Adaptive Scheduling of Cloud Tasks Using Ant Colony Optimization

Sambit Kumar Mishra, Bibhudatta Sahoo and P. Satya Manikyam
National Institute of Technology, Rourkela, India
Email: {skmishra.nitrkl, bibhudatta.sahoo , psatya551}@gmail.com

ABSTRACT
Efficient scheduling of heterogeneous tasks to heterogeneous processors for any application is crucial to attain high performance. Cloud computing provides a heterogeneous environment to perform various operations. The scheduling of user requests (tasks) in the cloud environment is a NP-hard optimization problem. Researchers present various heuristic and metaheuristic techniques to provide the sub-optimal solution to the problem. In this paper, we have proposed an Ant Colony Optimization (ACO) based task scheduling (ACOTS) algorithm to optimize the makespan of the system and reducing the average waiting time. The designed algorithm is implemented and simulated in CloudSim simulator. Results of simulations are compared to Round Robin and Random algorithms which show satisfactory output.

Keywords
Ant Colony Optimization (ACO); cloud computing; Makespan; task scheduling; VM.

1. INTRODUCTION

The term “Cloud Computing" refers to the provision of computing resources as a service. All the physical servers are connected to the Internet. These servers are located remotely, means the servers are remote from the location from where the services requested. The cloud computing systems are built from some virtualized data centers and a finite number of physical server (hosts) contributes a data center. A large number of servers required to provide services to huge number of cloud users. The cloud service provider (CSP) manages all the cloud resources to provide services and maintain certain level of quality of services (QoS). The cloud users submit their task to the CSP and an agreement had been done among them. This agreement is popularly known as service level agreement (SLA). The CSP provide services as per the SLA. The violation of SLA makes a difference in the profit. These QoS and SLA can handle with optimal allocation of resources.

The allocation of virtual resources (CPU, main memory, disc, bandwidth, etc.) play a significant role in reducing the energy consumption, optimizing makespan, reducing waiting time and increasing throughput of the system [1, 2, 3]. In cloud platforms, scheduling takes place at two levels. First, when the tasks are uploaded to the cloud, the scheduler assigns the requested tasks to different virtual machines, attempting to reduce the completion time of multiple applications across virtual machines. Second, allocating the virtual machines to physical machines to balance the load or to reduce energy consumption etc. (for example, Amazon EC2 uses elastic load balancing (ELB) to control how incoming requests are handled). The arrival of tasks (workloads) may cause bursty or non-bursty traffic. Example of bursty traffics are bursty surges and Internet flash crowds, there by aggressively grouped together in small periods there by create spikes with high arrival rate. The presence of burstiness in request arrival rate can cause degradation of the performance of task scheduler.

We have used the Expected Time to Compute (ETC) model as task model. This ETC model represents the task heterogeneity as well as machine heterogeneity. The task allocation or scheduling problem is a well-known NP-complete problem. The NP-complete nature of the problem focuses tremendous interest for researchers to propose sub-optimal solutions. Researchers proposed some heuristic approaches and metaheuristic approaches to optimize different objectives [4]. We have proposed a heuristic algorithm based on Ant Colony Optimization (ACO) technique to optimize makespan and average waiting time of the system. The main idea behind this approach is to use feedback mechanism and try to imitate the style of nature ant colonies to search for good by connecting to each other with the help of pheromone laid on food path traveled. Makespan is defined as the time required to execute a finite set of tasks by the system. A good scheduler of tasks should be able to adapt its strategy for scheduling to the varying environment and type of tasks [3].

1.1 Basic Idea of Ant Colony Optimization

The Ant Colony Optimization (ACO) is initially proposed by Colorni et al. [5]. The main idea behind ACO is based on the behavior of ant food searching process [6]. Ants used to search for food randomly, and after finding their food ants

return to their ant colony by keeping pheromone trail on the path from food place to colony. Once they found the food path, next time instead of random traveling, they used to follow pheromone trail, by hoping that this trail leads to food. With the time, however, the trail starts to evaporate, which leads to lessen an attractive strength. If the ants take more time to travel through this path, the more time trail used to present. Simultaneously, the shorter path to food from source has more pheromone trail, since by the time being ants used to find this short path and goes through the short path. The density of pheromone on the longer path is less than, the shorter one. The pheromone evaporation eliminates the convergence of the local optimal solution. The effect of pheromone evaporation is very important when it is applying to real time problems [7].

