]) and CT_{ij} is the completion time of executing task i on VM_j.

If the completion time CT_{ij} is within the user defined deadline (in our case is 1800 sec), task i can be executed. Otherwise, it cannot be executed.

Completion Time: As seen in Figure 2, the execution time is the time that VM takes to execute the task. Waiting time is the time difference between task arrival time and task execution time. Hence, the completion time that VM_j will take to execute task i can be calculated as below.

$$CT_{ij} = arv_i + ET_{ij} \quad (3)$$

[1]http://aws.amazon.com/ec2/instance-types/

where arv_i is the arrival time of task i and ET_{ij} is the execution time of task i. The execution time ET_{ij} is calculated as below.

$$ET_{ij} = \frac{inst_i}{C_j} \tag{4}$$

where $inst_i$ is the number of instructions that task i will need to execute on VM_j and C_j is the capacity of VM_j, which can be calculated below.

$$C_j = (Pe_j * MIPS_j) \tag{5}$$

where Pe_j is the number of processors in VM_j, $MIPS_j$ is million instructions per second of each processor in VM_j.

3. STANDARD GSO ALGORITHM

GSO algorithm is a swarm intelligence algorithm developed by Krishnan and Ghose. GSO is a population-based algorithm. As control is not centralised at a single point, it is more scalable [3]. In our earlier work [4], GSO algorithm has been applied for dynamic VMs allocation in cloud computing.

GSO is based on the behaviour of glowworms. A glowworm that produces more light (high luciferin) means that it is closer to an actual position and has a high objective function value. Each glowworm selects a neighbour that has a higher luciferin value than its own, according to a probabilistic estimation, and moves towards it. These movements are based only on local information. Therefore, it enables the glowworms to divide into subgroups leading to the detection of multiple optima of the given objective function.

GSO algorithm starts by positioning glowworms randomly in the workspace and all the glowworms contain an equal quantity of luciferin. A GSO algorithm comprises four phases, i.e., initialisation, luciferin updating, moving and local radial range updating. The GSO algorithm is shown in Algorithm 1.

Algorithm 1 GSO

Initialise parameters β , p, s, n_t
$\forall j$, set $\ell_j(0) = \ell_0$
$\forall j$, set $\gamma_d^j(0) = \gamma_0$
while *termination_condition_not_meet* **do**
  for $j = 1$ **to** m **do**
    $\ell_j(t+1) = (1-p)\ell_j(t) + \gamma f(x_j(t+1))$
    $N_j(t) = \{n : ||x_n(t) - x_j(t)|| \leq \gamma_d^j(t); \ell_j(t) \leq \ell_n(t)\}$
    for each $n \in N_j(t)$ **do**
      $p_{jn}(t) = \frac{\ell_n(t) - \ell_j(t)}{\sum_{k \in N_j(t)} \ell_k(t) - \ell_j(t)}$
    end for
    $x_j(t+1) = x_j(t) + s\left(\frac{x_n(t) - x_j(t)}{||x_n(t) - x_j(t)||}\right)$
    $\gamma_d^j(t+1) = \min\{\gamma_s, \max\{0, \gamma_d^j(t) + \beta(n_t - |N_j(t)|)\}\}$
  end for
  $t \leftarrow t+1$
end while

Table 2. Notations

i	Index for tasks
M	Total number of tasks
j	Index for VMs
V	Total number of VMs
$\ell_j(t)$	Luciferin value of VM_j
$p_{jn}(t)$	Probability of task in VM_j to select VM_n
$\gamma_d^j(t)$	Local radial range of VM_j
γ_s	Max sensor range of $\gamma_d^j(t)$
$N_j(t)$	Neighbours set of VM_j
$\|N_j(t)\|$	Actual number of neighbours
n_t	Number of desired neighbours
$\|\|x\|\|$	Euclidean norm of x
β	Change rate of the neighbourhood range
s	Step size of moving
γ	Luciferin enhancement coefficient
p	Luciferin decay coefficient $(0 < p < 1)$

4. GSO BASED TASK SCHEDULING (GSOTS) ALGORITHM

GSO algorithm is applied to search for VM that minimises the execution cost. Each glowworm represents a VM and the luciferin of VM is the execution cost. According to the nature of glowworms, they always move towards their neighbours having higher luciferin than its own. But in our algorithm, a VM is attracted towards its neighbour which has lowest execution cost, which is reverse of the characteristics of the glowworm. We have the four phases of the process of GSOTS as follows. For convenience and clarity, notations are listed in Table 2.

