Cloudlet Scheduling Based Load Balancing on Virtual Machines in Cloud Computing Environment

Aida A. Nasr[1], Nirmeen A. El-Bahnasawy[2], Gamal Attiya[2], Ayman El-Sayed[2]

[1] Faculty of Artificial Intelligence, Kafrelsheikh University, Egypt.
[2] Computer Science and Engineering Dept., Faculty of Electronic Engineering, Menoufia University, Egypt.
Aida.nasr2009@gmail.com, nermeen.abd@el-eng.menofia.edu.eg, gamal.attiya@yahoo.com, ayman.elsayed@el-eng.menofia.edu.eg

Abstract

Management of cloud computing resources is critical, especially when several cloudlets are submitted simultaneously to cloud computing. Therefore, it is very important to use high efficient cloudlet scheduling techniques to guarantee efficient utilization of computing resources. This paper presents a two-phase approach, called *SAAC*, for scheduling cloudlets onto Virtual Machines (*VMs*) of cloud computing environment to balance workload on the available *VMs* and minimize makespan (i.e., the completion time at the maximum loaded *VM*). In the first phase, the *SAAC* approach applies the Simulated Annealing (*SA*) to find a near optimal scheduling of the cloudlets. While, in the second phase, the *SAAC* approach improves the cloudlets distribution by applying the Ant Colony Optimization (*ACO*) considering the solution obtained by the *SA* as the initial solution. The *SAAC* approach overcomes the computational time complexity of the *ACO* algorithm and low solutions quality of the *SA*. The proposed approach is evaluated by using the CloudSim, and the results are compared with that obtained by the most recent algorithms in terms of schedule length, load balancing, and time complexity.

Keywords: Cloud computing, Cloudlet scheduling, Load balancing, Ant colony, Simulated annealing

1 Introduction

Cloud computing is a new type of shared infrastructure. It has huge pools of computing resources and allows end users to use these resources over the Internet on-demand by pay-per-use strategy [1]. Users and enterprises can use cloud-computing to obtain high processing power without bearing the elevated price of building huge data centers [2]. Providers, like amazon *EC2*, let their clients to access, assign, and manage a group of Virtual Machines (*VMs*) that run inside the data centers and only charge them for the period of using the machines [3].

These days, cloud computing is becoming an efficient computing model. It provides high-performance computing over the Internet. However, one of the key challenges that degrade cloud-computing performance is cloudlets scheduling. This problem characterizes by the presence of limited number of *VMs* on which several cloudlets need to be executed. Therefore, an efficient cloudlet scheduling technique is required to guarantee good schedule of the cloudlets onto the *VMs* in order to minimize makespan and enhance cloud performance.

Recently, many researchers have developed meta-heuristic algorithms to solve the scheduling problem. However, most of the existing algorithms ignore several constraints that may affect the scheduling process [4]. In other words, they ignore the requirements of different cloudlets (i.e., memory and processing load requirements) and the availability of cloud computing resources (i.e., the available memory and processing power of *VMs*) [5].

This paper presents a new two-phase approach for cloudlets scheduling in cloud computing environment to balance workload on the *VMs* and minimize makespan (i.e., schedule length). The proposed approach, called *SAAC*, combines Simulated Annealing (*SA*) and Ant Colony Optimization (*ACO*). It takes into consideration the requirements of different cloudlets and the availability of computing resources. The proposed approach first finds a near optimal scheduling of cloudlets by applying the *SA* and then improves the cloudlets distribution by applying the *ACO* considering the solution obtained by the *SA* as the initial solution. The proposed *SAAC* approach overcomes the computational time complexity of the *ACO* algorithm and low solutions quality that obtained by the *SA*.

The remainder of this paper is organized as follows. Section 2 introduces cloud computing and cloudlet scheduling problem. Section 3 illustrates the formulation of scheduling problem. Section 4 presents the existing scheduling techniques, while the proposed approach is developed in Section 5. Section 6 presents

*Corresponding Author: Aida A. Nasr; E-mail: Aida.nasr2009@gmail.com
DOI: 10.3966/160792642019092005005

the simulation results while Section 7 presents the conclusion of this research work.

2 Cloud Computing and Cloudlet Scheduling Problem

Cloud computing environment contains several components namely, client, cloud information system, datacenter broker, and Virtual Machines (*VMs*) [6], as shown in Figure 1. Client is responsible for submitting cloudlet(s) into cloud environment to be processed. All information of the submitted cloudlets is stored in cloud information system. This information includes cloudlet length, arrival time and resources requirements. The information system also monitors the availability of cloud-computing resources. Datacenter broker contains a scheduler, the backbone of scheduling process, which is responsible for assigning the cloudlets onto the *VMs*. It determines the execution order of each cloudlet. The *VMs* are the main components in cloud computing environment that responsible for execution of cloudlets.

