Artificial Bee Colony Algorithm for Cloud Task Scheduling

Medhat A. Tawfeek, Ashraf El-Sisi, Arabi E. keshk, F. A. Torkey
Department of Computer Science
Menoufia University
Egypt
medhattaw@yahoo.com, ashrafelsisim@yahoo.com, arabikeshk@yahoo.com, torkey1951@yahoo.com

***Abstract*—Cloud computing services are becoming ubiquitous, and are becoming the primary source of computing power for both enterprises and personal computing applications. One of the fundamental issues in this environment is related to task scheduling. The scheduler should do the scheduling process efficiently in order to utilize the available resources. In this paper a cloud task scheduling policy based on artificial bee colony algorithm compared with different scheduling algorithms has been proposed. The main goal of the proposed algorithm is minimizing the makespan of a given tasks set. Artificial bee colony algorithm models the behavior of honey bees and can be used to find solutions for difficult or impossible combinatorial problems. Algorithms have been simulated using Cloudsim toolkit package. Experimental results showed that the artificial bee colony algorithm outperformed ACO, FPLTF and FCFS algorithms.**

Keywords—cloud computing; task scheduling; makespan; artificial bee colony

I. INTRODUCTION

Cloud computing platform can provide a variety of resources, including infrastructure, software, and services, to users in an on-demand fashion. Cloud computing services eliminate the costs of purchasing and maintaining the infrastructures for cloud users, and allow the users to dynamically scale up and down computing resources in real time based on their needs [1]. To access these resources, a cloud user submits requests for resources. The cloud provider then provides the requested resources from a common resource pool, and allows the user to use these resources for a required time period. The customer is interested in reducing the overall execution time of tasks on the machines [2]. The processing units in cloud environments are called as virtual machines (VMs). The VMs should execute the tasks as early as possible and these VMs run in parallel. Because hundreds of thousands of VMs are used, it is difficult to manually assign tasks to computing resources in clouds [3]. So, efficient algorithms are needed for task scheduling in the cloud environment. A good task scheduler should adapt its scheduling strategy to the changing environment and the types of tasks [4]. Therefore, dynamic task scheduling algorithms that are based on meta-heuristics, such as Artificial Bee Colony (ABC), ant colony optimization (ACO) and Particle Swarm Optimization (PSO) are preferred for clouds. ABC algorithm simulates the foraging behavior of honey bees [5]. It shows good performance in many application problems and large scale optimization problems. ABC algorithm has been applied to various problems including training of neural networks, Traveling Salesman Problem (TSP), clustering and image segmentation [6]. In this paper, we use ABC algorithm to find the near-optimal resource allocation for tasks in the dynamic cloud system to minimize the makespan of tasks on the entire system. Then, this scheduling strategy was simulated using the Cloudsim toolkit package. Experimental results compared to Ant Colony Optimization (ACO) in [7], Fastest Processor to Largest Task First (FPLTF) in [8] and First Come First Served (FCFS) in [9] showed that ABC algorithm satisfies expectation. The remainder of this paper is organized as follows. Section 2 introduces background and scans the some of the important related work. Section 3 covers the basics of honey bee Colonies and the details of cloud scheduling based ABC algorithm. The implementation and simulation results are seen in section 4. Finally, Section 5 concludes this paper.

II. BACKGROUND & RELATED WORK

A. Cloud Computing Environment and Combinatorial Optimization Problem

Cloud computing is a virtual pool of resources which are provided to users. It gives users virtually unlimited pay-per-use computing resources without the burden of managing the underlying infrastructure. The goal of cloud computing service providers is to use the resources efficiently and gain maximum profit [10]. This leads to task scheduling as a core and challenging issue in cloud computing. Scheduling of tasks in cloud computing is a combinatorial optimization problems. In combinatorial optimization problems, we are looking for an object from a finite or possibly infinite set. This object is typically an integer number, a subset, a permutation, or a graph structure [11]. Due to the practical importance of combinatorial optimization problems, many algorithms to tackle them have been developed. These algorithms can be classified as either complete or approximate algorithms. Complete algorithms are guaranteed to find for every finite size instance of a combinatorial optimization problem an optimal solution in bounded time. In approximate methods we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time especially for combinatorial optimization problems

that are NP-hard [12]. Among the basic approximate methods we usually distinguish between constructive methods and local search methods. Constructive algorithms generate solutions from scratch by adding components to an initially empty partial solution until a solution is complete. Local search algorithms start from some initial solution and iteratively try to replace the current solution by a better solution in an appropriately defined neighborhood of the current solution [13]. A new kind of approximate algorithm has emerged which basically tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. This class of algorithms includes ABC, PSO, ACO, simulated annealing (SA), tabu search (TS) and others [11]. A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions [12].

