De-centralised dynamic task scheduling using hill climbing algorithm in cloud computing environments

Mona Mohammadi and
Amir Masoud Rahmani*

Department of Computer Engineering, Science and Research Branch,
Islamic Azad University,
Tehran, Iran
Email: monamohamadi801@yahoo.com
Email: rahmani@srbiau.ac.ir
*Corresponding author

Abstract: There are many tasks in cloud computing that should be executed by available resources to acquire high performance, reduce task completion time, increase utilisation of resources and etc. Since task scheduling problem in cloud computing is an NP-hard problem, designing an efficient scheduling strategy to achieve the intentions listed above is challenging. Task scheduling is the process of allocating tasks to available resources such that performance metrics are improved. This paper proposes a dynamic scheduling algorithm that uses hill climbing algorithm. It tries to minimise completion time of tasks while maximising throughput and utilisation of resources. This algorithm allocates independent tasks to available resources to achieve load balance. The simulation results show that the algorithm can achieve load balance and reduces completion time of tasks.

Keywords: cloud computing; task scheduling; dynamic scheduling algorithm; hill climbing algorithm; load balancing.

Reference to this paper should be made as follows: Mohammadi, M. and Rahmani, A.M. (2017) 'De-centralised dynamic task scheduling using hill climbing algorithm in cloud computing environments', *Int. J. Cloud Computing*, Vol. 6, No. 1, pp.79–94.

Biographical notes: Mona Mohammadi received her BSc in Computer Engineering from Islamic Azad University, Tehran-North, in 2008. She received her MSc in Computer Engineering from Science and Research Branch, Islamic Azad University of Tehran, Iran, in 2016. Her major interests are distributed systems, grid and cloud computing.

Amir Masoud Rahmani received his BS in Computer Engineering from Amir Kabir University, Tehran, in 1996, his MS in Computer Engineering from Sharif University of technology, Tehran, in 1998 and his PhD degree in Computer Engineering from IAU University, Tehran, in 2005. He is a Professor of Computer Department at the IAU University. He is the author/co-author of more than 140 publications in technical journals and conferences. He served on the program committees of several national and international conferences. His research interests are in the areas of distributed systems, ad hoc and sensor wireless networks, scheduling algorithms and evolutionary computing.

1 Introduction

Cloud computing is defined by NIST (Mell and Grance, 2011) as:

> "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models."

Cloud is an extension of distributed computing. It is also a collection of parallel and distributed systems (Katyal and Mishra, 2013). It provides secure, quick, convenient data storage and net computing services.

The three service models of cloud computing are as follows:

- software as a service (SaaS)
- platform as a service (PaaS)
- infrastructure as a service (IaaS)

These services are available to users in a pay-per-use-on-demand model, which can access shared IT resources like server, data storage, application, network and so on through internet (Agarwal and Jain, 2014; Selvarani and Sadhasivam, 2010). In IaaS, computing infrastructure is provided as a service to the requester in form of virtual machine (VM). VM is a software that implements a VM on a running physical machine. Each VM (that is created) behaves like a physical machine; it runs different operating systems and applications (Achar et al., 2012).

In cloud computing environment, a task is defined as an atomic unit that should be scheduled and assigned to a resource by scheduler. A job or a meta-task is a set of atomic tasks that are considered to be scheduled. Based on tasks dependency, the tasks are divided into two categories: independent and dependent tasks. The tasks that do not need to communicate with each other are called independent tasks. Dependent tasks have precedence order and they are executed in order of precedence relation (Annette et al., 2013; Bessai et al., 2012). Recently, the task scheduling problem in cloud computing has attracted attention of many researchers. Task scheduling process in cloud can be divided into three stages namely (Agarwal and Jain, 2014; Jiang and Sheng, 2012):

1 Resource discovering and filtering:

   Data centre broker discovers the resources that are available in the network system (by querying CIS) and collects status information that are related to resources.

2 Resource selection:

   Suitable resource is selected based on scheduling criteria and certain parameters of task and resource. This is the deciding stage and this paper focuses on this stage and it proposes a new algorithm for allocating tasks to resources.

3 Task submission:

   The task is submitted to the resource that is selected.

Task scheduling in distributed systems such as cloud computing is the allocation of tasks to available resources in a way that improves performance metrics. The distributed system consists of computing machines that have different performances; these machines are connected to each other by high-speed communication paths. Since, in allocation of T tasks to H VMs, the number of allocations will be $|H|^{|T|}$ and the number of running states will be |T|!, the task scheduling problem in distributed systems is an NP-hard problem (Gomathi and Krishnasamy, 2013; Rahmani and Rezvani, 2009). Since the task scheduling problem is an NP-hard problem, deterministic solutions are not efficient solutions; therefore, researchers use random and heuristic search methods. The most well-known heuristic techniques are the algorithms which improve the scheduling metrics iteratively, probabilistic optimisation algorithms and the constructive heuristic algorithms (Kashani and Jahanshahi, 2009; Navimipour et al., 2014).

If cloud scheduler knows the workload of all data-centres and the scheduler independently allocates tasks to VMs, this kind of scheduling is called centralised task scheduling. The centralised task scheduling facilitates finding the appropriate VM for each task that is in the entire cloud, but it is only suitable for small-scale clouds. If cloud schedulers cooperate with each other to find suitable VM for task, this kind of scheduling is called de-centralised task scheduling (Wang et al.,2006). Scheduling methods in distributed systems can be classified into two categories: heuristic and hybrid techniques. Heuristic methods are classified into static and dynamic scheduling (Mathew et al., 2014).