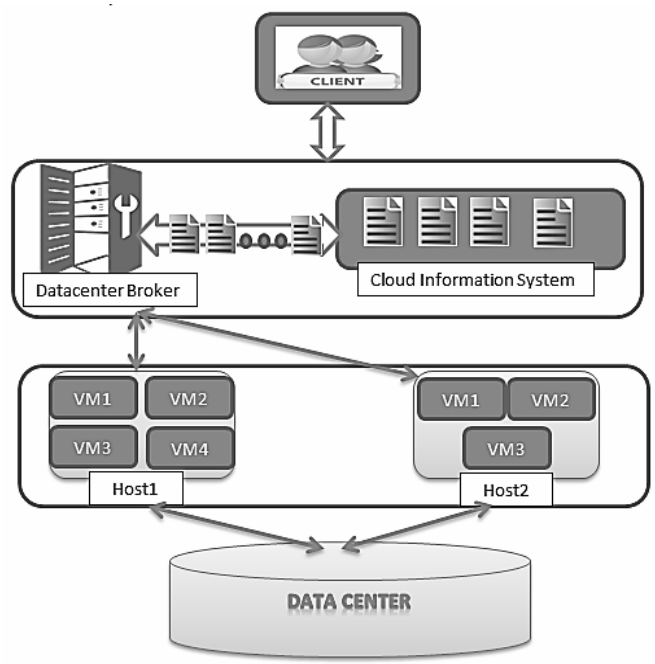

Figure 1. Components of cloud computing environment

In cloud computing environment, several cloudlets arrive to the system at the same time. Each cloudlet needs to be assigned into a suitable *VM* to be executed in a shortest time. However, because the number of available *VMs* is less than the number of submitted cloudlets, the problem is how to find a good schedule of the cloudlets onto the *VMs* to minimize the execution cost or makespan. This problem is called cloudlet scheduling problem. Briefly, a set of *n* cloudlets needs to be processed on *m VMs* in the cloud-computing environment. The cloudlets require certain resources while the *VMs* have limited resources as memory and processing. Thus, the purpose is to schedule the cloudlets onto the *VMs* such that the completion time is minimized, the requirements of cloudlets are met, and the availability of the *VMs* resources is not violated.

3 Problem Formulation

The cloudlet-scheduling problem may be formulated as an optimization problem to be solved by optimization approaches. Designing a mathematical model to the cloudlet scheduling problem involves two steps; (i) formulate a cost function to represent the objective of the cloudlet scheduling, (ii) formulate set of constraints in terms of the cloudlets requirements and the availability of the *VMs* resources.

To formulate the scheduling problem, let *n* be the number of cloudlets, *m* be the number of *VMs*, e_{iv} be the processing time of cloudlet *i* on machine *v*, and X_{iv} is a binary variable such that X_{iv} is 1 if the cloudlet *i* is assigned to VM *v* and 0 otherwise, as shown in Eq. (1).

$$x_{iv} = \begin{cases} 1, & if\ cloudlet\ i\ assigned\ to\ machine\ v \\ 0, & Otherwise \end{cases} \quad \textbf{(1)}$$

Therefore, the scheduling problem may be formulated mathematically as:

$$\min Z = \sum_{i=1}^{n}\sum_{v=1}^{m} e_{iv} x_{iv} \quad \textbf{(2)}$$

Subject to

$$\sum_{v=1}^{m} x_{iv} = 1, \qquad \forall cloudlet\ i \quad \textbf{(3)}$$

$$\sum_{i=1}^{n} MI_i\, x_{iv} \le L_v \qquad \forall VM \quad \textbf{(4)}$$

$$\sum_{i=1}^{n} mem_i\, x_{iv} \le Mem_v \qquad \forall VM \quad \textbf{(5)}$$

In this model, the objective of scheduling is formulated, Eq. (2), to minimize the execution time. Indeed, several constraints are modeled to meet the requirements of cloudlets and not violate the availability of *VM* resources. The first constrain, Eq. (3), guarantees that each cloudlet *i* is assigned to exactly one virtual machine *v*. The second constraint, Eq. (4), guarantees that the total processing load of all cloudlets, in Million Instructions (*MI*), assigned to a virtual machine v doesn't exceed the available processing (L_v) of that *VM*. The third constraint, Eq. (5), guarantees that the total memory requirements of all cloudlets scheduled into a virtual machine *v* should not

exceed the maximum available memory of that *VM*. Where, mem_i is the memory requirements of cloudlet *i,* and Mem_v is the available memory of virtual machine v.

After formulating the scheduling problem, the solution may be obtained by solving the formulated problem. The optimal solution to the formulated problem may be achieved by applying an optimization method such as exhaustive search or branch-and-bound. However, the problems of applying such methods are that they provide very high time complexity and it is very difficult to be used in case of large number of cloudlets. Therefore, most of the research works trend to use heuristic methods which achieve near optimal solution.

The optimal solution (S_{opt}) may be defined as the best solution that achieves the lowest *makespan*. According to [7], the system can achieve the lowest *makespan* (i.e. optimal solution) if and only if the next conditions are met:

- Each cloudlet is assigned to distinct *VM.*
- Each cloudlet starts execution as soon as possible.