4.1 Initialisation

The initial population consists of number of glowworms, which are considered as candidate solutions to the problem. The initial population is generated randomly. Each VM_j has a location $x_j(t)$, that is defined by the completion time that VM takes it to execute the task, a luciferin value $\ell_j(t)$, a local radial range $\gamma_d^j(t)$, max sensor range γ_s, the size of moving step s, the number of desired neighbours n_t, the luciferin decay coefficient p $(0 < p < 1)$, the luciferin enhancement coefficient γ and the change rate of the neighbourhood range β.

4.2 Luciferin Update

Initially, each VM_j converts the objective function value $f(x_j(t+1))$ to a luciferin value $\ell_j(t+1)$ by using the formula below.

$$\ell_j(t+1) = (1-p) * l_j(t) + \gamma f(x_j(t+1)) \tag{6}$$

where $\ell_j(t)$ is the luciferin value of VM_j at iteration t and $f(x_j(t+1))$ represents the value of the objective function at iteration $t+1$ as calculated below.

$$f(x_j(t+1)) = Price_j * CT_{ij} \tag{7}$$

Therefore, the luciferin values of VMs are updated according to the objective function values. A lower luciferin value means a better result as the goal is to minimise the execution cost. Also, the luciferin value is decreased along the time to simulate the decay.

4.3 Movement

In the movement phase, a task in VM_j chooses to move toward one of its neighbours n that has a lower luciferin value (lower EC) and within the local radial range γ_d which represents the task deadline. The movement phase consists

of four further steps.

Step 1. **Find Neighbours:** The set of VMs neighbours which meets the deadline of executing tasks can be written as below.

$$N_j(t) = \{n : ||x_n(t) - x_j(t)|| \leq \gamma_d^j(t); \ell_j(t) \geq \ell_n(t)\} \quad (8)$$

where n is the index of neighbour VM, $x_n(t)$ and $x_j(t)$ are completion time of VM_n and VM_j, respectively and $\ell_j(t)$ and $\ell_n(t)$ are luciferin values for VM_j and VM_n, respectively.

Step 2. **Calculate Probabilities:** For each VM_j, the probability to figure out the movement direction toward the neighbour with a lower luciferin value is calculated by using the formula below.

$$p_{jn}(t) = \frac{\ell_n(t) - \ell_j(t)}{\sum_{k \in N_j(t)} \{\ell_k(t) - \ell_j(t)\}} \quad (9)$$

where $p_{jn}(t)$ is the probability of task in VM_j to move to VM_n.

Step 3. **Selection:** The task located in VM_j selects a VM_n from the neighbour set that has the highest probability over others in the neighbour set.

Step 4. **Movement:** The new completion time of the VM after allocating the task is calculated using the formula below.

$$x_j(t+1) = x_j(t) + s\left(\frac{x_n(t) - x_j(t)}{||x_n(t) - x_j(t)||}\right) \quad (10)$$

where $x_j(t+1)$ and $x_j(t)$ are the new and current completion times of VM_j, respectively.

4.4 Local Radial Range Update

The local radial range γ_d^j is updated as below in order to formulate the neighbour set in the next task scheduling for adding more flexibility to the VM.

$$\gamma_d^j(t+1) = \min\{\gamma_s, \max\{0, \gamma_d^j(t) + \beta(n_t - |N_j(t)|)\}\} \quad (11)$$

The pseudocode of the GSOTS algorithm is shown in Algorithm 2. The input parameters of the GSOTS algorithm include VMs list and tasks details. The main steps of GSOTS are as follows.

GSO parameters and VMs are initialised (lines 3-6). The algorithm finds a VM in the list which can complete the execution of the task in the lowest cost if a task were assigned to it. This ensures that the overall execution cost is as low as possible. For each task i in the *taskList* that needs to be scheduled (line 7), the algorithm will search for a suitable VM_j from the *vmList* (line 10), the luciferin of VM_j will be updated (line 11). The neighbour set will be calculated (lines 12-18), if any of the neighbouring VM has less execution cost than the original VM and is within the local radial range γ_d, then this VM will be added to the neighbour set n_t. The size of the neighbour set n_t is predefined by the user.

The *selectedVm* is one of the neighbouring VMs which has the highest probability to move task i to it (lines 19-21). After that, the radial range which defines the neighbour set will be updated (line 22). Finally, the algorithm will be terminated when suitable VM for scheduling the task is found (line 29).