This paper proposes a de-centralised dynamic heuristic scheduling algorithm that uses hill-climbing algorithm which schedules tasks between data-centres. In dynamic scheduling, tasks arrive at different times in run time of the application and scheduling decisions are made in run time and the tasks are allocated to resources in run time of application also (Annette et al., 2013).

Dynamic scheduling can be performed in online mode or batch mode. The proposed algorithm performs scheduling in online mode. Dynamic scheduling algorithms are more suitable in heterogeneous environment of cloud computing. But the overhead of dynamic algorithms is high, because scheduling decisions are made immediately and system information is updated instantly (Mathew et al., 2014).

There are many algorithms like Min-Min, Max-Min, smallest cloudlet to fastest processor (SCFP), activity-based costing (ABC) and some meta-heuristics like genetic algorithm (GA) and simulated annealing (SA) that already exist for tasks scheduling problem, but most of the existing algorithms are centralised whereas the cloud computing is a distributed system. Our proposed algorithm is de-centralised and it focuses on optimising the task scheduling algorithm with a meta-heuristic algorithm called hill climbing algorithm. Hill climbing is a solution to certain classes of optimisation problems. The idea is started with a sub-optimal solution (sub-optimal resource) to a task scheduling problem (*start at the base of a hill*) then the solution is converted into a better solution repeatedly (*walk up the hill*) until there is no neighbour that has a better solution (*the top of the hill is reached*).

Using the hill climbing algorithm and considering completion time metric in the proposed task scheduling algorithm, makes the algorithm to allocate independent tasks to processing resources in order to achieve load balance. Load balancing is a method which distributes workloads across VMs such that no VM is overloaded while other VMs are under-loaded.

Load balancing is taken into account as a vital part for parallel and distributed systems. It helps cloud computing systems by improving the general performance, better computing resources utilisation, energy consumption management, enhancing the cloud services' QoS, avoiding SLA violation and maintaining system stability through distribution, controlling and managing the system workloads (Mesbahi and Rahmani, 2016). Load balancing helps optimal utilisation of resources and enhances the performance of the system.

The rest of this paper is organised as follows. In Section 2, some previous works about task scheduling methods are described. In Section 3, we propose our distributed scheduling algorithm. In Section 4, we evaluate our developed model and analyse the simulation results and finally we present our conclusions and future works in Section 5.

2 Related work

As mentioned above, there are several task scheduling approaches that have already been proposed in cloud computing environments. In the following, we explain some of previous studies.

Santhosh and Ravichandran (2013) have presented a scheduling approach that focuses on providing a solution to online scheduling problem. This algorithm uses 'IaaS' model that is offered by cloud computing. Real-time tasks are submitted to processing resources in a way that maximises total utility and efficiency. This scheduling algorithm aborts the task when it misses its deadline. Whenever a task misses its deadline, it will be migrated to another VM. This method improves the overall performance of the system and maximises the total utility.

Liu et al. (2013) have introduced a task scheduling model that is based on MO-GA algorithm. This algorithm decreases energy consumption while increasing the service provider's profit. The steps of the algorithm are as follows: at first, initial population is generated by greedy and random methods, then the individuals are modified by the operators which operate during the evolution process of the MO-GA algorithm, solutions (individuals) are ranked based on the fitness function. Only the solutions which have the best rank are stored in the Pareto archive which contains different non-dominated solutions that are generated during creation of the generations. The results are stored at external Pareto archive and then the optimal solution is selected.

Kaur and Verma (2012) have presented a scheduling algorithm which is based on meta-heuristic algorithms. It decreases execution time and execution cost. At first, the algorithm generates an initial population. The initial population is the output of scheduling algorithms such as longest cloudlet to fastest processor (LCFP), smallest cloudlet to fastest processor (SCFP) and eight random schedules, and then the fitness function is evaluated for each individual. While the number of generations is not 13, the algorithm does the following steps:

1 it selects fitter individuals for reproduction and crossovers between individuals by two-point crossover

2 it mutates individuals by simple swap operator and evaluates the fitness function of the modified individuals and then generates a new population.

Ma et al. (2013) have proposed a feasible and flexible dynamic task scheduling algorithm called DGS, which allocates virtual resources dynamically. This algorithm uses greedy method to allocate tasks to appropriate processing resources. DGS monitors and calculates the actual amount of resources required for applications; then it dynamically adjusts virtual resources to increase utilisation of resources and achieve load balance. Improved greedy strategy is used to allocate tasks to appropriate VMs dynamically in order to respond quickly to users.

Chawla and Bhonsle (2013) present dynamically optimised cost-based task scheduling algorithm which efficiently allocates tasks to available resources in cloud. To solve the task scheduling problem, a greedy approach is used. In this algorithm, the cost-based tasks are prioritised based on task profit in a descending order. The tasks higher profit can be allocated to the machines with minimum cost, which in turn makes maximum profit. The resource with minimum cost is selected and tasks are allocated to it until its capacity is completed. The process of selecting the resource and allocating tasks is sequential since they are prioritised based on the user's needs.

