Dynamic Task Scheduling Algorithm based on Ant Colony Scheme

Kamolov Nizomiddin Baxodirjonovich[#1], Tae-Young Choe [*2]

[#] Department of IT Convergence Engineering,
Kumoh National Institute of Technology,
Daehak-ro, Gumi, Gyeongbuk, South Korea
[1] knbuzb@gmail.com

[*] Department of Computer Engineering,
Kumoh National Institute of Technology,
Daehak-ro, Gumi, Gyeongbuk, South Korea
[2] choety@kumoh.ac.kr

***Abstract*—Many scientific applications running in Cloud Computing system are workflow applications that contains large number of tasks and in which tasks are connected by precedence relations. Efficient scheduling the workflow tasks become a challenging issue in Cloud Computing environments because the scheduling decides performance of the applications. Unfortunately, finding the optimal scheduling is known as NP-hard. Ant Colony Optimization algorithm can be applied to design efficient scheduling algorithms. Previous scheduling algorithms that use Ant Colony mechanism lack rapid adaptivity. This paper proposes a task scheduling algorithm that uses a modified Ant Colony Optimization. The modified version uses probability in order for ants to decide target machine. The proposed task scheduling algorithm is implemented in WorkflowSim in order to measure performance. The experimental results show that the proposed scheduling algorithm reduce average makespan to about 6.4% compared to a scheduling algorithm that uses basic Ant Colony Optimization scheme.**

Keyword-task scheduling, dynamic load balancing, Ant Colony Optimization, WorkflowSim, Pegasus workflows

I. INTRODUCTION

Scientific computing includes a huge number of fields such as biology, chemistry, physics, finance, geophysics, mathematics and mechanics. These sciences use applications that usually distributed, thus the applications may execute faster than sequential ones. It is significant in some fields including weather forecasting or financial modeling to get results as fast as possible [1]. Scientific applications usually may produce many workflow tasks. A workflow is a set of tasks which is dependent to each other. In general, a workflow can be represent as a Direct Acyclic Graph (DAG). Fig.1 shows an example of DAG.

Workflow scheduling is a mapping the tasks in DAGs to resources based on scheduling algorithms [2]. The goal of workflows scheduling in Cloud computing is to achieve a mapping that has shortest makespan. Makespan is the time difference between the first tasks start time and the last tasks finish time. Minimizing the makespan affects performance of the Cloud computing system. Fig. 2 shows an example of workflow scheduling to two Clouds. Cost is a significant factor in scheduling of cloud computing because of ultra large scale and pay-per-use business model. Market driven cloud users and providers can have mutual benefits from an efficient scheduling system.

Since cloud services require great amount of control and manage resources, a good workflow scheduling in important to manage jobs and tasks. Workflow scheduling plays a key role in the workflow management system. After submitting workflow by a client, a broker or scheduler is used to run the scheduling algorithm so that the system can start to make decision. In cloud-based infrastructure, the physical machines are virtualized into unified resources called virtual machines (VMs). The scheduler decides which VMs will be used, as well as which tasks will be executed on each of these resources. It allocates workflow tasks to suitable virtual machine so that the process of computation can be executed to satisfy QoS constraints specified by users such as deadline and cost. This QoS-based optimization aims to minimizing execution cost or make execution time as short as possible and a specified budget [3].

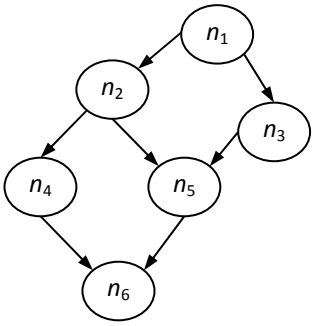

Fig. 2. An example of DAG

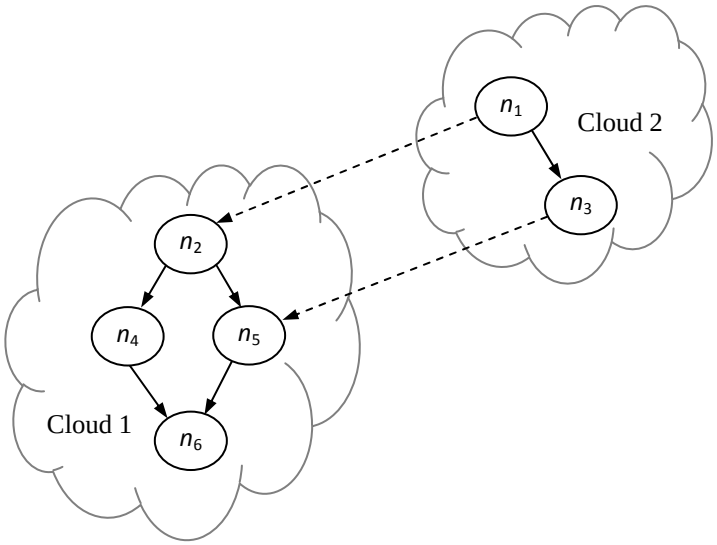

Fig. 1. Mapping workflow tasks to Cloud environment

Since task scheduling complicated process, precedence between the tasks is done scheduling more difficult. In this paper we compare our proposed Probabilistic Load-balancing Ant Colony (PLAC) algorithm with Min-Min and basic ACO. Min-Min choose small tasks first, as result large tasks stay in a long time queue list which is highly influence makespan of total tasks. Even through ACO shows better performance than Min-Min algorithm, it does not consider balancing the loads between VMs. So it is added load balancing factor for balance VMs load and reduce the average makespan. In order to do that it is utilized execution times of successfully finished tasks and number of tasks in a queue. Based on execution times of finished tasks and number of waiting tasks we proposed Expected Execution Time (EET) of tasks as a load measurement of resource.