In ant colony optimization system, every ant is a computation agent. It constructs the optimal solution to the problem iteratively. The solutions obtained at intermediate states are referred as solution states. In each generation every ant transit from one state 'i' to state 'j' moving towards a locally optimal solution. Thereby, in each iteration, every ant computes a set of feasible solutions to its present state by using probability and move it to one of the local optimal states. For every ant k, the probability of $P_{ij}^{k}$of moving to state j from state i generally depends on two parameters, coefficient of vaporization phenomenon or pheromone concentration denoted by τ_{ij} indicating the previous move and visibility of the move denoted by η_{ij} indicating the past worth of the move. These values keep on updating for the better optimal solution in each iteration. In every iteration, their value may increase or decrease based on their effect to get an optimal solution. In general, the probability of ant k moving from state i to state j is given by Eq. (1).

$$\boldsymbol{P_{ij}^{k}} = \frac{(\boldsymbol{\tau_{iz}^{\alpha}})(\boldsymbol{\eta_{iz}^{\beta}})}{\sum_{\boldsymbol{VM\_list} \in \boldsymbol{allowed_k}} (\boldsymbol{\tau_{iz}^{\alpha}})(\boldsymbol{\eta_{iz}^{\beta}})} \qquad (1)$$

- τ_{ij} denotes pheromone concentration for transition from state i to j.
- 0 ≤ α is a parameter to control the influence pheromone concentration.
- η_{ij} is the desirability of transition from state i to state j. It is calculated using priori information, i.e., it is computed by a priori knowledge, typically $1/d_{ij}$, where d is the distance).
- $\beta \geq 1$ is a parameter to control the influence of η_{ij}.
- *VM_list* ∈ $\text{allowed}_{k,}$ where *VM_list* is list of VMs allowed by ant k.

The overall result of this update is that when one ant found the path to food sourced from the colony, remaining ants most probably follow that path, and this positive feedback of previous ants which have already found the path eventually leads to all ants follow the shortest path.

Contributions: This work has the following contributions:

- Presented the study of task scheduling in cloud computing environment using Ant Colony Optimization.
- Proposed an ACO-based task scheduling algorithm (ACOTS) to increase the stability of the system, reduce the waiting time, and minimize the makespan of the system.
- Evaluate a comparison analysis among our algorithm with Random and Round-Robin (RR) algorithm.

The main aim of writing this paper is to propose an algorithm for minimizing the makespan by maximizing the utilization of CPU-time, and reducing the waiting time. The remaining of the paper is prepared as follows. Section 2 outlines an overview of some related work; Section 3 describes a brief idea about the cloud system model and represents the problem statement along with some constraints. Section 4 illustrated our proposed work to optimize the makespan and average waiting time in a heterogeneous computing environment. Section 5, explains about simulation results and effectiveness of our algorithm. Section 6, concludes the paper.

2. RELATED WORK

Multiple heuristic methods concerning the allocation of tasks as well as balancing of the tasks to optimize some parameters have proposed in the literature. Due to a significant amount of interactive data or information and have a deadline to execute services, usually, in most of the time, users could not get the service with acceptable QoS. After receiving the task from the user, the resources of the cloud data center are virtualized. Task execution time is more relevant to everyone, and for this, there should be a decent management of physical resources by an efficient mapping between the tasks and VMs [8]. The mapping requires a balancing of loads among VMs and minimizes the makespan of the system. Most of the existing task scheduling algorithms based on Ant Colony Optimization focused on to reduce the makespan [9, 10]. Some research work tries to reduce makespan as well as other defined objectives. Very little research work focuses on balancing the load, makespan of the system, and starvation of tasks simultaneously. ACO is an evaluation approach, and many modifications and improvements have done to the basic ACO with time. Various research works have been done for task scheduling in Grid and Cloud as follows.

Tawfeek et al. [9] tried to reduce the makespan of the data center through task scheduling. They constrained each ant to visit one VM once and designed heuristic function based on the transfer time and expected execution time of tasks

on different VMs. Wen et al. [11] came with the idea of combining ACO with PSO to improve the performance of the system. They proposed this to increase the convergence speed and eliminate the problem of falling in local optimal solution.