Achar et al. (2012) have presented a scheduling algorithm, which uses tree-based data structure that is called virtual machine tree (VMT) and executes tasks efficiently. The algorithm is as follows: first, the scheduling algorithm prioritises the tasks and VMs and then VMT is constructed. Each node of the tree represents a VM. Number of groups of tasks is equal to the number of leaves in the VMT. When grouping the tasks is done, suitable VMs from VMT are selected for execution. The tasks in each group are selected sequentially(based on their priority) and submitted to the selected VM.

Jiang and Sheng (2012) have proposed a graph-based task scheduling algorithm, which considers both private resources and public resource to achieve minimum. First, a bipartite graph is built. The vertices of bipartite graph can be divided into two disjoint sets U and V such that every edge connects a vertex of U to one vertex of V. U and V are independent sets. U denotes task collection, and V denotes public VMs or private VM collection. The edge's weight (that connects one task from the U collection to one VM in V collection) is the cost of executing task in VM. The Hopcroft-Karp algorithm is used for matching vertices in the bipartite graph. Edges are chosen such that no endpoint is shared between the two edges and the edges' weights are minimal. The Hopcroft-Karp algorithm repeatedly increases the size of a partial matching by finding augmenting paths.

Malik et al. (2012) have proposed a model for grouping VMs based on network latency. The algorithm has three different phases. The first phase is the initialisation phase which is the pre-processing phase of the algorithm. In this phase, variables are initialised. The grouping decisions in the next phase are made based on these initial pre-configurations. The next phase of the algorithm is the configuration phase which is the group discovery. In this phase, the algorithm groups the VMs based on network latency. The algorithm runs for each VM and it decides for grouping VMs by finding latency to its neighbours. The final phase of the algorithm is the reconfiguration phase. In this phase, the algorithm creates new groups and adds the existing leaders of the old groups to each of the new groups. This phase of the algorithm tries to group the VMs, which belong to a single VM group. These VMs are candidate VMs that can be grouped with some other VMs. In this phase, every candidate VM is probed for latency to the leader of all the groups. It tries to find a leader VM that has minimum latency among all, so that the minimum latency should be smaller than threshold. If the minimum latency is

less than the threshold then it groups the candidate VM with the leader VM group that has minimum network latency.

Wu et al. (2013) have proposed an algorithm that is based on QoS-derive in cloud computing. First, the algorithm computes priority of tasks and sorts them in a descending order based on their priorities and then allocates each task to the service that can complete the task execution as soon as possible.

Saxena and Chauhan (2014) have presented an algorithm which is called dynamic fair priority task scheduling algorithm. First, it groups tasks into two groups, namely: deadline-based and cost-based tasks. For deadline-based tasks, the algorithm arranges them into three priority queues: high, mid and low, then it takes one task from each queue and computes turn-around time of each VM. For each task, the VM that has minimum turn-around time is selected and the task is allocated to the selected VM. For cost-based tasks group, the algorithm arranges tasks into three priority queues: high, mid and low, then it computes cost of each VM and each task is allocated to the VM whose MIPS is bigger than task's length and has minimum cost.

Agarwal and Jain (2014) have presented a generalised priority algorithm for efficient execution of tasks. In the proposed strategy, tasks are initially prioritised based on their length and VMs are also ranked (prioritised) based on the value of their MIPS and then the task which has the highest length is allocated to the VM which has the highest MIPS.

Selvarani and Sadhasivam (2010) have presented an algorithm which improves the ABC algorithm which was presented in Ingole et al. (2009). ABC prioritises the tasks and uses cost drivers. It helps solving problems like poor cost control and starvation. In this algorithm, first the resources are selected, then the priority levels of the tasks are calculated, then the tasks are sorted according to their priority then they are placed in three different lists named: high priority, medium priority and low priority and then job grouping algorithm is applied to the above lists in order to allocate task-groups to different available resources.

Omidi and Rahmani (2009) have proposed an algorithm which schedules independent tasks in multiprocessor systems. The proposed algorithm is based on particle swarm optimisation (PSO) algorithm which is a heuristic algorithm.

Huh et al. (2012) have presented an algorithm which limits the maximum virtual runtime difference among cores. For a pair of cores, the algorithm periodically sorts tasks in the descending order of their virtual run time. It executes the partition algorithm and the migrate algorithm. It limits the load difference between two cores with the largest weight in the task collection.

Bessai et al. (2012) have presented an algorithm which has two main approaches: cost-based approach and time-based approach. The time-based approach tries to decrease the overall completion time (i.e., execution and communication time). The cost-based approach tries to decrease overall cost. As the cost-based approach, the time-based approach has three major phases which are listed below:

1 *Tasks sorting phase:* in this phase, the tasks of the workflow which are independent of each other are grouped. The tasks which are in the same workflow level are independent tasks which can be executed simultaneously.

2 *Resource allocation phase:* in this phase, the cost-based approach and the time-based approach are different. The first approach allocates task to the VM which has minimum execution cost and minimum communication cost (CC) while the second approach attempts to allocate task to the VM which has minimum execution time and

minimum communication time. The task allocation in the two approaches starts at k^{th} level. The following strategies determine the value of k:

a top-down

b bottom-up

c mixed allocation strategy.