The rest of this paper is organized as follows: Chapter 2 is dedicated to related works. Proposed task scheduling algorithm is described in Chapter 3. Chapter 4 is dedicated to experimental results. As such experimental settings are introduced and performance results of algorithms are analyzed. Finally, in Chapter 5 we conclude our paper and introduce some future work in Chapter 5.

II. RELATED WORKS

First Come First Serve (FCFS) algorithm is usually considered for parallel processing. It selects the resources with the smallest waiting queue for the incoming tasks. The disadvantage of FCFS is that it is non-preemptive. In non-primitive scheduling process is automatically queued and processing occurs according to process order. The shortest tasks which are at the back of the queue have to wait for the long task at the front to finish. Its turnaround and response time quite low [4].

Min-Min algorithm chooses the task with the smallest length (Million Instruction) from the list. After that, the task will be assigned to resource which has minimum expected completion time. Drawback of the algorithm is that it chooses small tasks to be executed firstly, which in turn large task delays for long time [5].

Max-Min algorithm is very similar to Min-Min, but it chooses the task with the largest length and it assigns it to resource that has minimum expected completion time. Drawback of the algorithm is that it chooses large

tasks to be executed firstly, which in turn small task delays for long time [6].

Priority scheduling algorithm considers the priority of jobs for scheduling. It is based on multiple criteria decision making model. The list of the jobs is sorted based on priority. The task with highest priority is chosen and it assigns it to resource that has minimum expected completion time. Drawback of the algorithm low-priority processes may wait long time [7].

Ant colony optimization (*ACO*) algorithm introduced by Dorigo based on the behavior of real ants [8], it is a heuristic algorithm for the solution of combinatorial optimization problems. The basic idea of ACO is to simulate the foraging behavior of ant colonies. When a group of ants go to search food source from the nest, they communicate with special kind of chemical called pheromone. Once the ants discover a path to food, they deposit pheromone on the path. By sensing pheromone on the ground, an ant can follow the trails of the other ants to the food source. As this process continues, most of the ants tend to choose the shortest path which has huge amount pheromone on this path. This collective pheromone-depositing and pheromone-following behavior of ants becomes the inspiring source of ACO.

At time zero, ants are positioned on different towns, the initial values $\tau_{ij}(0)$ for trail intensity are set on edge (i, j). The first element of each ant's tabu list is set to be equal to its starting town [9]. Thereafter the k-ant moves from town i to town j with a probability that is defined as:

$$P_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}(t)^{\alpha} \cdot \eta_{ij}(t)^{\beta}}{\sum_{l=allowed_k} \tau_{il}(t)^{\alpha} \cdot \eta_{il}(t)^{\beta}} & \text{if } j \in allowed_k \\ 0 & \text{otherwise} \end{cases}, \quad (1)$$

where $allowed_k = \{N\} - \{tabu_k\}$, $tabu_k$ is the tabu list of k_{th} ant, $\tau_{ij}(t)$ is the pheromone value on edge (i, j), η_{ij} is the value of the heuristic value, and $\eta_{ij}(t) = 1/d_{ij}$. Where d_{ij} is the distance between node n_i and node n_j. α, β are two parameters that control the relative weight of the pheromone trail and heuristic value. Finally the most optimal and effective path is selected and globally updated [10, 11]. Although ACO algorithm is suitable when input workflow is not stereotype, it does not consider load state in VMs.

Load Balancing Ant Colony Optimization (*LBACO*) algorithm is proposed by Li et al. [12]. LBACO algorithm adds balancing factor to compute probability of each VM as follows:

$$p_{j}^{k} = \begin{cases} \dfrac{\tau_{j}(t)^{\alpha} \cdot CC_{j}^{\beta} \cdot LB_{j}^{\gamma}}{\sum_{i=1}^{n} \tau_{i}(t)^{\alpha} \cdot CC_{i}^{\beta} \cdot LB_{i}^{\gamma}} & \text{if } j \in \{1...n\} \\ 0 & \text{otherwise} \end{cases}, \quad (2)$$

where τ_j(t) is the pheromone of VM_j, CC_j is the computing capacity of VM_j, LB_j is the load balancing factor of VM_j, α, β, γ are three parameters that control relative weights of components, and n is the number of VMs in the system. LBACO decides the load balancing factor based on differences between entire task load and the task loads of the previous iteration duration. Since the decision of load state depends on the relative loads against entire load history, it could lead to stubborn and not adaptive scheduling. Also LBACO does not consider workflow model.