4 Related Work

Recently, several cloudlet-scheduling techniques are developed to solve the scheduling problem. Two types of algorithms are developed; heuristic algorithms and meta-heuristic algorithms. Heuristic algorithms use the predictions to achieve a near optimal solution [8]. On the other hand, the meta-heuristic methods search the solution space in a direct manner and produce efficient results on the broad domain problems, but these methods have high time complexity [9]. Genetic Algorithm (*GA*) and Ant Colony Optimization (*ACO*) are the famous algorithms. In [10], the authors use *GA* with *ACO* to develop a two-phase algorithm for the scheduling process. In the first phase, the authors apply one *GA* generation and select the best solution. In the second phase, they apply the *ACO* algorithm, using the *GA* solution of as initial solution, to obtain near optimal solution. Another metaheuristic cloudlet-scheduling algorithm is Simulated Annealing (*SA)* [11]. In [12], the authors developed a new cloudlet-scheduling algorithm for cloud computing based on *SA* and greedy. The algorithm uses greedy strategy as initial stage to get a near optimal solution and then improve the solution by applying the *SA*. Other meta-heuristic algorithm developed to solve the scheduling issue is the Particle Swarm Optimization (*PSO*) [13].

5 Proposed Approach

This section presents a two-phase approach for cloudlet scheduling in cloud computing to balance workload on the available *VMs* and achieve minimum schedule length. The first phase of the proposed algorithm finds a near optimal solution rapidly by applying the well-known Simulated Annealing (*SA*) while the second phase improves cloudlets allocation by applying the Ant Colony Optimization (*ACO*). The developed approach exploits the low time complexity of *SA* and the quality of solution of the *ACO* algorithm. In addition, the approach overcomes the drawback of the *ACO* that occurs in the first stage due to absence of pheromone by generating an initial solution using the *SA* algorithm.

5.1 Simulated Annealing

Simulated Annealing (*SA*) is a global optimization technique that attempts to find lowest point in an energy landscape [14]. The main idea of this method is derived from cooling molten metal slowly to generate a regular crystalline structure. The distinctive feature is that it starts with an initial solution, at high temperature, and incorporates random jumps to potential new solutions. At each temperature level, a new solution is generated and the quality of this solution is tested. The algorithm accepts the new solution as the current solution if it optimizes the objective function. Figure 2 shows sample of *SA* solution space of 500 cloudlets at 4*VMs*. From the figure, solution *B* is generated after current solution *A* and the algorithm accepts solution B as it improves quality of solution. In *SA*, to achieve good solution, initial temperature should be high. In this paper, the *SA* is developed to generate a good solution at low running time. The algorithm starts at temperature *Temp=1000* and then searches for good solution. The algorithm repeats the searching process until the quality of a solution does not change 10 consecutive solutions. Algorithm 1 presents the *SA* algorithm steps.

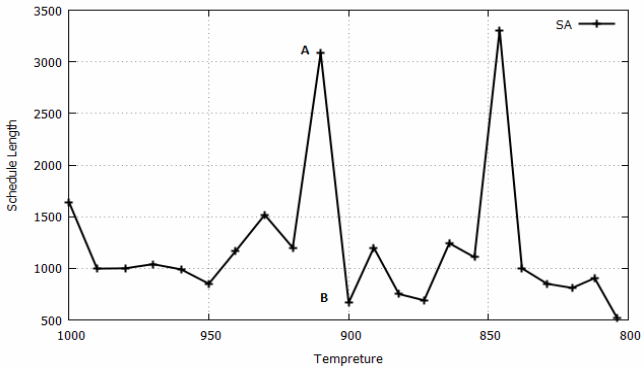

Figure 2. Sample of SA solution space of 500 cloudlets and 4 *VMs*

Algorithm 1. Simulated Annealing (SA) Steps

1. Generate Current solution randomly
2. Set No_rejects=0
3. Set good solution= Current solution
4. Calculate E_{good} and E_s
5. Initialize SA parameters; Temp and cooling rate $\alpha \in [0\text{-}1]$
6. Generate new solution

7. Calculate E_{new}
8. Calculate $\Delta = E_{new} - E_s$
9. **If** $\Delta < 0$
10. $E_s=E_{new}$ and Current Solution=New Solution
11. **Else**
12. Generate a random value r $\in$ [0-1]
13. **If** (r < exp (- Δ/Temp))
14. No_rejects++
15. $E_s=E_{new}$ and Current Solution=New Solution
16. **End if**
17. **End if**
18. **If** ($E_{good}>E_s$)
19. $E_{good}=E_s$ and good Solution=Current Solution
20. **End If**
21. Set Temp=Temp – ($\alpha \times Temp$)
22. **If** (Temp >1 and No_rejects<10)
23. Go to step (6)
24. **Else**
25. Return SA solution=good Solution
26. **End if**

5.2 Ant Colony Optimization

The Ant Colony Optimization (*ACO*) algorithm takes its characteristics from the ability of ants to find the shortest path between the food and their nest. The ants connect by laying trails of pheromone. Ants choose their path by computing probability *P*. Where, *P* depends on pheromone trails on the ground. When the pheromone of any path increases, the number of ants that choose this path will increase. Thus, each new solution is better than the previous [15]. Algorithm 2 illustrates the steps of *ACO* algorithm.