Table 1. Comparison of various ACO based scheduling algorithms

Referenced work	Improvement strategy	Performance metrics	Nature of tasks	Environment
[9]	Basic ACO	Makespan	Independent	Cloud
[10]	perform local search after every iteration	Makespan	Dependent	Cluster
[11]	combined ACO with PSO	Makespan	Dependent	Cloud
[12]	Instead of updating individual solution. It update single result set to improve the load balancing.	Load	Independent	Not mentioned
[13]	Balancing the load for OCCF considering Complex Network	Load	Independent	Not mentioned
[14]	By finding overloaded and under-loaded perform Load Balancing	Load, SLA, Energy Consumption	Independent	Cloud
[15]	Load of systems of hot spots are identified and shifted by ACO	Load	Independent	Cloud
[16]	Basic ACO	Load	Independent	Cloud
[17]	modification in pheromone updation strategy	Makespan	Independent	Grid
[18]	Adaptively changed the pheromone evaporation rate.	Makespan, Load	Independent	Grid
[19]	Standard deviation based pheromone updation	Makespan	Independent	Grid
[20]	Balancing the load of Virtual Machines	Makespan, Load	Independent	Cloud

Some researchers are worked to balance the load of VMs through task scheduling algorithm thereby improve the system performance in the cloud environment. Zhang et al. [12] proposed a load balancing algorithm for open cloud computing Federation. Li et al. [13] proposed a new Ant Colony Based Load Balancing algorithm to schedule independent task with the objective of minimizing the makespan and balancing the load among all virtual machines. Kumar et al. [14] used the trailing and foraging pheromones to find out under loaded and overloaded virtual machines. Gu and LU [15] designed a strategy for balancing the load dynamically based on ACO. In their strategy, they identified the VMs which having resources more than the threshold value is found and they named them as hot spots. The tasks on those overloaded machines are moved to nearest machines having less load (or under loaded machines). In [16], the tasks are scheduled to virtual machines on First Come First Serve basis. They have used the ACO technique to find out under loaded VMs.

As from Table 1, very less work concentrated on load balancing and reducing makespan simultaneously. Little research work is done for task scheduling using ACO for reducing makespan, optimizing waiting time and balancing the load in the cloud computing environment. In some of the research work, tasks have generated randomly and some have taken very fewer tasks to evaluate their algorithms. But to evaluate one algorithm properly, there is need to generate the task as per the real time task arrivals in the cloud computing system. By keeping all these facts, we design an adaptive task scheduler to reduce makespan, load balancing using ACO.

3. TASK SCHEDULING USING ANT COLONY OPTIMIZATION IN CLOUD

The cloud system linked with some data centers. These data centers constitute from a finite set of physical servers or hosts. With the help of hypervisor or virtual machine manager (VMM), some VMs are deployed in every hosts. The virtual cloud resources in the form of virtual machines execute all user requests. The CSP follow the specified SLAs during the delivery of services to cloud users. We have designed an Adaptive Task Scheduling (ATS) algorithm using ACO. The ATS is able to take care condition of bursty arrivals. The mathematical formulation of ATS is as follows. Decreasing the makespan and waiting time is primary objective and balancing load is secondary objective. We had followed the basic steps as [5], but we had incorporated our approaches to define appropriate parameters.

3.1 Representation of Problem

The problem is represented as two sets VM_list, task_list where the set VM_list represents the set of m number of VMs and task_list represents the set of n number of tasks. The mapping of n tasks to m VMs will affect the cloud system

performance. According to this mapping, various parameters are optimized. Here, mapping of tasks to VMs done by ants. Initially, all ants placed on the VMs randomly. During each iteration, every ant builds a new solution by moving from one virtual machine to another virtual machine for next task until all tasks allocated to virtual machines. These iterations indicated by 1 ≤ iter ≤ max_no_iter where max_no_iter is the maximum number of allowed iterations. The symbol table is listed in Table 2.