III. PROPOSED SCHEDULING ALGORITHM

A. System model

This paper propose *Probabilistic Load-balancing Ant Colony* (PLAC) algorithm used by a broker. The broker receives requests from users as shown in Fig. 3. A request can be constructed with multiple tasks in the form of workflow. The broker selects a ready task among arrived workflow tasks. The *ready* task means that the task can execute immediately because all parent tasks of the task are completed. The broker asks to PLAC algorithm in order to select suitable VM. PLAC algorithm selects the target VM using ants and pheromones. The broker forwards the task to the selected VM. If the broker receives completed task from a host, it returns the task to its owner and informs to PLAC algorithm. PLAC algorithm updates values in its data structure including expected execution time of a VM.

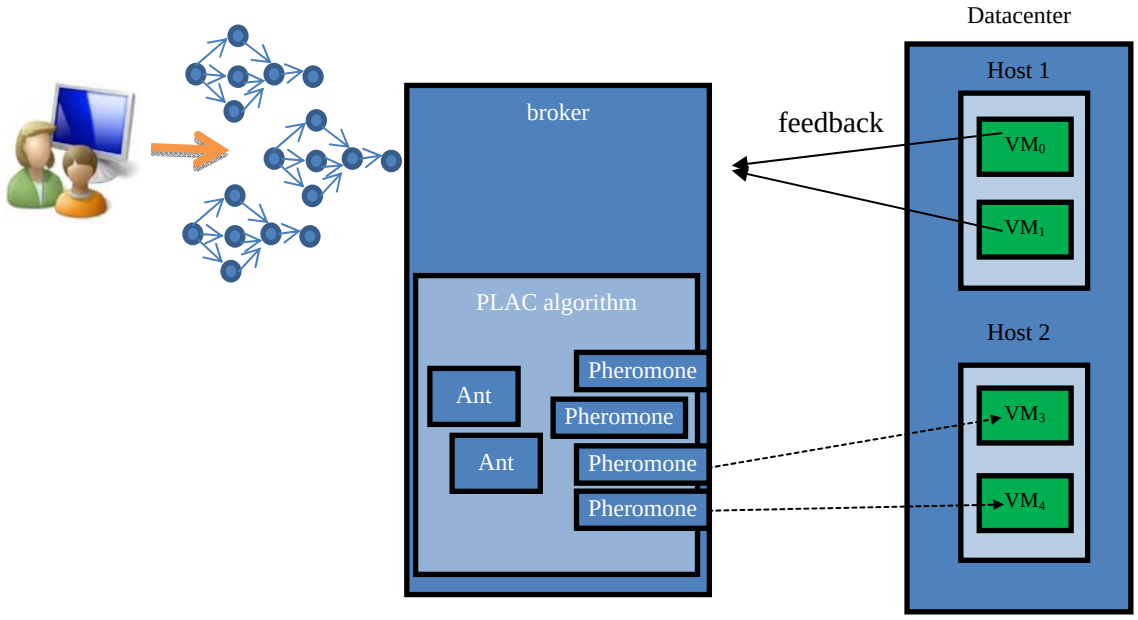

Fig. 3. The system model

In order to schedule a task proposed by the broker, PLAC algorithm let each ant select a target VM. An ant computes expected execution time of the task in each VM. Expected makespan of a VM can be calculated by adding the expected execution time of the task to the VM. Using the makespans, computing powers of VMs, and pheromones, the probability of each VM is computed. The sum of the probabilities of all VMs is 1.0. Each ant randomly votes a VM based on the probabilities. PLAC algorithm selects a VM that has the highest poll by the rule of 'decision by majority'. Thus the number of ants should be greater than the number of VMs. Since a VM that does not have the shortest makespan can be selected, there is a chance to escape from local optima.

B. Process of PLAC algorithm

PLAC algorithm is composed of an initialization step and repeated two steps: choosing target VM step and updating pheromone step. In the initialization step, computing capacity CC_i of VM_i is calculated based on formula (3).

$$CC_j = pe\_num_j \cdot pe\_mips_j + vm\_bw_j, \tag{3}$$

where pe_num_j is the number of processing elements in VM_j, pe_mips_j is the MIPS (Million Instructions Per Second) of each processing elements in VM_j, and vm_bw_j is communication bandwidth ability of VM_j. Next, pheromone $\tau_j(0)$ of VM_j at time 0 is initialized to CC_j as formula (4).

$$\tau_j(0) = CC_j \tag{4}$$

The values are shared by all ants.

In the choosing target VM step, each ant computes values for decision. Given task n_i, expected computation time $ET_j(i)$ in VM_j is calculated using formula (5).

$$ET_j(i) = \frac{task\_length_i}{pe\_num_j \cdot pe\_mips_j}, \tag{5}$$

where $task\_length_i$ is the length of task n_i. Each ant computes the time for all VMs. For VM_j, expected execution time $EET_j(t)$ of VM_j at time t is calculated using formula (6).

$$EET_j(t) = \sum_{n_i \in VM_j} ET_j(i) - R_j(t) \tag{6}$$

where $R_j(t)$ is the remained process time of current running task in VM_j as shown in Fig. 4. $R_j(t)$ is an expected time computed from submitted time, the current time, and execution of task running in VM_j currently. Broker maintains queues for VMs and expects currently running task based on feedbacks of finished tasks.