Algorithm 2. An Iteration of the ACO Algorithm

1. Assign y ants on m VMs according to the initial SA solution
2. Initialize τ_{iv} for each path between $cloudlet_i$ and VM_v
3. **While** cloudlet-list is not empty repeat
4. **For** k=0 to y
5. Ant_k selects a suitable VM_v for the selected $cloudlet_i$ according to P_{iv}^k (T)
6. Insert VM_v in $Tabu_k$ and remove it from $allowed_k$
7. Remove the selected_cloudlet from cloudlet_list
8. Update local pheromone
9. **End For**
10. **End while**
11. Update global pheromone

5.2.1 Initialization of Pheromone

When a cloudlet *i* assigned to a virtual machine *v*, a new path is created with pheromone trial τ_{iv}. By using the *SA* solution, the algorithm assigns each ant on a specific *VM* according to the same order of initial solution [18]. Then, it initializes the pheromone trial for each edge according to Eq. (6):

$$\tau_{iv}^k(0) = \tau_c + \frac{MIPS(\mathrm{VM}_v)}{task_i\_length} \quad \textbf{(6)}$$

Where, τ_{iv}^k (0) is the pheromone trial value at initial iteration *t=0* for ant *k*, and τ_c is constant.

5.2.2 Virtual Machine Selection for Next Cloudlet

Each ant applies the next probability *P* in Eq. 7 to choose VM_v for next cloudlet *i*.

$$\begin{cases} P_{iv}^k(t) = \dfrac{\tau_{iv}(t) \times \eta_{iv}}{\sum_{s \in allowed_k} \tau_{is}(t) \times \eta_{is}} & if\ v \in allowed_k \\ 0 & Otherwise \end{cases} \quad \textbf{(7)}$$

Where, τ_{iv} is the pheromone trial of cloudlet *i* in VM_v, $allowed_k$ is the available *VMs* of ant *k* that are not chosen yet for any cloudlet by the ant. The *VMs* that are chosen are stored in $tabue_k$. $\eta_{iv} = 1/d_{iv}$ is heuristic information representing the visibility of ant *k* at iteration *t*, where d_{iv} is the expected execution time of cloudlet *i* at VM_v.

5.2.3 Pheromone Updating

After each ant creates a path, it updates the local pheromone of this path by Eq. 8.

$$\begin{cases} \Delta \tau_{iv}^k(\mathrm{t}) = \dfrac{Q}{L^k(\mathrm{t})} & if\ (i,v) \in T^k(t) \\ 0 & Otherwise \end{cases} \quad \textbf{(8)}$$

Where, $T^k(t)$ represents $Tabu_k$ (the collection of *VMs* that ant *k* visited) at iteration *t*, $L^k(t)$ is the expected makespan of ant *k* and *Q* is a control parameter. After generating a new solution, the global pheromone updates by the Eq. 9.

$$\tau_{iv}(\mathrm{t+1}) = (1-\rho)\tau_{iv}(t) + \Delta\tau_{iv}(t) \quad \textbf{(9)}$$

Where, $\rho \in$ [0-1] is the trial volatility coefficient, and $\Delta\tau_{iv}(\mathrm{t})$ is computed by $\Delta\tau_{iv}(\mathrm{t}) = \sum_{k=0}^{n} \Delta\tau_{iv}^k$.

5.3 Two-Phase *SAAC* Approach

Algorithm 3 illustrates the developed *SAAC* approach. The approach combines both the well-known Simulated Annealing (*SA*) and the Ant Colony Optimization (*ACO*). In the first phase, the *SA* is applied to generate an initial solution, while, in the second phase, the *ACO* is applied to find the best schedule of cloudlets on the available *VMs*. The developed *SAAC* approach solves the tradeoff between

minimizing schedule length and time complexity by passing the *SA* solution to the *ACO* algorithm. Where, the *ACO* algorithm enhances the *SA* solution until it achieves the optimal solution.

Algorithm 3. Two-Phase SAAC Approach

1. Set No_rejects=0
2. Set Current solution=SA solution and Best solution=SA solution
3. **For** t=0 to t_{Max} do
4. Generate new solution using ACO algorithm
5. **If** new Solution quality is less than the current solution quality
6. No_rejects++
7. **If** (No_rejects>4)
8. End the algorithm and Return Best Solution
9. **End If**
10. **Else**
11. Best solution=new solution
12. **End If**
13. Current solution = new solution
14. **End For**

From Algorithm 3, the *SAAC* starts with setting *No_rejects* equal to zero. This value is used to prevent repeating the same solution and monitors the quality of searching. If *No_rejects* reaches to 4 this means that the quality is decreased and the solution is bad. At this moment, the *SAAC* approach will stop and gives the best solution.