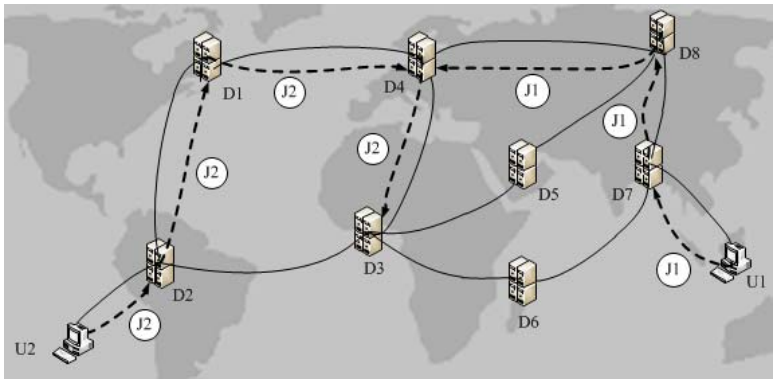
3 *Pareto selection phase:* at the end of the previous phase, L task assignments are obtained. In this phase, the overall completion time of each task assignment is computed; then only non-dominated solutions are maintained.

3 Proposed algorithm

This paper proposes a de-centralised dynamic task scheduling which is called distributed scheduling hill climbing (DSHC). It uses hill climbing algorithm to schedule tasks between neighbour data-centres. Since we use CloudSim toolkit for simulation of the algorithm, according to the CloudSim toolkit, scheduling of CPU resources can be classified into two levels: host and VM. At host level, host shares cores or processing elements (PEs) with each VM that is running on it. In the VM level, each VM divides the resources between tasks that are running on it. Each VM receives resources from the host. If tasks can move between two certain data-centres, the two data-centres are neighbour data-centres. As shown in Figure 1, each data-centre has its own scheduler. In this distributed system, cloud topology is static and does not change with time. The workload of the entire cloud is not changed during the run time. Users submit their tasks to the scheduler of data-centres during the run time, therefore DSHC algorithm is dynamic.

After one time or several times scheduling between neighbouring data-centres, the scheduler allocates the task to the suitable VM. Initial data-centre is the first data-centre that the task is submitted to before executing. For example, D2 is the initial data-centre for task J2. After three times scheduling between neighbouring data-centres, scheduler of D3 allocates the task J2 to the suitable VM.

Figure 1 Distributed scheduling algorithm based on hill climbing (see online version for colours)



Notes: D1,…,D8: data-centres,U1, U2: users, J1, J2: jobs, dashed lines: job movement, straight lines: connection between neighbour data-centres.

In the following, we list the steps of the DSHC algorithm for independent tasks:

1 computing the minimum completion time (MCT) of task in the initial data-centre

2 computing the response time (RT) of neighbouring data-centres

3 selecting suitable VM and allocating the task to the VM.

In the following sections, we describe the steps of the DSHC algorithm.

3.1 Computing the MCT of task in the initial data-centre

When the user submits task to the initial data-centre, the scheduler (which is in initial data-centre) estimates the completion time of the task in each VM that is in the initial data-centre and finds the VM that has MCT, then initial data-centre sends the machine instructions (MI) of the task to all of the neighbour data-centres.

Since the scheduling policy at VM level is space-shared, we compute estimated completion time (ECT) of the task as:

$$\mathrm{ECT} = \mathrm{EST} + \frac{\mathrm{MI}}{\mathrm{AveMIPS} * \mathrm{Cores}} + \mathrm{FTT} \tag{1}$$

where Cores is the number of cores that the task requires. EST is the earliest start time of task. File transfer time (FTT) is the time that the required files for executing tasks move from the storage area network (SAN) to the VM. The AveMIPS is the average processing power of cores which are in a host which has m processing cores. We compute AveMIPS as:

$$\mathrm{AveMIPS} = \sum_{i=1}^{m} \frac{\mathrm{CAP}(i)}{m} \tag{2}$$

where CAP(i) is the processing power of core i.

The EST depends on the position of the task in the waiting queue. Since tasks use processing cores exclusively (space-shared mode), they are put in waiting queue. The tasks are in waiting queue until there are enough processing cores so they can be executed by available VMs.

3.2 Computing the RT of neighbouring data-centres

When each scheduler which is in the neighbouring data-centre receives the MI of the task from the initial data-centre, the scheduler estimates the completion time of the task in all of its VMs and considers the VM with MCT for the task then each neighbouring data-centre computes the RT and sends it to the initial data-centre.

RT equals to sum of the MCT of the task in the neighbouring data-centre and CC between initial data-centre and neighbouring data-centre. CC equals to sum of the transmission time and propagation time. We compute The CC and the RT as:

$$\mathrm{CC} = \frac{\mathrm{MI}}{\mathrm{R}} + \mathrm{RTT} \tag{3}$$

$$\mathrm{RT} = \mathrm{CC} + \mathrm{MCT} \tag{4}$$

The MI is the machine instructions or processing requirements of the task and R is the data transfer rate between initial data-centre and in neighbouring data-centre and round-trip time (RTT) is the propagation time of the task. MCT is the minimum time that the task can complete execution in neighbour data-centre.

3.3 Selecting suitable VM and allocating the task to the VM

When the initial data-centre receives the RTs from the neighbours, it finds minimum response time (MRT). Initial data-centre compares MRT with MCT of the task in its VM. If the MCT of the task in initial data-centre is less than the MRT of the neighbour, then the scheduler which is in the initial data-centre allocates the task to its VM which has the MCT for the task. Otherwise, the initial data-centre selects the neighbours which their RT are less than MCT of the task, then roulette wheel selection algorithm randomly selects one data-centre from the data-centres which have already been selected.