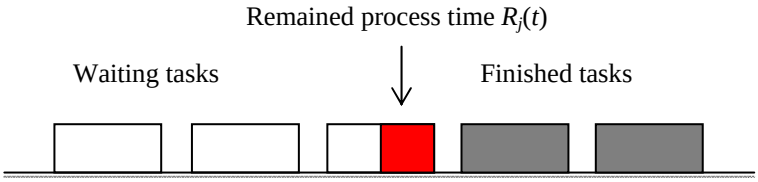

Fig. 4. Remained process time

Load balancing factor $LB_j(t)$ of VM_j at time t is calculated using formula (7). The load balancing factor is the ratio of entire loads in the system over loads in VM_j. The high value of the load balancing factor means that the VM is lightly loaded compared to other VMs.

$$LB_j(t) = \frac{\sum_{VM_k} EET_k(t)}{EET_j(t)} \tag{7}$$

Probability $P^k_j(t)$ of choosing VM_j by ant Ant_k at time t is calculated using formula (8).

$$P_j^k(t) = \frac{\tau_j(t)^\alpha \cdot CC_j^{\ \beta} \cdot LB_j(t)^\gamma}{\sum_k \tau_k(t)^\alpha \cdot CC_k^{\ \beta} \cdot LB_k(t)^\gamma}, \tag{8}$$

where α, β, γ are variable parameters which control influence of pheromone value, computing capacity, and load balancing factor, respectively.

After calculating all probabilities of VMs, each ant selects a VM based on the probability values. VM with the highest probability means that it has more chance to be selected. Although a VM has the lowest probability, it can be selected in order to escape from local optima. All ants vote their target VM and a VM that has the highest poll is selected as the final target for the task.

Updating pheromone step changes pheromone value based on decided target VM as follows:

$$\tau_j(t+1) = \tau_j(t)(1-\rho) + \begin{cases} \dfrac{1}{T(t)} & \text{if } VM_j \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

C. Example of LBACO algorithm

Let us show an example of the proposed PLAC algorithm. Assume that there are 3 VMs (VM_1, VM_2, and VM_3), 10 tasks (n_1, n_2… n_{10}), and 4 ants (Ant_1, Ant_2, Ant_3, and Ant_4). Let's assume that all VMs have 100 MIPS computing power, the size of each task is 1000 Million Instructions (MIs), network bandwidth is 1000 Instructions per second, and parameters are follows: $\alpha = 3$, $\beta = 2$, $\gamma = 8$, $\rho = 0.01$. Computing capacities of VMs are calculated in the initialization step using formula (3) as follows:

$CC_1 = 1 \cdot 100 + 1000;$ $CC_2 = 1100;$ $CC_3 = 1100.$

Initial pheromones of all VMs are initialized as follows according to formula (4):

$\tau_1(0) = CC_1 = 1100;$ $\tau_2(0) = 1100;$ $\tau_3(0) = 1100.$

Every ant chooses a VM for the next task according to formula (5) - (7). In the case of the first task n_1, from formula (5),

$ET_1(n_1) = 100 / (1 \cdot 1000) + 0 = 0.1;$ $ET_2(n_1) = 0.1;$ $ET_3(n_1) = 0.1.$

Since there is no task in queue of VMs at the initial time, $EET(j) = ET_j(n_1)$. Thus using formula (6) – (8)

$EET(1) = ET_1(n_1);$ $EET(2) = ET_2(n_1);$ $EET(3) = ET_3(n_1);$

$LB_1(0) = (0.1 + 0.1 + 0.1) / 0.1 = 3;$ $LB_1(0) = (0.1 + 0.1 + 0.1) / 0.1 = 3;$ $LB_1(0) = (0.1 + 0.1 + 0.1) / 0.1 = 3;$

$P_1^{\ 1}(0) = 1100^3 \cdot 1100^2 \cdot 3^8 / (1100^3 \cdot 1100^2 \cdot 3^8 + 1100^3 \cdot 1100^2 \cdot 3^8 + 1100^3 \cdot 1100^2 \cdot 3^8) = 1/3;$

$P_2^{\ 1}(0) = 1/3;$ $P_3^{\ 1}(0) = 1/3$

Since probabilities are the same for three VMs, Ant_1 choose target VM randomly. Assume that Ant_1 chooses VM_1. Similarly other 3 ants decide target VMs in order to decide final target for task n_1. Ant_2 has the same probability values $P_1^{\ 2}(0)$, $P_2^{\ 2}(0)$, and $P_3^{\ 2}(0)$, to decide a target as 1/3. Ant_2 is assumed to select VM_2, Likewise Ant_3 and Ant_4 has the same probability values. Ant_3 and Ant_4 are assumed to select VM_3 and VM_4, respectively. By the rule of 'decision by majority', VM_1 is selected as the target of task n_1. After assigning task n_1 to VM_1, local pheromones are updated according to formula (8). Since the shortest makespan $T(0)$ is 100 / 1000 = 0.1, $\Delta\tau(t)$ is 1 / 0.1 or 10. Thus

$$\tau_1(1) = 1100 \cdot (1 - 0.01) + 1 / 0.1 = 1099; \quad \tau_2(1) = 1100 \cdot (1 - 0.01) = 1089; \quad \tau_3(1) = 1089.$$