B. Cloudsim

The Simulation is a technique where a program models the behavior of the system (CPU, network etc.) by calculating the interaction between its different entities using mathematical formulas, or actually capturing and playing back observations from a production system [14]. Cloudsim is a framework developed by the GRIDS laboratory of university of Melbourne which enables seamless modeling, simulation and experimenting on designing Cloud computing infrastructures [22]. Cloudsim can be used to model data centers, host, service brokers, scheduling and allocation policies of a large scaled cloud platform. Hence, the researcher has used Cloudsim to model datacenters, hosts, VMs for experimenting in simulated cloud environment [15]. Cloudsim offers the following features:

- Support model of large scale Cloud computing.
- A self contained platform for modeling data centers, service brokers, scheduling, and allocations policies.
- Flexibility to switch between space-shared and time-shared allocation of processing cores to virtualized services.

C. Related Work

Cloud scheduling has always been a research subject whose objective is to ensure that every computing resource is distributed efficiently and fairly and in the end improves resource utility [3, 10]. Millions of user share cloud resources by submitting their computing task to the cloud system. Scheduling these millions of task is a challenge to cloud computing environment [7]. Task scheduling is well studied within the computer operating systems [9]. Most of them can be applied to cloud environment with suitable modifications. The First Come First Serve (FCFS) algorithm is a simple job scheduling algorithm. A job which makes the first requirement will be executed first. The main problem of FCFS is its convoy effect. If all jobs are waiting for a big job to finish, the convoy effect occurs. The convoy effect may lead to longer average waiting time and lower resource utilization [9]. The Fastest Processor to Largest Task First (FPLTF) algorithm schedules tasks to resources according to the workload of tasks. The algorithm needs two main parameters such as the CPU speed of resources and workload of tasks. The scheduler sorts the tasks and resources by their workload and CPU speed then assigns the largest task to the fastest available resource. If there are many tasks with heavy workload, its performance may be very bad [8].

Scheduling based genetic algorithm is proposed in [16, 17]. This algorithms optimizes the energy consumption, carbon dioxide emissions and the generated profit of a geographically distributed cloud computing infrastructure. Scheduling in cloud environment based ACO algorithms are proposed in [2, 4, 7]. In these methods, the requests are collected; the scheduler considers the approximate execution time for each task and use heuristic approach to possibly make better decision. Tasks are scheduled only at some predefined time. This enables batch heuristics to know about the actual execution times of a larger number of tasks. An optimized algorithm for virtual machine placement in cloud computing scheduling based on A multi-objective ant colony system algorithm in cloud computing is proposed in [10].

A model of self-organization that takes place within a colony of honey bee has been presented in [18]. This foraging technique is used in the field of robotics. ABC algorithm also were used in many fields such as digital signal processing [19], job shop scheduling problem [5], block matching for motion estimation [20]. ABC is also used in web services. Honey Bees and Dynamic Server Allocation in Internet Hosting Centers is proposed in [21]. In this method, a new decentralized honey bee algorithm which dynamically allocates servers to satisfy request loads is proposed to maximize revenue. It models servers and request queues in a hosting center as foraging bees and flower patches respectively.

The existing scheduling techniques in clouds, consider parameter or various parameters like performance, makespan, cost, scalability, throughput, resource utilization, fault tolerance, migration time or associated overhead. In this paper, cloud task scheduling based ABC approach has been presented for allocation of incoming jobs to virtual machines (VMs) considering in our account the makespan to help in utilizing the available resources optimally, minimize the resource consumption and achieve a high user satisfaction.

III. CLOUD SCHEDULING BASED ARTIFICIAL BEE COLONY ALGORITHM

A. The Artificial Bee Colony(ABC)

The ABC algorithm is an optimization algorithm based on the intelligent foraging behavior of honey bee swarm that was proposed in 2005 [22, 23]. This new Metaheuristic is inspired by the intelligent foraging behavior of honey bee swarm. Common honey bees assume different roles within their colony over time. A typical hive may have 5,000 to 20,000 individual bees. Mature bees (20 to 40 days old) usually become foragers. Foraging bees typically occupy one of three roles: active foragers, scout foragers and inactive foragers. Active foraging bees travel to a food source, examine neighbor food sources, gather food and return to the hive. Scout bees investigate the area surrounding the hive, often a region of up to 50 square miles, looking for

attractive new food sources. Roughly 10 percent of foraging bees in a hive are employed as scouts. At any given time some of the foraging bees are inactive. These inactive foragers wait near the hive entrance. When active foragers and scouts return to the hive, depending on the quality of the food source they've just visited, they may perform a waggle dance to the waiting inactive bees. There's strong evidence that this waggle dance conveys information to the inactive bees about the location and quality of the food source. Inactive foragers receive this food source information from the waggle dance and may become active foragers. In general, an active foraging bee continues gathering food from a particular food source until that food source is exhausted, at which time the bee becomes an inactive forager [24]. Cloud task scheduling is one of the most widely studied problems in computer science research. Cloud task scheduling can be defined by the following question: Given a set of jobs, a set of VMs, a set of constraints, and an objective function, how should jobs be allocated to resources?