5.4 Time Complexity

The time complexity of *SA* depends on two factors; the number of iterations (*itr*) and the product ($n \times m$). It has time complexity $O(itr \times n \times m)$, where *itr* is often *>1000* for the initialize temperature=2500 and cooling rate=0.01. The time complexity of the *ACO* is the summation of steps for generate number of solutions $O(t_{max} \times n \times m)$, and time complexity for evaluating the solutions $O(t_{max} \times n)$. So, the overall time complexity of the *ACO* is $O(t_{max} \times n \times m + t_{max} \times n$. Where, t_{max} is the maximum number of generations. The time complexity of *SAAC* is the summation of the time complexity of both the *SA* and the *ACO*. The *SA* has time complexity $O(itr \times n \times m)$, while the time complexity of the *ACO* is $O(t_{max} \times n \times m)$. Thus, the overall time complexity of the new *SAAC* algorithm is $O(itr \times n \times m + t_{max} \times n \times m)$. The *SAAC* approach uses *No_rejects* variable to decrease t_{max} *(i.e. from the experiments* $t_{max} < 30 << n$*)*. So, the time complexity for *SAAC* approach may be recomputed as $O(itr \times n \times m)$. Since, *itr* for *SAAC* algorithm is less than *itr* for *SA* algorithm *(i.e. itr<200)*. Therefore, time complexity of *SAAC* approach is less than time complexity of *SA* algorithm.

6 Simulation Results

This section presents performance evaluation of the proposed two-phase *SAAC* scheduling approach. In this evaluation, the well-known *CloudSim* is used to simulate the cloud computing [16]. The simulation environment is a 64-bit windows 7 operating system installed in a laptop core i5 with 8 GB *RAM*. In addition, list of cloudlets are generated with lengths from 1000 to 10,000 MI, and a list of *VMs* is generated with $MIPS \in$ [100-1000]. The initial values of the *SAAC* approach parameters are $y=50$, $\rho=0.5$, $\tau_c=0.3$, $Temp=1000$, $\alpha=0.01$ and t_{max}=150. The results of the proposed *SAAC* approach are compared with both the *SA* and the *ACO* in terms of schedule length, load balancing and time complexity.

6.1 Schedule Length

Schedule length (*SL*) or makespan is the elapsed execution time at the maximum loaded virtual machine. Table 1 shows a simple comparison between the proposed *SAAC* approach and both the *ACO* and the *SA* algorithms in terms of makespan. From the table, the proposed *SAAC* approach is more efficient than the *ACO* and *SA* algorithms for all test cases. The new *SAAC* approach has lower schedule length at smaller number of *VMs* and larger number of cloudlets.

Table 1. Schedule length of *SAAC*, *ACO* and *SA* at different numbers of cloudlets and *VMs*

Conditions		Makespan in seconds		
No. of Cloudlets	No. of *VMs*	*SAAC*	ACO	SA
500	2	**2518**	3036	2687
	4	**1960**	1990	2000
	8	**1111**	1118	2125
1000	2	**4335**	5946	5941
	4	**2217**	2321	4768
	8	**1213**	1218	5768
1500	2	**9281**	10508	10599
	4	**3604**	3650	7077
	8	**2540**	2553	9491
2000	2	**11379**	11931	21039
	4	**6013**	6020	8993
	8	**2841**	2850	12014
2500	2	**13379**	13431	11672
	4	**6937**	7000	12537
	8	**3055**	3070	6651
3000	2	**18123**	21142	11937
	4	**8369**	8878	24194
	8	**3243**	3350	13597
3500	2	**20105**	25200	20596
	4	**9937**	10950	38299
	8	**4007**	4115	15560
4000	2	**43526**	55433	40985
	4	**11012**	12025	45472
	8	**4370**	4390	18616

From Table 1, both the *SA* and the *ACO* algorithms provide high schedule length with large number of cloudlets. In some cases (at 4000 cloudlets and 2 *VMs*), the new *SAAC* approach can save more than **2 hours** than both the *SA* and the *ACO* algorithms. The *SAAC* approach minimizes the schedule length by **47.24%** with *SA* and **14.5%** with *ACO*. This is because the *SAAC* approach uses the advantages of *SA* and *ACO* algorithms with additional modifications to minimize the schedule length.

6.2 Balancing Degree

Balancing Degree (*BD*) is the degree of balancing the workload on the available *VMs* after scheduling. The balancing degree may be calculated as follows.

$$BD = SL(S_{opt}) / SL(S_{fin}) \quad \textbf{(10)}$$

Where, S_{opt} is the optimal solution and S_{fin} is the final solution obtained from the applied algorithm. In this study, S_{opt} is assumed as the ideal solution, where S_{opt} is computed as the summation of total *MI* of all the cloudlets over the total *MIPS* of the available *VMs*. That is, $S_{opt} = Toal_{MI}/Total_{MIPS}$. From Eq. (10), the algorithm with high balancing degree achieves near optimal solution.