In the case of the 2nd task n_2:

$$ET_1(n_2) = 100 / 1000 + 0 = 0.1; \quad ET_2(n_2) = 100 / 1000 + 0 = 0.1; \quad ET_3(n_2) = 0.1;$$

$$EET(1) = 0.1 + 0.1 = 0.2; \quad EET(2) = 0.1; \quad EET(3) = 0.1;$$

$$LB_1(1) = (0.1 + 0.1 + 0.2) / 0.2 = 2; \quad LB_2(1) = (0.1 + 0.1 + 0.2) / 0.1 = 4; \quad LB_3(1) = (0.1 + 0.1 + 0.2) / 0.1 = 4;$$

$$P_1^1(1) = 1099^3 \cdot 1100^2 \cdot 2^8 / (1099^3 \cdot 1100^2 \cdot 2^8 + 1089^3 \cdot 1100^2 \cdot 4^8 + 1089^3 \cdot 1100^2 \cdot 4^8) = 0.002;$$

$$P_2^1(1) = 1089^3 \cdot 1100^2 \cdot 4^8 / (1099^3 \cdot 1100^2 \cdot 2^8 + 1089^3 \cdot 1100^2 \cdot 4^8 + 1089^3 \cdot 1100^2 \cdot 4^8) = 0.499;$$

$$P_3^1(1) = 0.499;$$

Assume that Ant_1 chooses VM_2 for task n_2. Since values to decide targets are the same for Ant_2, Ant_3, and Ant_4, it is assumed that Ant_2, Ant_3, and Ant_4 select VM_3, VM_2, and VM_2, respectively. So, by the rule of 'decision by majority', $VM2$ is selected. Next, local pheromone is updated according to formula (9).

$$\tau_1(1) = 1099 \cdot (1 - 0.01) = 1088.01;$$

$$\tau_2(1) = 1089 \cdot (1 - 0.01) + 10 = 1088.11;$$

$$\tau_3(1) = 1089 \cdot (1 - 0.01) = 1078.11;$$

IV. Implementation and Experimental Results

In order to measure performance of the proposed PLAC algorithm, Min-Min, ACO, and PLAC algorithms are implemented in WorkflowSim [13], an extension of CloudSim [14] toolkit. The CloudSim toolkit is a java programming based discrete event Cloud simulation toolkit. It is built on the top of GridSim [15] simulation toolkit, which provides simulation of large scale cloud computing environments, service brokers, provisioning, allocation policies, and so on. Unfortunately, CloudSim is lack of support for schedule the scientific workflows. WorkflowSim adds functionalities in order to support simulation of scientific Workflows. WorkflowSim can use scientific workflows generated by Pegasus workflow management system [1]. The workflow characteristics are taken from diverse workflows domain such as biology, astronomy, earthquake science, and gravitational physics in order to resemble real workflows.

A. Simulation parameters

The characteristics of resource used in the experiment are given in TABLE I. There is one datacenter and 25 hosts in it. Each host has several VMs based on their power. CPU scheduling in a resource is modeled as space-shared.

TABLE I. Characteristics of resources

Datacenter	
Number of Datacenter	1
Number of hosts	25
VM (Virtual Machine)	
Number of VMs	100
MIPS of PE per VM	1000-3000 MIPS
VM memory	512
Bandwidth	1000
Type of manager	Space shared

TABLE II shows characteristics of tasks. Montage workflow which is generated by Pegasus Workflow Management System. It has 100 tasks with precedence relations.

TABLE II. Characteristics of tasks

workflows type	Montage_100
number of PE requirement	1

TABLE III illustrates scheduling parameters which is used in simulation.

TABLE III. Scheduling parameters

Number of workflows	5-50
Number of ants in colony	8
Iteration number	50
α	3
β	2
γ	8
ρ	0.01

B. Simulation Results

Fig. 6 and Fig. 7 show comparison result of three algorithms on average makespan and minimum makespan respectively. It can be seen that our proposed algorithm shows clearly better performance than other two when have used more than 20 Montage_100 workflows (which consist of 100 tasks with precedence).