Probability of selecting each neighbour data-centre is equal to fitness value of the neighbour. The neighbouring data-centre which has better fitness has more chances to be selected. The fitness value of each neighbouring data-centre is:

$$\text{Fitness}(n) = \frac{1/\text{ResponseTime}\ (n)}{\sum_{m=1}^{k} 1/\text{ResponseTime}\ (m)} \tag{5}$$

where Fitness(n) is the fitness value of the neighbouring data-centre n, ResponseTime(n) is the RT of the neighbouring data-centre n, k is the number of data-centres which the initial data-centre selects. The data-centre which roulette wheel selection algorithm selects is named Sel-D.

If RT of data-centre Sel-D is less than the deadline of the task, then initial data-centre sends the task to the data-centre Sel-D, otherwise the scheduler allocates the task to the VM which is in initial data-centre and has MCT.

If initial data-centre sends the task to the data-centre Sel-D, then data-centre Sel-D performs all the steps that the initial data-centre has done until now, hence data-centre Sel-D examines each neighbouring data-centre. If the neighbouring data-centre has a lower RT that is less than deadline, the Sel-D sends the task to the neighbouring data-centre. The data-centre Sel-D is replaced with the neighbouring data-centre. This process is repeated until there is no neighbour that has lower RT, thus we use the hill climbing algorithm in the DSHC algorithm.

The pseudo code of the hill climbing algorithm which DSHC algorithm uses is shown in the following.

Hill climbing algorithm

1 Start with initial state (initial data-centre)

2 **While** (neighbour of initial data-centre has lower response time that is less than deadline)

Do:

- 2.1 Send the task to the neighbour data-centre.
- 2.2 Replace the initial data-centre with neighbour data-centre.

Initial data-centre ← neighbour data-centre.

The pseudo code of the DSHC algorithm is shown as follows.

Distributed scheduling hill climbing (DSHC) algorithm

Input Tasks

Output The scheduled list

Step 1 Initial data-centre receives task that should be scheduled by the scheduler

Step 2 The scheduler computes completion time of the task in each VM which is in initial data-centre then it finds the VM that has MCT.

Step 3 Initial data-centre sends the MI of the task to neighbours.

Step 4 Each neighbour data-centre finds the VM that has MCT, and then the neighbour computes the response time

$$RT = MCT + CC$$

$$CC = \frac{MI(bit)}{R(bps)} + RTT$$

Step 5 Each neighbour sends the RT to the initial data-centre.

Step 6 When the scheduler which is in the initial data-centre, receives RTs from the neighbours, it finds the minimum RT.

Step 7 **if** MCT of the task in the initial data-centre is less than the minimum RT of the neighbour

Then

Step 8 The scheduler allocates the task to the VM which is in initial data-centre and has MCT for the task.

else

Step 9 Initial data-centre selects neighbours which their RT are less than MCT of the task in initial data-centre.

Step 10 Initial data-centre computes fitness value of each neighbour that has already been selected.

$$Fitness(n) = \frac{1 / ResponseTime(n)}{\sum_{m=1}^{k} ResponseTime(m)};$$

Step 11 Roulette wheel selection algorithm selects one data-centre according to fitness value of the data-centre. The data-centre which has better fitness has more chances to be selected.

The data-centre which roulette wheel selection algorithm selects is named Sel-D.

Step 12 **if** the RT of data-centre Sel-D, is less than deadline **then**

Step 13 initial data-centre sends task to the data-centre Sel-D

Step 14 **if** data-centre Sel-D has neighbours **then**

Initial data-centre is replaced with data-centre Sel-D and data-centre Sel-D performs steps 1 to 12.

Step 15 **else**

Step 16 scheduler allocates task to the VM which is in data-centre Sel-D and has MCT for the task

else

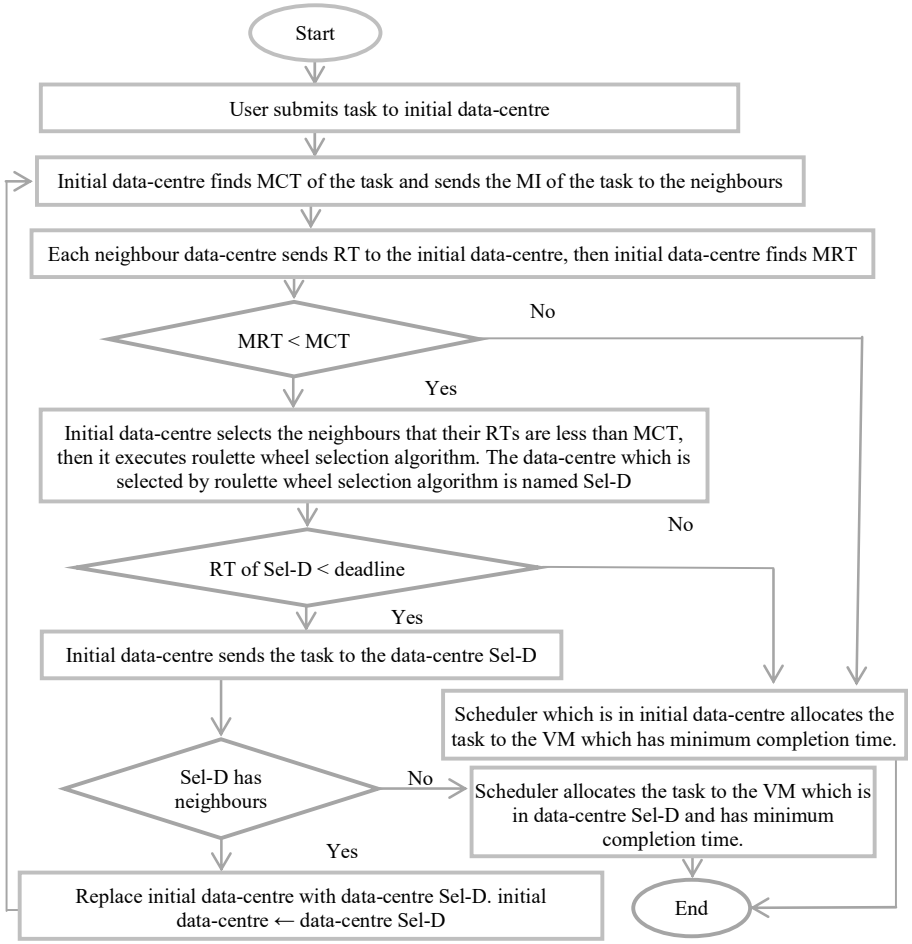
Step 17 Scheduler allocates the task to the VM which is in initial data-centre and has minimum completion