Table 2. Symbol table with description

Symbol	Description
n	Total number of input tasks
m	Total number of VMs
α	Influence pheromone concentration control parameter
β	Influence control parameter of η_{ij}.
η_{ij}	Desirability of transition from state i
ρ	Coefficient of vapour phenomenon
τ_{ij}	Pheromone concentration between i^{th} task and j^{th} VM
z	Number of ants
Q	Adaptive parameter in each
VM_list	The finite set of VMs
task_list	The finite set of tasks
max_no_iter	Maximum number of iteration

3.2 Constraint Satisfaction

We have used constraint satisfaction method to avoid the local optimal solutions and avoid infeasible solutions. There by minimizing the time of the task and VM mapping.

3.3 Pheromone Updating Rule

The main important thing of ant system is updation of pheromone as it influences the performance of the task scheduling. Pheromone updation means when pheromone evaporates on edges, then new pheromone will be deposited on the edges visited by ants. The pheromone value directly influences the solution quality built by all ants. Pheromone concentration between ith task and jth VM is shown by τ_{ij}. This value will be changed after every iteration as shown in Eq. (2).

$$\tau_{ij}(t) = (1 - \rho)\,\tau_{ij}(t) + \Delta\tau_{ij}(t) \qquad (2)$$

where ρ is coefficient of vapour phenomenon, 0 < ρ < 1 and $\Delta\tau_{ij}$(t) is computed by Eq. (3).

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \qquad (3)$$

where $\Delta\tau_{ij}^{k}(t)$ is the pheromone *concentration* between ith task and jth VM according to ant k.

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \dfrac{Q}{MS^{k}(t)}, & if\ (i,j) \in O^{k}(t) \\ 0, & if\ (i,j) \notin O^{k}(t) \end{cases}$$

where tour traveled by ant k is given by $O^k(t)$ in iteration t, $MS^k(t)$ is cost or length of path, and Q is the adaptive parameter. When it comes to scheduling, we defined it as makespan (the maximum expected processing time) given by Eq. (4).

$$MS^{k}(t) = argmax_{j \in J} \sum_{i \in I} (d_{ij}) \qquad (4)$$

We have calculated the availability time of VM after processing all tasks assigned by ant k. Where d_{ij} is a simple heuristic that helps to visible the path from i to j is defined by Eq. (5).

$$d_{ij} = \frac{Task\_len_i}{VM\_Speed_j} + VM_{availability\_time} \qquad (5)$$

Here, d_{ij} depends on two things. One is time taken to execute the i^{th} task and j^{th} VM and another is VMs availability time indicating after how much time VM can execute the ith task. This equation plays key role in performance of ATS. Since, it includes processing time and availability time, it reduces the makespan of the system and balances load on the virtual machines.

We have calculated the visibility of pheromone on the path from i^{th} task to j^{th} VM is defined by heuristic d_{ij} by following Eq. (6).

$$\eta_{ij} = \frac{1}{d_{ij}} \qquad (6)$$

When every ant finds out their best path to assign task set to VM set, then one ant update value of τ_{ij} globally by considering the global best path till iteration t as in Eq. (7).

$$\tau_{ij}(t) = \tau_{ij}(t) + \frac{Q}{MS^+} \quad if\ (i,j) \in T^+ \qquad (7)$$

Here, MS^+ denotes the makespan of the system for the optimal allocation (T^+).

3.4 Probabilistic Transition Rule

The probabilistic transition rule shows how the probability is assigned to map i^{th} task to j^{th} VM and it is shown in (1). During every iteration of ACO algorithm, each ant built and update the total probability values of every VM for each task. According to the probability, the tasks are assigned to different VMs.

4. ALGORITHM FOR ATS

Makespan minimization is an essential responsibility in the cloud computing environment to achieve maximum utilization of resources [20, 21]. In this section, we have discussed our proposed algorithm for adaptive task scheduler using ACO is shown in Algorithm 1 named it as ACO-based Task Scheduling (ACOTS). The proposed algorithm runs for a finite number of iterations (*max_no_iter*). The *task_list*, *VM_list*, ETC_{ij} are provided as input to the Algorithm 1 to get the makespan of the system as output. The ETC matrix represents the expected time to compute the i^{th} task on j^{th} VM. The ETC matrix is build using the task length from *task_list* and processing speed of the VM from *VM_list* (i.e., ETC_{ij} = *Length (task_list$_i$)* / *Speed(VM_list$_j$)*). The variable *iter* is initialized with 1 and incremented by 1 after each iteration until the value of *iter* reaches to *max_no_iter*. Initially, the execution time (*ET*) for all VM is set to 0. Then, we have to find out the appropriate VM from the *VM_list* for each task through each ant movement. Therefore, we have estimated the probability value of the task for each VM through k^{th} ant using Eq. (1).