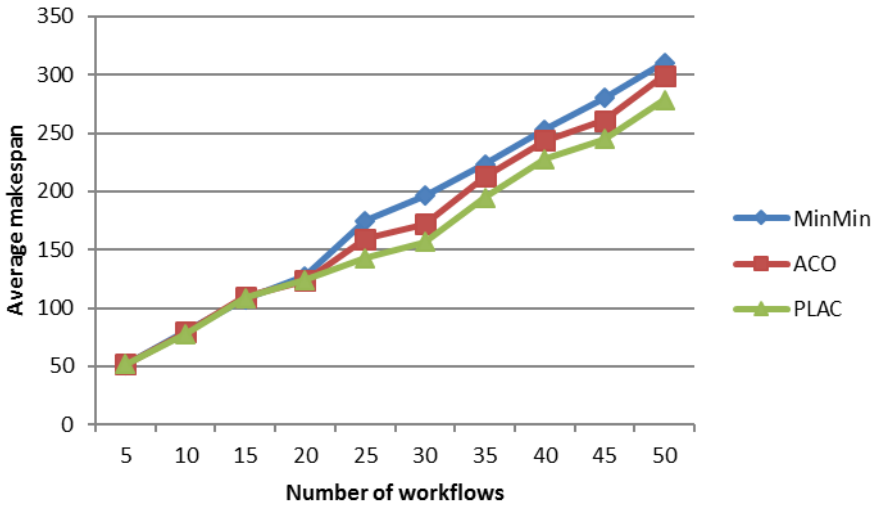

Fig. 6. Average makespan of Montage_100 workflows

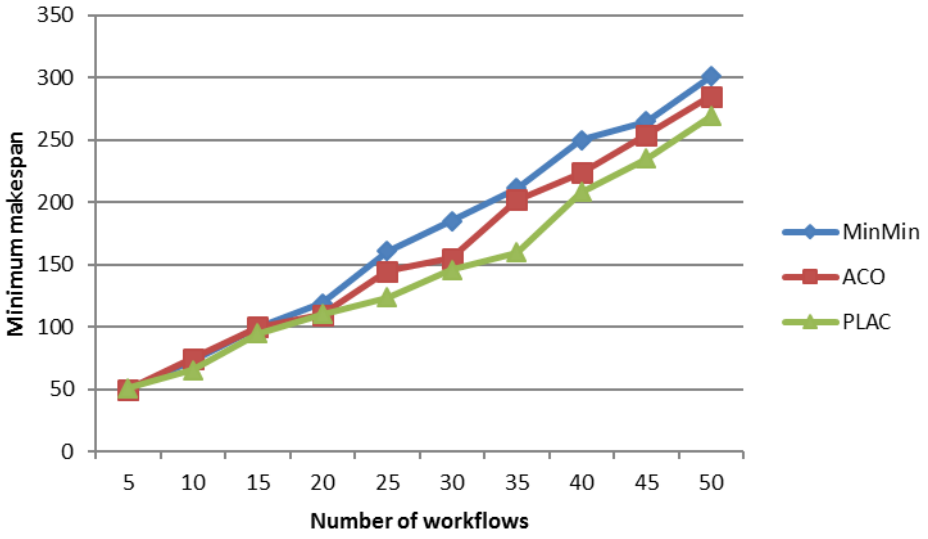

Fig. 7. Minimum makespan Montage_100 workflows

In addition, we tried to compare performance of three algorithms with workflows where each workflow contains only includes one task, in order to prove that our algorithm shows better performance even using

primitive tasks. Fig. 8 and Fig. 9 show comparison using average makespan and minimum makespan, between Min-Min, ACO, and PLAC. ACO and PLAC algorithms show better performance than Min-Min algorithm.