Algorithm 2 GSOTS

```
1: Input: vmList, taskList
2: Output: selectedVm
3: Initialise parameters β , p, s, n_t, γ_d^j
4: Set t = 1
5: Initialise tasks on VMs randomly
6: g = Glowworm(Vm)
7: for i ∈ taskList do
8:    selectedVm ← NULL
9:    while termination_condition_not_meet do
10:      for j ∈ vmList do
11:         ℓ_j(t) = g.updateLuc(Vm) by Eq.(6)
12:         for n = 1 to n_t do
13:            if ℓ_n(t) < ℓ_j(t) then
14:               if ||x_n(t) − x_j(t)|| < γ_d^j then
15:                  n_t = n
16:               end if
17:            end if
18:         end for
19:         p_jn(t) = g.calcProb(j) by Eq.(9)
20:         Select the highest p_jn(t)
21:         selectedVm = VmList.get(n)
22:         Update γ_d^j by Eq.(11)
23:      end for
24:      t = t + 1
25:   end while
26:   if selectedVm ≠ NULL then
27:      Return selectedVm
28:   end if
29: end for
```

Termination Condition: Iterations are indexed by t. The algorithm will be terminated if there is no improvement in reducing the execution cost from the last iteration.

5. SIMULATION

CloudSim 3.0.3 simulator [5] is used to evaluate the proposed algorithm. CloudSim is widely used to simulate cloud system components such as data centres and VMs. It supports policies for task scheduling, VM placement and selection, power models for data centre resources, and provides different types of workloads.

In CloudSim, task scheduling algorithms are deployed on the *Broker*, which has all the details relating to the VM creation and task scheduling to these VMs. The *Broker* is a Java object responsible for accepting and scheduling users' requests.

One data centre is created with two types of PMs. The characteristics of data centre and PMs are shown in Tables 3 and 4, respectively. Tasks are generated from a standard formatted workload of a NASA Ames Research Centre [6].

As mentioned in Krishnanand and Ghose [3], the choice of GSO parameters will impact on the performance of the algorithm. In terms of the total number of peaks captured, Krishnanand and Ghose suggested the parameter selection as in Table 5. Thus, only n_t and γ_s need to be selected.

We conduct experiments using GSOTS and three other scheduling algorithms: (i) STF, i.e., the shortest task is executed first, (ii) LTF, i.e., the largest task is executed first and (iii) PSO, i.e., task scheduling is based on PSO algorithm under the same condition, and compare performances

Table 3. Settings of data centre

Parameter	RAM	Storage	BW	VM scheduler	VMM
Value	2 GB	1 TB	10 GB	Time-shared	Xen

Table 4. Settings of PMs

PM	Processor	Pe	MIPS
PM1	Intel Core 2 Extreme X6800	2	27079
PM2	Intel Core i7 Extreme 3960X	6	177730

Table 5. Settings of GSO algorithm

Parameter	p	r	β	n_t	s	l_0
Value	0.4	0.6	0.08	5	0.03	0.05

of these four algorithms.

The simulations are done in the static and real workloads with time-shared policy. The performances of GSO, PSO, STF and LTF algorithms are compared in terms of total completion time, resource utilisation and execution cost.

5.1 Total Completion Time

In this section, we present two different scenarios. The first one uses fixed number of VMs with varying numbers of tasks while the second one uses varying numbers of VMs with fixed number of tasks.

Scenario 1: Fixed Number of VMs with Varying Numbers of Tasks

The number of VMs is fixed as 5 VMs and the number of tasks is gradually increased from 200 to 700 tasks. The y axis shows the effect on total completion time of increasing the number of tasks as shown in Figures 3a and 3b. It can be seen from Figures 3a and 3b that the total completion time increases over increasing number of tasks. Moreover, when the number of tasks is small, the differences between PSO and GSO is not obvious. Nevertheless, with the increasing number of tasks, the disparity of total completion time between the algorithms and GSO becomes larger. GSO outperforms PSO, STF and LTF in minimising the total completion time in both types of workload. PSO and STF come in second and third level respectively while LTF has the highest total completion time in both types of workload.

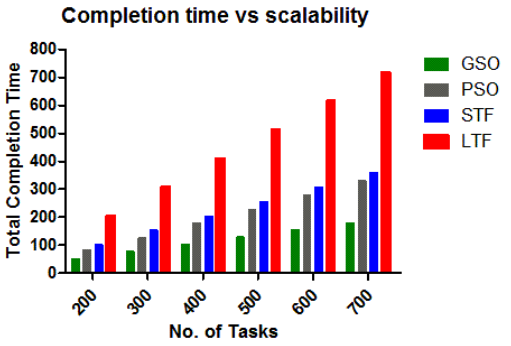
Scenario 2: Varying Numbers of VMs with Fixed Number of Tasks

The number of tasks is fixed as 100 tasks and the number of VMs is gradually increased from 5 to 20 VMs. The y axis shows the effect on total completion time over increasing number of VMs as shown in Figure 4.