The variable *VMT[k][i]* holds the VM_id for the allocation of i^{th} task through k^{th} ant. In step-21 of Algorithm-1, A_i keeps the ant_id for i^{th} task and this will be performed as follows. The matrix *prob* has probability values of each ant for different tasks. Therefore, the selection of ant will be the ant which has maximum probability value for the specific task. After that, we get the VM_id as *j*. Then, the allocation is done for i^{th} task to j^{th} VM and the *EX*-values of the VMs are updated using the previous *EX*-value and the *ETC*-value. The variable *Makespan_Iter* keeps the makespan value for that iteration. After successful completion of all iterations, the global makespan value is found out in step-32 of Algorithm 1.

5. SIMULATION & RESULTS

We have evaluated the proposed algorithm (ACOTS) through simulation with generated datasets. The experiments were done using CloudSim-3.0.3 simulator [22]. The version of the system is Intel Core i7 4^{th} Generation processor, 3.4 GHz CPU and 8GB RAM running on Microsoft Windows 8 platform. The task arrival rate is generated with Pareto distribution. The performance of ACO based task scheduling algorithm is highly depended on different parameters. We have conducted experiments to decide best parameter to enhance algorithms. To find out best value of each parameter, we have varied those parameters by keeping remaining parameters constant. The details of the tasks, virtual machines characteristics are explained in the Table 3.

Table 3. Simulation Environment

Entity	Attribute	Value
VM	Processing Power	1000-2000 MIPS
	Number of VMs	50
Task	Task Size	1000-10000 MI
	Number of Tasks	500

Algorithm 1 ACO-based Task Scheduling (ACOTS)

Input: *task_list, VM_list,* ETC_{ij}

Output: *Makespan*

1. *iter* = 1
2. *EX = 0*
3. While *iter* + + ≤ max_*no_iter* do
4. Initialize *VM_list[m][n]* for all ants set *VM_list* except first ant and task set *task_list*.
5. Initialize τ_{ij} to positive value $\forall i \in$ *task_list* and $\forall j \in$ *VM_list*
6. for each ant *k* = 1 to z do
7. for *i* = 1 to *n* do
8. *max_prob* = 0, *id* = 0;
9. for *j* = 1 to *m* do
10. Compute P_{ij}^{k} by using Eq. (1)
11. if $P_{ij}^{k} > max\_prob_{ij}$ then
12. $max\_prob_{ij} = P_{ij}^{k}$
13. *id* = *j*;
14. end if
15. end for
16. *VMT[k][i]* = *id*;
17. *Prob[k][i]* = max_prob_{ij}
18. end for
19. end for
20. for *i* = 1 to n do
21. A_i = Ant_ID{Max(prob[k][i]) over k, 1 ≤ k ≤ z}
22. end for
23. for *i* = 1 to n do
24. j = VMT[A_i][i]
25. Task_list[i] is allocated to VM_list[j]
26. EX[j] = EX[j] + ETC_{ij}
27. end for
28. *Makespan_Iter[iter]* = Max{ *EX over all VMs*}
29. Update local pheromone by using (2)
30. Update global pheromone by using (7)
31. end while
32. *Makespan* = Max{ *Makespan_Iter*}
33. return *Makespan*

5.1 Setting & Evaluation of ACO Parameters

As Explained, the performance of the ACO algorithm depends on the values of parameters. Here, we have considered six parameters and those are *α, β, ρ, z, Q,* and *max_no_iter*. We tested possible values of every parameter by keeping other parameters constant.

The number of tasks, VMs and their attributes are kept constant for the simulation. Initially, we have taken the value of parameters as α = 0.5, β = 0.5, ρ = 0.5, z = 5, Q = 50, and max_no_iter = 100. We have estimated the makespan of system by changing the parameters in the order of α, β, ρ, z, Q, and max_no_iter within the range from the more influencing factor as shown in Table 4. In Table 4, the second column (Range) is in the form of a-b-c which means that

parameter varied from a to c with step wise increment of b. The makespan value by changing the parameters in the order of *α, β, ρ, z, Q,* and *max_no_iter* are shown in Figure 1 to Figure 6 respectively.