Figure 3, Figure 4 and Figure 5 show the balancing degree of scheduling different cloudlets by three different algorithms at 2, 4, and 8 *VMs* respectively. From the figures, the *SAAC* achieves higher *BD* ratio than *SA* and *ACO* algorithms, because it achieves the lowest makespan. Let's take Figure 3 as an example. From the figure we note that the new algorithm always has higher *BD* ratio than the others. Although, the new algorithm gives *BD* ratios near of *ACO* in Figure 4, and Figure 5, it achieves the near optimal solution at very little time than the *ACO*. This is shown in Figure 6, Figure 7, and Figure 8.The developed *SAAC* improves *BD* ratio with **45.9%** than *SA* and **4.8%** than *ACO* algorithms.

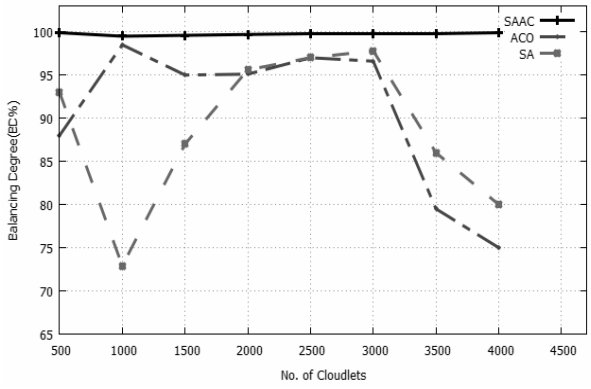

Figure 3. *BD* by different algorithms at 2*VM*

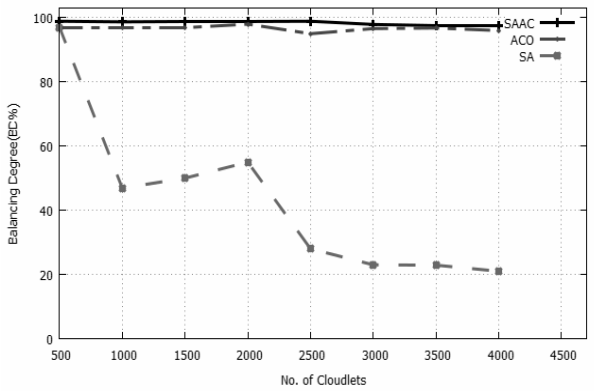

Figure 4. *BD* by different algorithms at 4*VM*

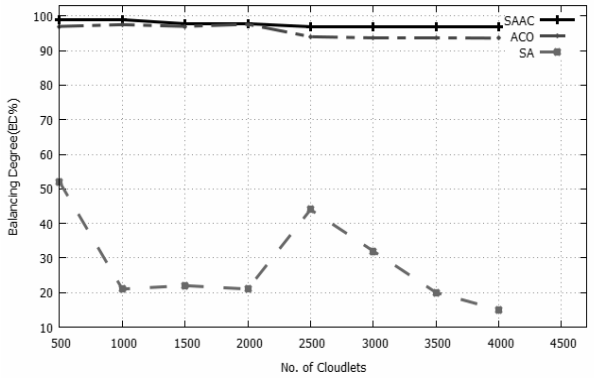

Figure 5. *BD* by different algorithms at 8*VM*

6.3 Computation Time Complexity

Figure 6, Figure 7 and Figure 8 show the computation time complexity of the *SAAC,* the *SA* and the *ACO* at different values of *VMs*; 2, 4, and 8 respectively. The figures show that the *SAAC* approach has lower time complexity than both the *SA* and the *ACO.* The *SAAC* approach combines both the *SA* and the *ACO* algorithms to improve the overall performance of cloud computing. The new approach avoids the high time complexity by using a new variable called *No_rejects*, which counts the number of rejected solutions, and stops the repetition when reaching to the specific value. This prevents generating new bad solutions more times for minimizing time complexity.