In the distributed system, if neighbours of a data-centre simultaneously send the MI of their tasks to the data-centre then the data-centre computes RT of each neighbour that send the MI of the task. The data-centre sends the RTs to the neighbours. After that, if one neighbour sends task to the data-centre, the data-centre updates the RTs and sends the updated RTs to the neighbours which have sent the MI. The pseudo code is shown as follows.

Updating response time	
Step 1	**if** neighbours send the MI of their task to a data-centre **simultaneously then**
	The data-centre computes response time of the neighbours (that sent the MI of task), then the data-centre sends the response times to the neighbours.
Step 2	**if** one neighbour sends task to the data-centre **then**
	The data-centre sends the updated response time to the neighbours which have sent the MI of the task.

Figure 2 shows the flowchart of DSHC algorithm.

Figure 2 Flowchart of DSHC algorithm



4 Simulation results

We used CloudSim 3.0.3 to evaluate the DSHC algorithm's performance. The CloudSim toolkit supports modelling of cloud's components, such as data-centres, hosts, VMs, scheduling and resource provisioning policies. We implemented three task scheduling algorithms: the DSHC algorithm, the local-round robin algorithm and the random algorithm.

4.1 The evaluation criteria

We evaluated the three algorithms which are mentioned above. The scheduling criteria are as follows:

1 *Completion time* is the time that task execution is completed and user can see the result.

2 ∂ is a criterion for evaluating whether a system is well load balanced or not. ∂ is the workload's standard deviation. We can compute the standard deviation as:

$$\delta = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}} \tag{6}$$

where $n \in N$ is the number of VMs, N is a natural number, x_i is the load of VM_i and μ is the average load of VMs.

4.2 Simulation setup

We evaluated the above criteria in simulation of the algorithm with 20–100 tasks in 20 VMs. In each data-centre, the number of processing cores is 6 and the number of hosts is 2. Table 1 shows the hosts' specifications.

Table 1 Hosts' specifications

Hosts	*Host 1*	*Host 2*
No of PEs	2	4
RAM(MB)	10,204	5,024
VM scheduling	Time shared	Time shared

The processing power of cores is limited in an interval of [1,000, 2,000]. The length of the tasks is limited to [,2000, 6,500]. The CC between neighbouring data-centres is limited to [0–2] seconds.

4.3 Simulation results and analysis

4.3.1 Effect on completion time

The average completion time of tasks in three algorithms are represented in Table 2.

Table 2 comparison of average completion time of tasks

No of cloudlets	*Time(seconds)*		
	DSHC	*Random*	*Local-round robin*
20	3.25	4.38	4.26
40	5.14	8.07	6.99
60	8.13	11.22	10.36
80	11.39	14.66	13.46
100	14.45	18.46	16.906

Figure 3 Comparison of average completion time of tasks (see online version for colours)

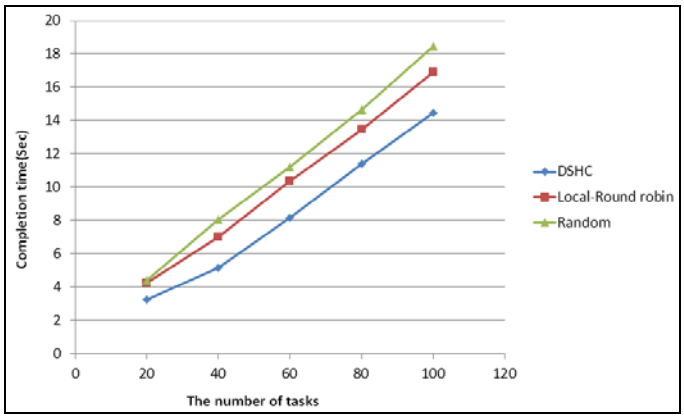


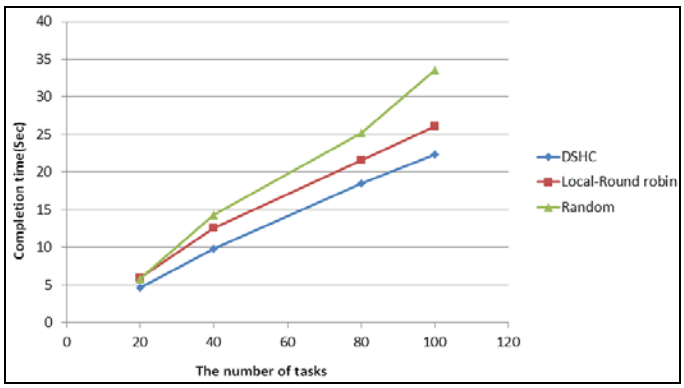
Table 3 Comparing the average completion time of final tasks in data-centres