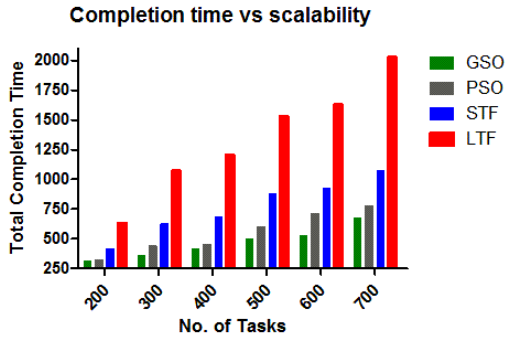
It can be seen from Figure 4 that with the increasing number of VMs, the total completion time of PSO and GSO scheduling algorithms shows different performance while the total completion time of LTF and STF algorithms remain stable, almost unchanging. In addition, when increasing the number of VMs, the total completion time of all algorithms are stabilised without linear reduction, which indicates that more VMs do not mean shorter the completion time given the set of tasks.

5.2 Resource Utilisation

The load of all VMs can be calculated by using the infor-



(a) Total completion time in the static workload



(b) Total completion time in the real workload

Figure 3. Comparisons of all algorithms in static and real workload with varying numbers of tasks.

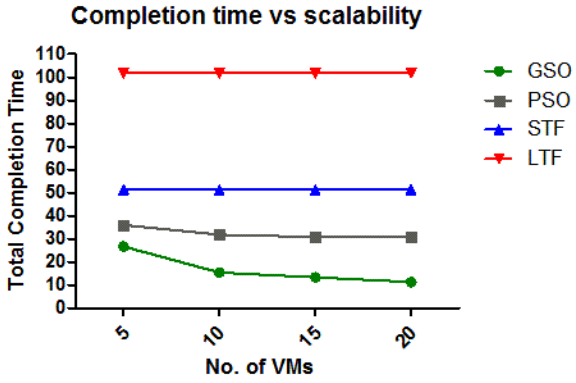


Figure 4. Total completion time in the static workload with varying numbers of VMs

Table 6. Mean and standard deviation of the completion time

Algorithm	μ	σ
GSO	41.20	23.04
PSO	41.20	33.33
STF	41.20	55.88
LTF	41.20	91.41

mation that is received from the data centre. Thus, standard deviation (SD) is calculated to measure the deviations of tasks' load on VMs. SD of load can be defined as below.

$$\sigma_j = \sqrt{\frac{\sum_{j=0}^{V}(CT_{ij} - \mu)^2}{V}} \tag{12}$$

where σ_j is the standard deviation of load, V is number of all VMs. CT_{ij} is completion time of executing task i which can be calculated as in Eq.3 and μ_j is mean completion time of V VMs which can be calculated as below.

$$\mu_j = \frac{\sum_{j=1}^{V} CT_{ij}}{V} \tag{13}$$

If σ_j of the VM is less than or equal to μ_j, then the system is in a balance state. On the other hand, if σ_j is higher than μ_j, then the system is in an imbalance state. As we can see from Table 6, σ in GSO and PSO is lower than μ, which means that using GSO and PSO for scheduling tasks not only minimises the total completion time but also keeps the load of VMs in a balance state. On the other hand, STF and LTF fail to balance the load of VMs.

5.3 Execution Cost

According to the type of the VM used to run a task and the time required to complete the task, the execution cost of task can be calculated using Eq.2. The number of VMs is fixed as 5 VMs and the number of tasks is gradually increased from 200 to 700 tasks. The y axis in Figure 5 shows the effect on execution cost of increasing the number of tasks. From Figure 5, it is found that GSOTS outperforms PSO algorithm with respect to the execution cost by 12.8% - 37.13%.