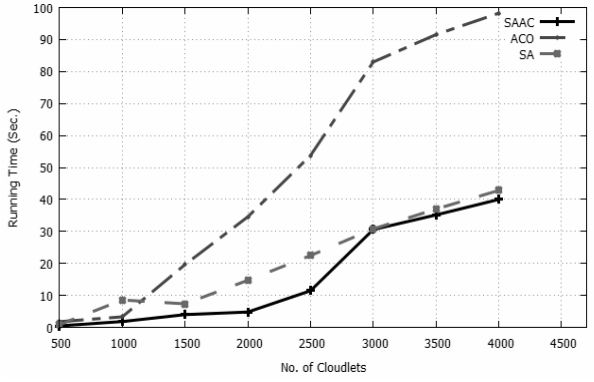

Figure 6. Running time of different algorithms at 2*VM*

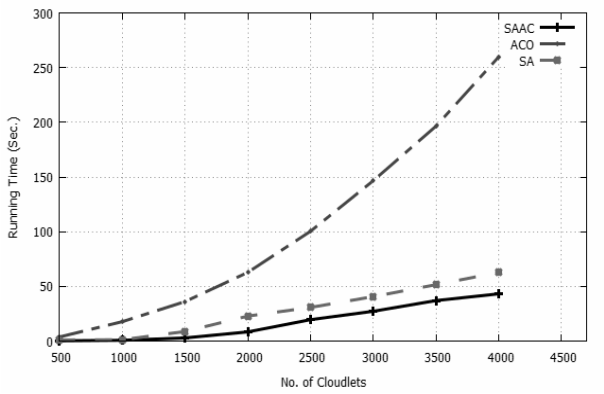

Figure 7. Running time of different algorithms at 4*VM*

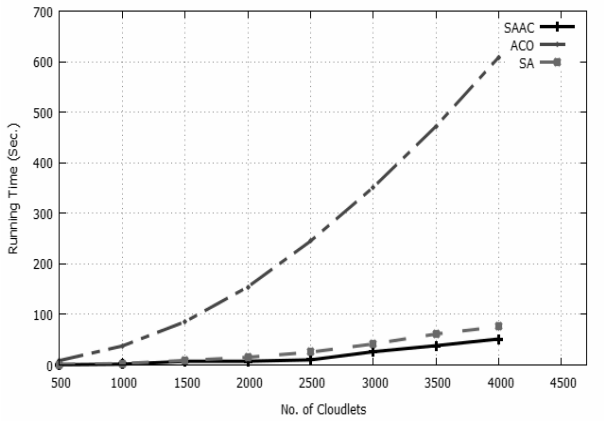

Figure 8. Running time of different algorithms at 8*VM*

From Figure 6, Figure 7 and Figure 8, the *SAAC* has running time less than the *SA* and the *ACO* algorithms. For example, in Figure 6, the *SAAC* approach takes **40** second to find the near optimal solution, whereas the *ACO* takes **100** second, and *SA* takes 45 second. In addition, from Figure 8, the *SAAC* approach takes less than **100** second to find the near optimal solution of **4000** cloudlets, whereas the *ACO* algorithm gets a solution at **600** second. On other words, the *SAAC* approach is better than the *SA* and *ACO* in terms of running time. It decreases the running time **20.5%** than *SA* and **87.12%** than *ACO* algorithms.

In summary, from the figures, the new *SAAC* is better than both the *SA* and the *ACO* algorithms in terms of schedule length, balancing degree, and running time.

6.4 Real Experiments

To insure that the new approach is more efficient than the other algorithms, the proposed *SAAC* is applied to schedule a list of real cloudlets on different numbers of *VMs*. The cloudlets are generated by using a standard formatted workload of a high-performance computing center, called *HPC2N,* in Sweden as a benchmark [17]. Table 2 shows the makespan obtained by 4 different algorithms.

Table 2. Experimental results of different algorithms

Conditions		Makespan in seconds				
No. of Cloudlets	No. *VMs*	SAAC	ACO	SA	PSO	Hybrid
1000	10	1980	2800	2783	2700	1983
	20	840	1254	1440	1497	854
	30	483	1345	1224	1505	531
3600	10	4614	4912	5678	4987	4616
	20	2192	2978	3728	3018	2194
	30	1651	2598	4395	2678	1649
6500	10	11135	21800	22343	22800	11137
	20	5416	10800	13075	11800	5420
	30	3492	6987	8280	8019	3500
8500	10	17886	19599	23284	23000	17889
	20	13513	14200	16857	15300	13510
	30	5609	7000	8525	6800	5609

From Table 2, the new *SAAC* algorithm achieves the lowest schedule length in all the test cases. It can distribute a large number of cloudlets into different numbers of *VMs* and achieves better schedule length than the current *SA, ACO, PSO,* and the *Hybrid* algorithm [18].

7 Conclusion

In this paper, a new two-phase *SAAC* approach is proposed for cloudlet scheduling in cloud computing environment. The proposed *SAAC* approach can schedule a large number of cloudlets into the available *VMs* considering several resources constraints, at low time complexity. It keeps the *VMs* in high balancing degree, and enhances the overall system performance. By comparing the new *SAAC* approach with the *SA* and the *ACO*, the new approach is more efficient than those algorithms. The experimental results show that the *SAAC* approach decreases running time by about 32.5% than SA and 87.12% than *ACO* algorithm. In addition, it achieves lower schedule length than *SA* and *ACO* algorithms. It decreases the schedule length by about 47.24% with *SA* and 14.5% with *ACO*. The *SAAC* also provides high load balancing degree. It improves balancing degree ratio by 45.9% than *SA* and 4.8% than *ACO* algorithms. Moreover, the new *SAAC* algorithm is compared with the *PSO* and the *Hybrid* algorithms.

References

[1] O. Terzo, L. Mossucca, *Cloud Computing with e-Science Applications*, Crc Press, 2015.

[2] C. Changchit, C. Chuchuen, Cloud Computing: An Examination of Factors Impacting Users' Adoption, *Journal of Computer Information Systems*, Vol. 58, No.1, pp. 1-9, September, 2018.

[3] B. Varghese, R. Buyya, Next Generation Cloud Computing: New Trends and Research Directions*, Future Generation*

Computer Systems, Vol. 79, pp. 849-861, February, 2018.

[4] R. K. Jena, Task Scheduling in Cloud Environment: A Multi-objective ABC Framework, *Journal of Information and Optimization Sciences*, Vol. 38, No. 1, pp. 1-19, February, 2017.