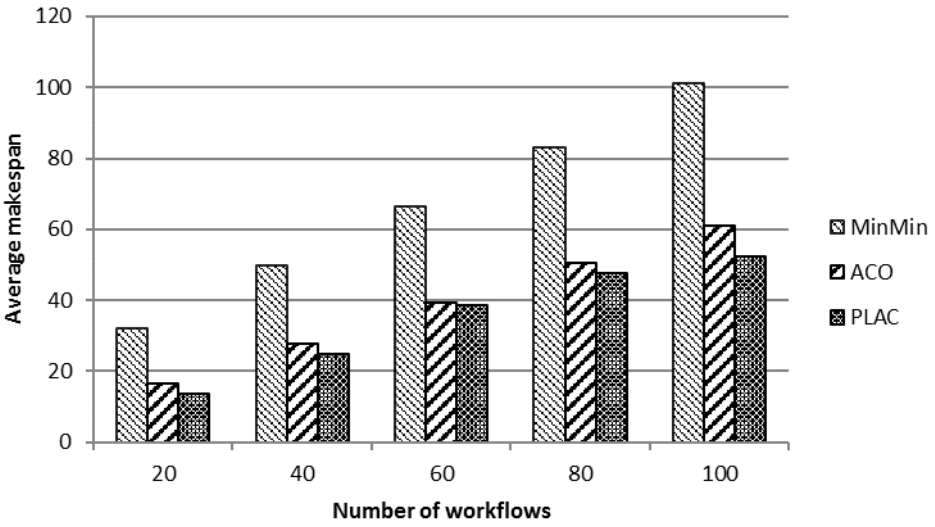

Fig. 8. Average makespan of 1000 workflows

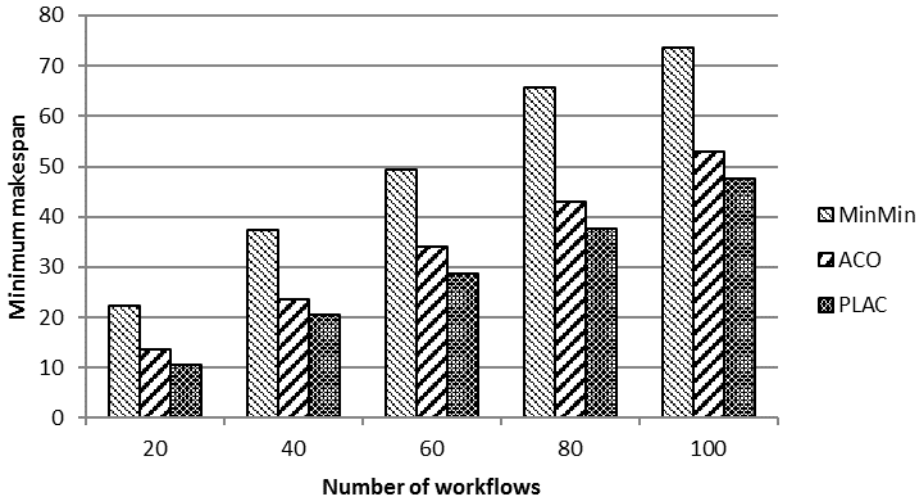

Fig. 9. Minimum makespan of 1000 workflows

Fig. 10 and Fig. 11 compare algorithms with varied type of Pegasus workflows **(**Montage_100, CyberShake, Epigenomics_100, Inspiral_100, Sipht_100). In the varied workflows, LBACO shows better performance than other two. When Montage_100 workflows are applied, average makespan of PLAC ranges 79.9% ~ 101.0% against that of Min-Min algorithm and 89.9% ~ 100.5% against that of ACO algorithm, respectively. In the case of 1000 workflows where each workflow consists of one task, average makespan of PLAC ranges 42.4% ~ 58.2% against that of Min-Min algorithm and 82% ~ 98% against that of ACO algorithm.