No of cloudlets	*Time(seconds)*		
	DSHC	*Random*	*Local-round robin*
20	4.64	5.9	5.98
40	9.8	14.34	12.52
80	18.56	25.25	21.64
100	22.37	33.6	26.05

Figure 3 shows the average completion time of tasks in three algorithms. It can be seen that in the random algorithm, the average completion time of tasks is the highest. Since the DSHC algorithm allocates task to the VM that has lower completion time, compared to the random and the local-round robin, the average completion time of tasks in the DSHC is the lowest. The average completion time of the final tasks in data-centres are illustrated in Figure 4 and Table 3. It can be seen that the average completion time of the final tasks in the DSHC is the lowest. Since the scheduling criterion in this algorithm is

the completion time, tasks are allocated to the VMs that have the MCT. In this way, the final tasks complete at lower time.

Figure 4 Comparing the average completion time of final tasks in data-centres (see online version for colours)



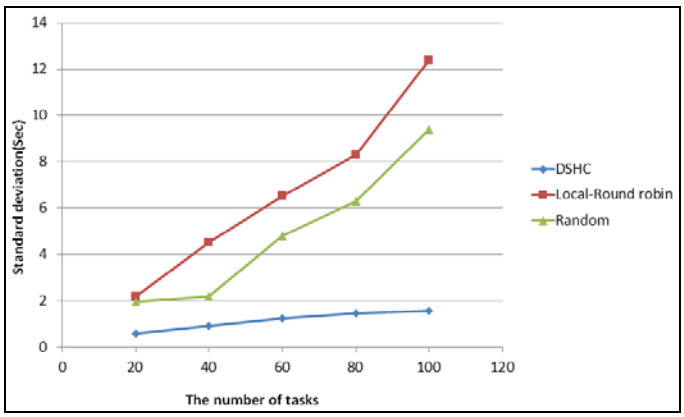
4.3.2 *Effect on load balancing (∂)*

The standard deviations in three algorithms are illustrated in Figure 5 and Table 4.

Table 4 Comparing standard deviations in three algorithms

No of cloudlets	*Time(seconds)*		
	DSHC	*Random*	*Local-round robin*
20	0.6	1.98	2.18
40	0.93	2.18	4.53
60	1.26	4.8	6.54
80	1.48	6.29	8.3
100	1.56	9.38	12.38

Figure 5 Comparing standard deviations in three algorithms (see online version for colours)



It can be seen that ∂ in local-round robin is the highest because in this algorithm, workloads of data-centres are different; one data-centre is overloaded while the other data-centres are underloaded. The system is load-imbalanced.

Since the DSHC algorithm allocates each task to the VM that has the MCT, standard deviation in this algorithm is the lowest.

5 Conclusions and future work

In this paper, we have introduced DSHC algorithm as a mechanism for scheduling tasks in cloud computing. This algorithm uses hill climbing algorithm for scheduling tasks between neighbouring data-centres. This may result in good performance and load balance. Compared to the random algorithm and local-round robin algorithm, the completion time of tasks is decreased in the DSHC algorithm.

The drawback of the proposed algorithm is that it does not schedule dependent tasks which have precedence relation. The energy management of a data-centre depends on various factors among which, task scheduling is a significant factor, but the DSHC does not consider energy consumption and does not provide fault-tolerance.

In future we plan to extend DSHC to consider dynamic cloud topology and dynamic workload of the entire cloud. In addition we plan to compare the DSHC with two new heuristic algorithms. We also plan to extend DSHC algorithm to provide fault-tolerance.

References

Achar, R., Thilagam, P.S., Shwetha, D., Pooja, H., Roshni and Andrea (2012) 'Optimal scheduling of computational task in cloud using virtual machine tree', *Third International Conference on Emerging Applications of Information Technology (EAIT)*, IEEE, pp.143–146.

Agarwal, A. and Jain, S. (2014) 'Efficient optimal algorithm of task scheduling in cloud computing environment', *International Journal of Computer Trends and Technology (IJCTT)*, Vol. 7, No. 5, pp.344–349.

Annette, J.R., Banu, W.A. and Shriram, S. (2013) 'A taxonomy and survey of scheduling algorithms in cloud: based on task dependency', *International Journal of Computer Applications*, Vol. 82, No. 15, pp.20–26.

Bessai, K., Youcef, S., Oulamara, A., Godart, C. and Nurcan, S. (2012) 'Bi-criteria workflow tasks allocation and scheduling in cloud computing environments', *Fifth International Conference on Cloud Computing*, IEEE Computer Society, pp.638–645, DOI:10.1109/CLOUD.2012.83.