[5] A. A. Nasr, S. A. Elbooz, Scheduling Strategies in Cloud Computing: Methods and Implementations, *American Journal of Engineering and Applied Sciences*, Vol. 11, No. 2, pp. 426-432, April, 2018.

[6] M. Kumar, A. K. Yadav, P. Khatri, R. S. Raw, Global Host Allocation Policy for Virtual Machine in Cloud Computing, *International Journal of Information Technology*, Vol. 10, No. 3, pp. 279-287, September, 2018.

[7] O. Sinnen, *Task Scheduling for Parallel Systems,* John Wiley & Sons, 2007.

[8] S. H. H. Madni, M. S. A. Latiff, M. Abdullahi, S. M. Abdulhamid, M. J. Usman, Performance Comparison of Heuristic Algorithms for Task Scheduling in IaaS Cloud Computing Environment, *PloS one*, Vol. 12, No. 5, e0176321, May, 2017.

[9] F. Villa, E. Vallada, L. Fanjul-Peyro, Heuristic Algorithms for the Unrelated Parallel Machine Scheduling Problem with One Scarce Additional Resource, *Expert Systems with Applications*, Vol. 93, pp. 28-38, March, 2018.

[10] C. Y. Liu, C. M. Zou, P. Wu, A Task Scheduling Algorithm Based on Genetic Algorithm and Ant Colony Optimization in Cloud Computing, *Proceedings of the 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*, Xian Ning, China, 2014, pp. 68-72.

[11] F. Fanian, V. K. Bardsiri, M. Shokouhifar, A New Task Scheduling Algorithm Using Firefly and Simulated Annealing Algorithms in Cloud Computing, *International Journal of Advanced Computer Science and Applications*, Vol. 9, No. 2, pp. 195-202, March, 2018.

[12] X. Liu, J. Liu, A Task Scheduling Based on Simulated Annealing Algorithm in Cloud Computing, *International Journal of Hybrid Information Technology*, Vol. 9, No. 6, pp. 403-412, June, 2016.

[13] A. I. Awad, N. A. El-Hefnawy, H. M. Abdel_kader, Enhanced Particle Swarm Optimization for Task Scheduling in Cloud Computing Environments, *Procedia Computer Science*, Vol. 65, pp. 920-929, 2015.

[14] A. Assad, K. Deep, A Hybrid Harmony Search and Simulated Annealing Algorithm for Continuous Optimization, *Information Sciences*, Vol. 450, pp. 246-266, June, 2018.

[15] Y. J. Moon, H. C. Yu, J. M. Gil, J. B. Lim, A Slave Ants Based Ant Colony Optimization Algorithm for Task Scheduling in Cloud Computing Environments, *Human-centric Computing and Information Sciences*, Vol. 7, No. 1, pp. 1-10, December, 2017.

[16] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, R. Buyya, CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, *Software: Practice and experience*, Vol. 41, No. 1, pp. 23-50, January, 2011.

[17] D. G. Feitelson, D. Tsafrir, D. Krakov, Experience with Using the Parallel Workloads Archive, *Journal of Parallel and Distributed Computing*, Vol. 74, No. 10, pp. 2967-2982, October, 2014.

[18] H. Idris, A. E. Ezugwu, S. B. Junaidu, A. O. Adewumi, An Improved Ant Colony Optimization Algorithm with Fault Tolerance for Job Scheduling in Grid Computing Systems, *PloS one*, Vol. 12, No. 5, e0177567, May, 2017.

Biographies

Aida A. Nasr received a Ph.D. in 2019, in the Dept. of Computer Science and Engineering, Faculty of Electronic Engineering, Menoufia University, Egypt. She is a Lecturer in the faculty of Artificial Intelligence, Kafrelsheikh University, Egypt. She received M.Sc. and B.S. from the Faculty of electronic Engineering, Menoufia University, Egypt in 2015 and 2010 respectively.

Nirmeen A. El-Bahnasawy received her B.S. in Electronic Engineering in 1998 and M.Sc. and Ph.D. degrees in Computer Science and Engineering from Menofia University in 2003 and 2013, respectively. Currently, she has been appointed as an Associate Professor at Menofia University in 2019. Her research interests include distributed computing, grid computing, and Cloud computing.

Gamal Attiya graduated in 1993 and obtained his M.Sc. degree in computer science and engineering from the Menoufia University, Egypt, in 1999. He received Ph.D. degree in computer engineering from the University of Marne-La-Vallée, Paris-France, in 2004. He is currently full Professor at the Department of Computer Science and Engineering, Faculty of Electronic Engineering, Menoufia University, Egypt.

Ayman El-Sayed received the B.Sc. (1994) and M.Sc. (2000) in computer science and engineering from the Menoufia University, Egypt. Ph.D. (2004) in computer network from Institute National De Polytechnique De Grenoble (INPG), France. He is full Professor/Dean of Faculty of Electronic Engineering, Menoufia University, Egypt. He is a IEEE senior member.