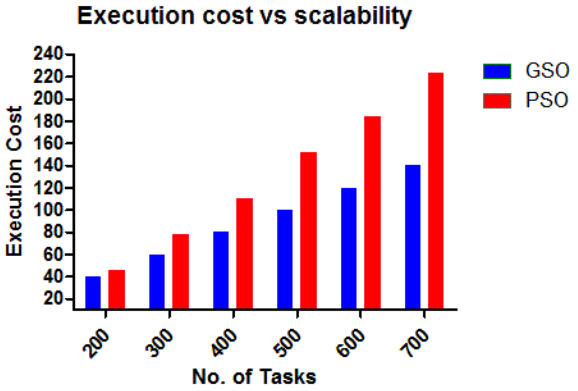


Figure 5. Execution cost in the static workload with varying numbers of tasks

6. RELATED WORK

As task scheduling is known to be a NP-hard problem, metaheuristic algorithms are efficient to solve it. Task scheduling based on GAs has been studied widely such as [7], [8], [9], [10]. Zhu *et al.* used hybrid GA algorithm to solve only load balancing when scheduling tasks in cloud computing [8].

Task scheduling based on ant colony optimisation (ACO) algorithm for load balancing and minimising the average execution time was studied in [11]. Simulation results showed that the proposed algorithm outperformed first come first serve (FCFS) and the basic ACO algorithms. A similar study by Tawfeek *et al.* also used ACO to minimise the execution time of tasks and the simulation results showed that the ACO outperformed FCFS and round robin (RR) algorithms [12]. Sun *et al.* improved the ACO algorithm to get a better performance when scheduling tasks in the cloud computing. The simulation results showed that the proposed algorithm had a good performance in minimising the execution time and balancing the load [13]. Task scheduling in view of both the task execution time and the system resource utilisation based on an improved PSO algorithm was proposed in [14]. In [15] and [16], a hybrid of PSO and simulated annealing (SA) was implemented on CloudSim to schedule tasks in the cloud. The results showed that the proposed algorithms can reduce the average execution time of task and increase resource utilisation.

In [17], a stochastic hill climbing algorithm was used to schedule tasks to VMs. Simulation results based on CloudAnalyst simulator showed the efficiency of the proposed algorithm when compared to RR and FCFS algorithms.

In [18], three different metaheuristics approaches (i.e., artificial bee colony (ABC), PSO and ACO) had been evaluated for cloud task scheduling. The proposed algorithms were better in minimising the total execution time compared to LTF, random and FCFS algorithms. Moreover, ABC algorithm outperformed other algorithms. The PSO and ACO came in second level and third level, respectively. Moreover, integrating ACO algorithm with GA for scheduling tasks was proposed by Dai *et al.* This algorithm considered multiple QoS constraints in the scheduling process and it had superior performance in balancing resources and minimising execution time [19].

However, these algorithms mainly focused on improving the execution time and resource utilisation when scheduling tasks to VMs. Moreover, the execution time did not include the waiting time of tasks while in our algorithm we consider the execution time and waiting time of tasks.

In [20], PSO was used to schedule workflow applications to cloud resources to minimise both computation and data transmission cost. PSO can achieve three times better cost saving compared to best resource selection (BRS) algorithm.

Zhang *et al.* considered the task scheduling across clouds and used GA to minimise the completion time and cost [10]. The total cost includes transmission cost and usage charges of the VMs, while the total completion time includes the transmission time and execution time of tasks. In our work, we do not consider any data transmission.

This paper proposes a solution for improving the execution cost of scheduled tasks while keeping the total comple-

tion time is minimised through the use of GSO algorithm. Completion time includes the execution and waiting time of tasks in VMs.

7. CONCLUSION

We have applied GSO algorithm to solve the task scheduling problem in cloud computing. The proposed algorithm aims to minimise the execution cost of task using Amazon EC2 pricing model under deadline constraint. The simulation results have shown that GSOTS outperforms PSO, STF and LTF algorithms in terms of minimising the total completion time when scheduling different number of tasks and VMs in cloud. In addition, GSOTS can utilise the resources more efficiently than other algorithms. Regarding the cost, GSOTS is able to reduce the execution cost more than PSO algorithm. Extending the GSOTS algorithm to consider dynamic task scheduling is one of our future work.

8. REFERENCES

[1] T. A. Genez, L. F. Bittencourt, and E. R. Madeira. Workflow scheduling for saas/paas cloud providers considering two sla levels. In *IEEE Network Operations and Management Symp.,(Maui,16-20 Apr.),* pages 906-912. IEEE, 2012.

[2] B. Jennings and R. Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Sys. Management,* 23(3):567-619, 2015.

[3] K. Krishnanand and D. Ghose. Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications. *Multiagent and Grid Sys.,* 2(3):209-222, 2006.