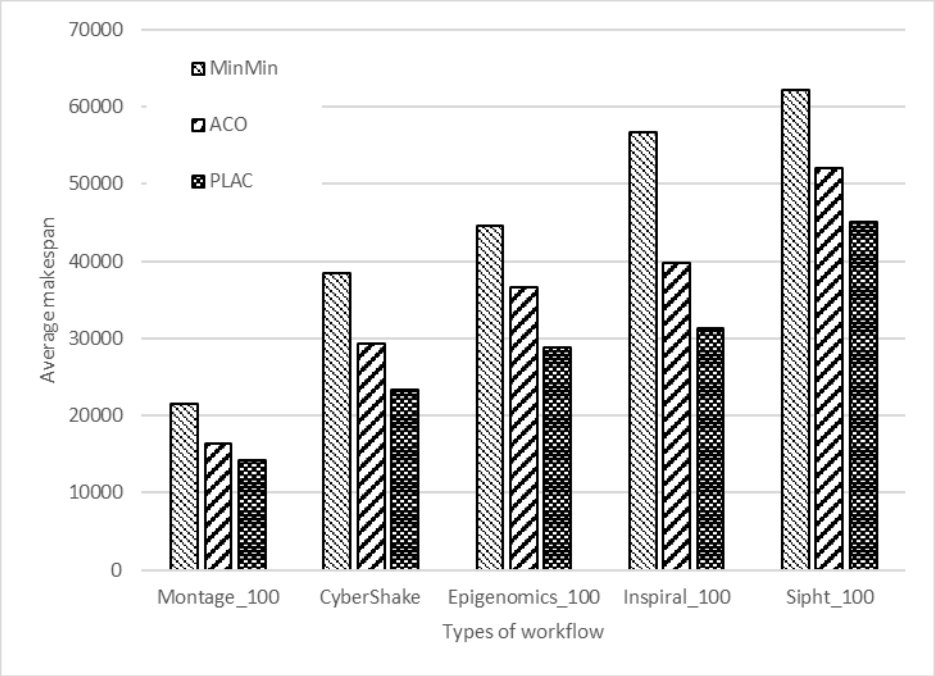

Fig. 10. Average makespan of varied Pegasus workflows

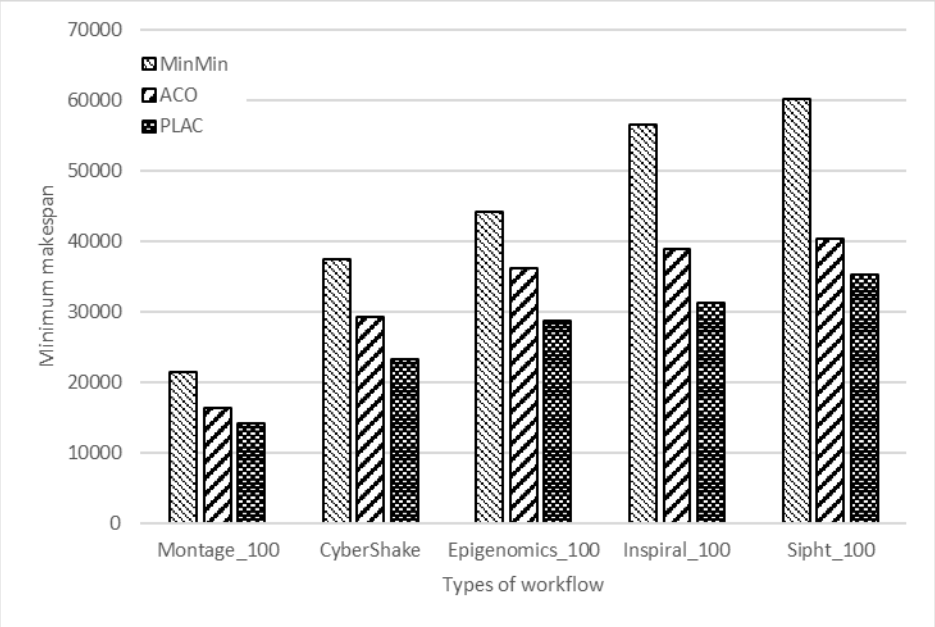

Fig. 11. Minimum makespan of varied Pegasus workflows.

V. Conclusion and Future Works

In this paper, PLAC algorithm is proposed. PLAC algorithm decides target VM considering balancing factors of VMs, manipulates feedback from VMs for maintaining exact queue length in each VM, and uses probabilistic choice and 'decision by majority' in order to escape from local optima. Experiments show comparison results of Min-Min, ACO, and PLAC algorithms using the various workflows. From Montage_100 workflows, 1000 independent tasks, and varied Pegasus workflows, our proposed PLAC algorithm shows better performance than Min-Min and ACO algorithms about 11.5% and 6.4% respectively in overall.

In this paper, we did not consider resource failure or task failure. Also, task clustering is not considered. As a future work we are going to expend our research by including fault tolerance features of Cloud Computing environments and considering task clustering like horizontal or vertical way.

Acknowledgment

This paper was supported by Research Fund, Kumoh National Institute of Technology.

REFERENCES

[1] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi, "Characterizing and Profiling Scientific Workflows," Future Generation Computer Systems Volume 29, Issue 3, pp. 682-692, United States, August 2012.

[2] S. K. Jayadivya and S. Mary Saira Bhanu, "QoS based scheduling of Workflows in Cloud Computing," International Journal of Computer Science and Electrical Engineering (IJCSEE), Volume 1, Issue 1, pp. 15-21 Singapore, 2012.

[3] Kai Wu, "A tunable workflow scheduling Algorithm based on particle Swarm optimization for cloud Computing," Master's projects, paper 358, San José State University, United States, 2014.

[4] Amit Agarwal and Saloni Jain, "Efficient Optimal Algorithm of task scheduling in Cloud computing Environment," International Journal of Computer Trends and Technology (IJCTT), volume 9, number 7, India, March 2014.

[5] Kobra Etminani, "A Min-Min Max-Min Selective Algorithm for Grid Task Scheduling," 3rd IEEE/IFIP International Conference in Central Asia on Internet 2007 (ICI 2007), Iran, September 2007.

[6] Pawar C.S and Wagh R.B, "Priority based Dynamic Resource Allocation in Cloud Computing," International Symposium on Cloud and Service Computing (ISCOS) 2012, India 2012.

[7] Pinal Salot, "A survey of various scheduling algorithm in cloud computing environment," International Journal of Research in Engineering and Technology, Volume 2, Issue 2, pp. 131-135, India, February 2013.

[8] M. Muthucumaru, Shoukat A. Howard, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems,", Proc. of the 8th Heterogeneous Computing Workshop (HCW'99), United States ,1999.

[9] C. Henri, L. Arnaud, and Z. Dmitrii, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," Proc. of the 9th Heterogeneous Computing Workshop (HCW'2000), United States, 2000.

[10] Manpreet Singh, "GRAAA: Grid Resource Allocation Based on Ant Algorithm," Journal of advanced information technology, Volume 1, Australia, 2010.

[11] S. Ravichandran, "Dynamic Scheduling of Data Using Genetic Algorithm in Cloud Computing," International Journal of computing Algorithm, Volume 02, Issue 01, June 2013.

[12] Kun Li, Gaochao Zhao, Yushuang Dong, and Wang D, "Cloud Task Scheduling based on Load Balancing Ant Colony Optimization," Chinagrid Conference pp. 3-9, China, August 2011.

[13] N. Rodrigo, Anton Beloglazov, and Rajkumar Buyya, "CloudSim: A toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning algorithm," Journal Software-Practice & Experience, Volume 41, Issue 1, India, January 2011.

[14] Weiwei Chen, "WorkflowSim: A toolkit for simulating Scientific Workflows in Distributed Environment" IEEE 8th International Conference, E-Science, United States, October, 2012.

[15] R. Buyya and M. Murshed, "Using the GridSim Toolkit for Enabling Grid Computing Education," Proc. of International Conference on Communication Networks and Distributed Systems Modeling and Simulation, Australia, January 27-31, 2002.