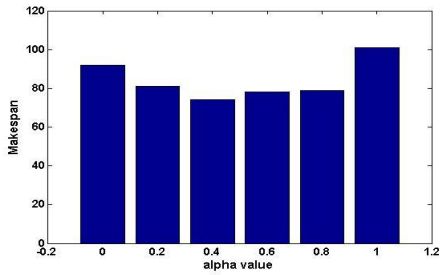

Figure 1. ACOTS Performance with Different α Values to calculate makespan

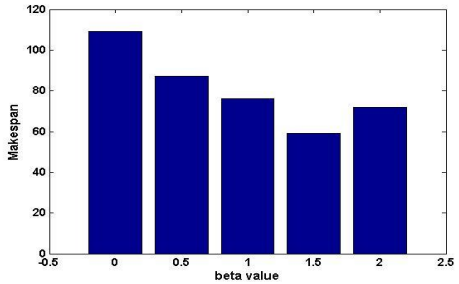

Figure 2. ACOTS Performance with different β values to calculate makespan

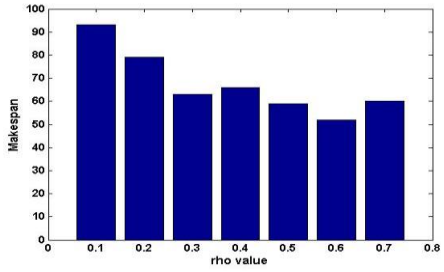

Figure 3. ACOTS Performance with different ρ values to calculate makespan

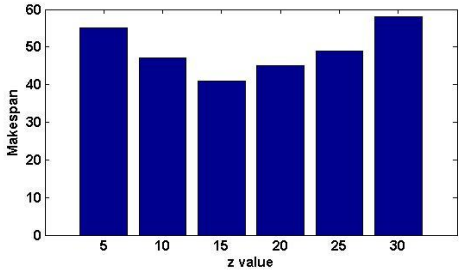

Figure 4. ACOTS performance with different z values to calculate makespan

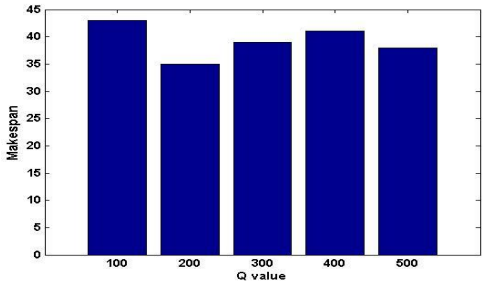

Figure 5. ACOTS Performance with different Q values to calculate makespan

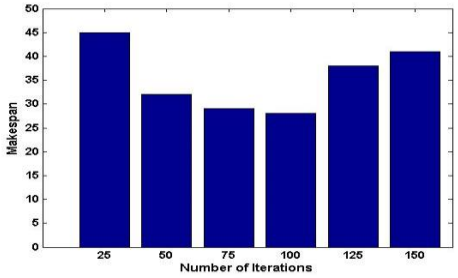

Figure 6. ACOTS Performance with Different number of iterations to calculate Makespan

From these results, we have set all the mentioned parameters for the comparison of ACOTS algorithm with Random and Round-Robin algorithm. The optimal values of these parameters are listed in the last column of Table 4. The number of heterogeneous input tasks is initially 200 and increased to 1000 in the interval of 200. The number heterogeneous VMs is fixed as 50. Fig. 7 and Fig. 8 shows the makespan and average waiting time comparison respectively for Random, Round-Robin, and ACOTS algorithms. It is surely evident from the bar-graphs that ACOTS is more effective when evaluated with the other two standard algorithms. Figure 7 and Figure 8 shows the makespan and average waiting time comparison respectively for Random, Round-Robin, and ACOTS algorithms. It is surely evident from the bar-graphs that ACOTS is more effective when evaluated with the other two standard algorithms.