Chawla, Y. and Bhonsle, M. (2013) 'Dynamically optimized cost based task scheduling in cloud computing', *International Journal of Emerging Trend & Technology in Computer Science (IJETTCS)*, Vol. 2, No. 3, pp.38–42, DOI:10.1109/CSC.2012.15.

Gomathi, B. and Krishnasamy, K. (2013) 'Task scheduling algorithm based on hybrid particle swarm optimization in cloud computing environment', *Journal of Theoretical and Applied Information Technology*, Vol. 55, No. 1, pp.33–38.

Huh, S., Yoo, J., Kim, M. and Hong, S. (2012) 'Providing fair share scheduling on multicore cloud servers via virtual runtime-based task migration algorithm', *32nd IEEE International Conference on Distributed Computing Systems*, pp.606–614, DOI:10.1109/ICDCS.2012.33.

Ingole, A., Chavan, S. and Pawde, U. (2011) 'An optimized algorithm for task scheduling based on activity based costing in cloud computing', *International Journal of Computer Applications® (IJCA)*, *2nd National Conference on Information and Communication Technology (NCICT)*, pp.34–37.

Jiang, W.Z. and Sheng, Z.Q. (2012) 'A new task scheduling algorithm in hybrid cloud environment', *International Conference on Cloud Computing and Service Computing*, IEEE Computer Society, pp.45–49.

Kashani, M.H. and Jahanshahi, M. (2009) 'Using simulated annealing for task scheduling in distributed systems', *International Conference on Computational Intelligence, Modelling and Simulation*, IEEE Computer Society, pp.265–269, DOI:10.1109/CSSim.2009.39.

Katyal, M. and Mishra, A. (2013) 'A comparative study of load balancing algorithms in cloud computing environment', *International Journal of Distributed and Cloud Computing*, Vol. 1, No. 2, pp.5-14.

Kaur, S. and Verma, A. (2012) 'An efficient approach to genetic algorithm for task scheduling in cloud computing environment', *International Journal of Information Technology and Computer Science (IJITCS)*, Vol. 4, No. 10, pp.74–79.

Liu, J., Luo, X.G., Zhang, X.M., Zhang, F. and Li, B.N. (2013) 'Job scheduling model for cloud computing based on multi-objective genetic algorithm', *International Journal of Computer Science Issues (IJCSI)*, Vol. 10, No. 1, pp.134–139.

Ma, L., Lu, Y., Zhang, F. and Sun, S. (2013) 'Dynamic task scheduling in cloud computing based on greedy strategy', in *Trustworthy Computing and Services*, pp.156–162, Springer, Berlin Heidelberg.

Malik, S., Huet, F. and Caromel, D. (2012) 'Latency based dynamic grouping aware cloud scheduling', *26th International Conference on Advanced Information Networking and Applications Workshops*, IEEE Computer Society, pp.1190–1195, DOI 10.1109/WAINA. 2012.91.

Mathew, T., Sekaran, K.C. and Jose, J. (2014) 'Study and analysis of various task scheduling algorithms in the cloud computing environment', *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, pp.658–664.

Mell, P. and Grance, T. (2011) *The NIST Definition of Cloud Computing*, Recommendations of the National Institute of Standards and Technology, NIST Special Publication 800-145 [online] http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

Mesbahi, M. and Rahmani, A.M. (2016) 'Load balancing in cloud computing: a state of the art survey', *International Journal of Modern Education and Computer Science*, Vol. 8, No. 3, pp.64–78.

Navimipour, N.J., Rahmani, A.M., Habibizad-Navin, A. and Hosseinzadeh, M. (2014) 'Job scheduling in the expert cloud based on genetic algorithms', *Kybernetes*, Vol. 43, No. 8, pp.1262–1275.

Omidi, A. and Rahmani, A.M. (2009) 'Multiprocessor independent tasks scheduling using a novel heuristic PSO algorithm', *2nd IEEE International Conference, Computer Science and Information Technology*, IEEE, pp.369–373.

Rahmani, A.M. and Rezvani, M. (2009) 'A novel genetic algorithm for static task scheduling in distributed systems', *International Journal of Computer Theory and Engineering*, Vol. 1, No. 1, pp.1793–8201, DOI:10.7763/IJCET.2009.

Santhosh, R. and Ravichandran, T. (2013) 'Pre-emptive scheduling of on-line real time services with task migration for cloud computing', *Proceedings of the 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering (PRIME)*, IEEE, pp.272–276.

Saxena, D. and Chauhan, R.K. (2014) 'A review on dynamic fair priority task scheduling algorithm in cloud computing', *International Journal of Science, Environment and technology*, Vol. 3, No. 3, pp.997–1003.

Selvarani, S. and Sadhasivam, G.S. (2010) 'Improved cost-based algorithm for task scheduling in cloud computing', *IEEE International Conference on Computational Intelligence and Computing Research* (*ICCIC*), pp.1–5, IEEE, DOI:10.1109/ICCIC.2010.570587

Wang, Q., Gao, Y. and Liu, P. (2006) 'Hill climbing-based decentralized job scheduling on computational grids', *Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences(IMSCCS'06)*, IEEE Computer Society, Vol. 1, pp.705–708.

Wu, X., Deng, M., Zhang, R., Zeng, B. and Zhou, S. (2013) 'A task scheduling algorithm based on QoS-driven in cloud computing', *International Conference on Information Technology and Quantitative Management (ITQM)*, Elsevier, pp.1162–1169.