[4] D. Alboaneen, H. Tianfield, and Y. Zhang. Glowworm swarm optimisation algorithm for virtual machine placement in cloud computing. In *Int. IEEE Conf. on Ubiquitous Intelligence Comp., Advanced and Trusted Comp., Scalable Comp. and Communications, Cloud and Big Data Comp., Internet of People, and Smart World Congress,(Toulouse, 18-21 July),* pages 1-7, 2016.

[5] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience,* 41(1):23-50, 2011.

[6] D. G. Feitelson, D. Tsafrir, and D. Krakov. Experience with using the parallel workloads archive. *Journal of Parallel and Dist. Comp.,* 74(10):2967-2982, 2014.

[7] C. Zhao, S. Zhang, Q. Liu, J. Xie, and J. Hu. Independent tasks scheduling based on genetic algorithm in cloud computing. In *5th Int. Conf. on Wireless Communications, Networking and Mobile Comp.,(Beijing, 24-26 Sep),* pages 1-4. IEEE, 2009.

[8] K. Zhu, H. Song, L. Liu, J. Gao, and G. Cheng. Hybrid genetic algorithm for cloud computing applications. In *Services Comp. Conf., IEEE Asia-Pacific,(Jeju Island, 12-15 Dec),* pages 182-187. IEEE, 2011.

[9] S. H. Jang, T. Y. Kim, J. K. Kim, and J. S. Lee. The study of genetic algorithm-based task scheduling for cloud computing. *Int. Journal of Control and Automation,* 5(4):157-162, 2012.

[10] M. Zhang, Y. Yang, Z. Mi, and Z. Xiong. An improved genetic-based approach to task scheduling in inter-cloud environment. In *Ubiquitous Intelligence and Comp. and IEEE 12th Intl. Conf. on Autonomic and Trusted Comp. and IEEE 15th Intl. Conf. on Scalable Comp. and Communications and Its Associated Workshops, IEEE 12th Intl. Conf. on,(Beijing, 10-14 Aug),* pages 997-1003. IEEE, 2015.

[11] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang. Cloud task scheduling based on load balancing ant colony optimization. In *6th Annual ChinaGrid Conf.,(Liaoning, 22-23 Aug),* pages 3-9. IEEE, 2011.

[12] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey. Cloud task scheduling based on ant colony optimization. In *Comp. Engineering & Sys., 8th Int. Conf. on,(Cairo, 26-28 Nov),* pages 64-69. IEEE, 2013.

[13] W. Sun, N. Zhang, H. Wang, W. Yin, and T. Qiu. Paco: A period aco based scheduling algorithm in cloud computing. In *Cloud Comp. and Big Data (CloudCom-Asia), Int. Conf. on,(Fuzhou,16-19 Dec),* pages 482-486. IEEE, 2013.

[14] Z. Liu and X. Wang. A pso-based algorithm for load balancing in virtual machines of cloud computing environment. In *Int. Conf. in Swarm Intelligence,(Berlin, 17 June),* pages 142-147. Springer, 2012.

[15] S. Zhan and H. Huo. Improved pso-based task scheduling algorithm in cloud computing. *Journal of Info. & Computational Science,* 9(13):3821-3829, 2012.

[16] H. S. Al-Olimat, M. Alam, R. Green, and J. K. Lee. Cloudlet scheduling with particle swarm optimization. In*Communication Sys. and Network Tech., 5th Int. Conf. on,(Gwalior, 4-6 Apr.),* pages 991-995. IEEE, 2015.

[17] B. Mondal, K. Dasgupta, and P. Dutta. Load balancing in cloud computing using stochastic hill climbing-a soft computing approach. *Procedia Tech.,* 4:783-789, 2012.

[18] G. F. Elhady and M. A. Tawfeek. A comparative study into swarm intelligence algorithms for dynamic tasks scheduling in cloud computing. In *IEEE 7th Int. Conf. on Intelligent Comp. and Info. Sys.,(Cairo, 12-14 Dec),* pages 362-369. IEEE, 2015.

[19] Y. Dai, Y. Lou, and X. Lu. A task scheduling algorithm based on genetic algorithm and ant colony optimization algorithm with multi-qos constraints in cloud computing. In *Intelligent Human-Machine Sys. and Cybernetics, 7th Int. Conf. on,(Hangzhou, 26-27 Aug.),* volume 2, pages 428-431. IEEE, 2015.

[20] S. Pandey, L. Wu, S. M. Guru, and R. Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *24th IEEE int. conf. on advanced info. networking and applications,(Perth, 20-23 Apr.),* pages 400-407. IEEE, 2010.