Table 4. Fixing of ACO Based ATS Parameters

Parameter	Range	Other Parameters	Best Value
α	0 – 0.2 – 1	β = 0.5; ρ = 0.5; z = 5; Q = 100; *max*_no_iter = 50	0.4
β	0 – 0.5 – 2	α = 0.3; ρ = 0.5; z = 5; Q = 100; *max*_no_iter = 100	1.5
ρ	0.1 – 0.1 – 0.7	α = 0.3; β = 1.5; z = 5; Q = 100; *max*_no_iter = 100	0.6
z	5 – 5 – 30	α = 0.3; β = 1.5; ρ = 0.6; Q = 100; *max*_no_iter = 100	15
Q	100 – 100 – 500	α = 0.3; β = 1.5; ρ = 0.6; z = 15; *max*_no_iter = 100	150
max_no_iter	25 – 25 – 150	α = 0.3; β = 1.5; ρ = 0.6; z = 15; Q = 150	75

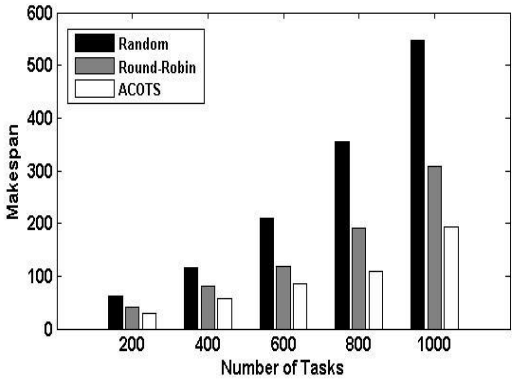

Figure 7. Comparison of makespan of various algorithms to calculate makespan

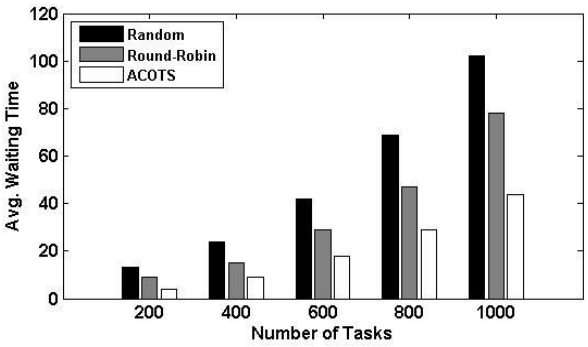

Figure 8. Comparison of waiting time of various algorithms to calculate makespan

6. CONCLUSION

We have studied the allocation of tasks in the heterogeneous cloud computing environment. We have proposed an algorithm (ACOTS) for the distribution of user requests to virtualized resources with optimal makespan and the minimum average waiting time. The proposed ACO-based approach also considered the bursty arrival of tasks in the cloud environment. The proposed algorithm assigns tasks to VMs using probability function to optimize the mentioned parameters. A task with a high priority gets service first to minimize the makespan of the system and to maximize the profit of CSP. The parameter of designed algorithm had set for the optimal performance of ACOTS through simulation in CloudSim simulator. We have compared our proposed algorithm with the Random and Round-Robin algorithm. The simulation results show the satisfactory results.

7. REFERENCES

[1] Fan, G., Yu, H., and Chen, L. (2016). A formal aspect-oriented method for modeling and analyzing adaptive resource scheduling in cloud computing. *IEEE Transactions on Network and Service Management*, 13(2), 281-294. DOI= 10.1109/TNSM.2016.2553157.

[2] Nayak, S. C., and Tripathy, C. (2016). Deadline sensitive lease scheduling in cloud computing environment using AHP. *Journal of King Saud University-Computer and Information Sciences*. DOI = https://doi.org/10.1016/j.jksuci.2016.05.003.

[3] Jain, N. K., Willke, T. L., Datta, K., and Yigitbasi, N. (2016). *U.S. Patent No. 9,342,376*. Washington, DC: U.S. Patent and Trademark Office.

[4] Mishra, S. K., Sahoo, K. S., Sahoo, B., & Jena, S. K. (2016). Metaheuristic Approaches to Task Consolidation Problem in the Cloud. *Resource Management and Efficiency in Cloud Computing Environments*, 168.

[5] Colorni, A., Dorigo, M., and Maniezzo, V. (1992). Distributed optimization by ant colonies. *In Toward a practice of autonomous systems: proceedings of the First European Conference on Artificial Life*, pp. 134, Mit Press.

[6] Liang, Y. C., and Smith, A. E. (2004). An ant colony optimization algorithm for the redundancy allocation problem (RAP). *IEEE Transactions on reliability*, *53*(3), 417-423. **DOI =** 10.1109/TR.2004.832816.

[7] Pacini, E., Mateos, C., and Garino, C. G. (2015). Balancing throughput and response time in online scientific Clouds via Ant Colony Optimization. *Advances in Engineering Software*, 84, 31-47. DOI = https://doi.org/10.1016/j.advengsoft.2015.01.005.

[8] Puthal, D., Sahoo, B. P. S., Mishra, S., and Swain, S. (2015, January). Cloud computing features, issues, and challenges: a big picture. In *Computational Intelligence and Networks (CINE), 2015 International Conference on* (pp. 116-123). IEEE.

[9] Tawfeek, M. A., El-Sisi, A., Keshk, A. E., and Torkey, F. A. (2013). Cloud task scheduling based on ant colony optimization. *8th International Conference on Computer Engineering & Systems (ICCES)*, 64-69. DOI = 10.1109/ICCES.2013.6707172.

[10] Chiang, C. W., Lee, Y. C., Lee, C. N., and Chou, T. Y. (2006). Ant colony optimisation for task matching and scheduling. *IEEE Proceedings-Computers and Digital Techniques*, 153(6), 373-380. DOI = 10.1049/ip-cdt:20050196.

[11] Wen, X., Huang, M., and Shi, J. (2012). Study on resources scheduling based on ACO algorithm and PSO algorithm in cloud computing. *11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science (DCABES)*, 219-222. DOI = 10.1109/DCABES.2012.63.

[12] Zhang, Z., and Zhang, X. (2010). A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. *2nd International Conference on Industrial Mechatronics and Automation (ICIMA)*, 2, 240-243. DOI = 10.1109/ICINDMA.2010.5538385.

[13] Li, K., Xu, G., Zhao, G., Dong, Y., and Wang, D. (2011). Cloud task scheduling based on load balancing ant colony optimization. *In Chinagrid Sixth Annual Conference (ChinaGrid)*, 3-9. DOI = 10.1109/ChinaGrid.2011.17.

[14] Nishant, K., Sharma, P., Krishna, V., Gupta, C., Singh, K. P., and Rastogi, R. (2012). Load balancing of nodes in cloud using ant colony optimization. *14th International Conference on Computer Modelling and Simulation (UKSim)*, 3-8. DOI = 10.1109/UKSim.2012.11.

[15] Lu, X., and Gu, Z. (2011). A load-adapative cloud resource scheduling model based on ant colony algorithm. *IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, 296-300. DOI = 10.1109/CCIS.2011.6045078.

[16] Dam, S., Mandal, G., Dasgupta, K., and Dutta, P. (2014). An ant colony based load balancing strategy in cloud computing. *Springer publishing in Advanced Computing, Networking and Informatics*, 2, 403-413. DOI = https://doi.org/10.1007/978-3-319-07350-7_45.

[17] Mathiyalagan, P., Suriya, S., and Sivanandam, S. N. (2010). Modified ant colony algorithm for grid scheduling. *International Journal on computer science and Engineering*, 2(02), 132-139.

[18] Liu, A., and Wang, Z. (2008). Grid task scheduling based on adaptive ant colony algorithm. *International Conference on Management of e-Commerce and e-Government (ICMECG'08)*, 415-418. DOI = 10.1109/ICMECG.2008.50.

[19] Bagherzadeh, J., and MadadyarAdeh, M. (2009). An improved ant algorithm for grid scheduling problem. *14th International CSI on Computer Conference (CSICC)*, 323-328. DOI = 10.1109/CSICC.2009.5349368.

[20] Kalra, M., and Singh, S. (2015). A review of metaheuristic scheduling techniques in cloud computing. *Egyptian informatics journal*, 16(3), 275-295. DOI = https://doi.org/10.1016/j.eij.2015.07.001.

[21] Mishra, S. K., Puthal, D., Sahoo, B., Jena, S. K., and Obaidat, M. S. (2017). An adaptive task allocation technique for green cloud computing. *The Journal of Supercomputing*. DOI = https://doi.org/10.1007/s11227-017-2133-4.

[22] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). 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. DOI = 10.1002/spe.995.