In this section, task scheduling based ABC algorithm will be proposed. Decreasing the makespan of tasks is the basic goal (objective function) from the proposed method. The number of possible solutions to assign 20 tasks on 20 VMs assuming that each VM execute only one task, just computes 20!. There are a total of 20!=2,432,902,008,176,640,000 possible solution. Even if having a computer that could evaluate 1 million solutions per second, examining all 20! Of possible solutions would require more than 77,000 years. This kind of extremely difficult combinatorial optimization problem is the type of problem that ABC algorithms are well suited to handle.

In ABC algorithm, there are three types of bees: active, inactive and scout. The counts of each of these types of bees are determined in the initial phase of ABC algorithm. In the first, ABC algorithm creates a collection of bees, each of which has a random solution. The ABC algorithm tracks the overall best solution found by any of the bees in the ABC and the best solution here is associated the Makespan length. Iterations are indexed by t and $1 < t < t_{max}$, where t_{max} is the maximum number of iterations allowed. Each iteration, each bee type is processed by the appropriate method. The t_{max} is used to control how many times the solve routine will iterate; this is necessary because in ABC algorithm scenarios you usually don't know when you've reached an optimal solution—if you know the optimal solution, you really don't have a problem to solve. In this case, the value of t_{max} was limited to 100 only so that the ABC algorithm did not produce the optimal result. The point here is that although ABC algorithms may produce an optimal result, you usually have no way of knowing this and so you must be willing to accept a "good" result.

The pseudo code of the proposed ABC procedure is shown in fig. 1.

***Input**: List of Cloudlet (Tasks) and List of VMs*
***Output**: the best solution for taks alloction on VMs*
***Steps**:*
1. ***Initialize**:*
***Set value of parameters** Number_of_Bees, Number_of_Inactive, Number_of_active, Number_of_Scout, Max_number_of_Vists, Prob_Mistake, Prob_Presuasion, tmax.*
***Set** Current_iteration_t=1.*
***Set** Global_Best_Solution=null.*
2. ***Generate** Random Solution for each bee*
3. ***Update** Current_Best_Solution*
4. ***For** k :=1 to Number_of_Bees*
***IF** k is an ActiveBee*
***Perform** ProcessActiveBee()*
***ElseIF** k is an ScoutBee*
***Perform** ProcessScoutBee()*
Else
***Perform** ProcessInactiveBee ()*
End IF
5. ***Increment** Current_iteration_t by one.*
6. ***If** (Current_iteration_t < t_{max})*
***Goto** step 4*
Else
***Print** Global_Best_solution.*
***End** If*
Stop

Fig. 1. Pseudo code of ABC procedure

In an initialization phase of the proposed algorithm, the parameters are initialized. The value of the Number_of_Bees variable could be derived from the other three variables (*Number_of_Inactive*, *Number_of_active*, *Number_of_Scout*), but the extra variable improves Pseudo readability here. The total number of bees (*Number_of_Bees*) will depend on the problem nature. More bees are better but slow the program's execution. *Max_number_of_Vists*, *Prob_Mistake* and *Prob_Presuasion* variable will be explained shortly. *Current_Best_solution* variable is null and *Current_iteration_t* will be 1 to indicate that the algorithm will be in the fisrt iteration (cycle). Each bee generates its random solution. And the *Global_Best_Solution* variable is update (equal the best solution from the founded solution from all bees). The iterative phase simulates the behavior of foraging bees to solve a problem. Each cycle, the bees are iterated using for loop and each Bee object is processed by the appropriate helper module. The iterative

phase is moderately complex and uses three helper modules (methods), ProcessActiveBee(), ProcessScoutBee() and ProcessInactiveBee(). The ProcessActiveBee() module is the heart of an SBC algorithm and is the most complex module in terms of branching. The ProcessActiveBee() module is presented in Fig. 2.

```
ProcessActiveBee()
  Generate a neighbor Solution.
  Generate a random number between zero to one in prob
  If (a neighbor solution quality > current bee solution quality)
      If (prob< Prob_Mistake)
          Increase numberOfVisits by one
      Else
          bee accepts the better neighbor solution
          Reset  numberOfVisits to zero
          Update Current_Best_Solution
          Perform DoWaggleDance()
      End If
 End If
 If (a neighbor solution quality < current bee solution quality)
      If (prob< Prob_Mistake)
          bee accepts the better neighbor solution
          Reset  numberOfVisits to zero
          Update Current_Best_Solution
          Perform DoWaggleDance()
       Else
          Increase numberOfVisits by one
      End If
  If (numberOfVisits> Max_number_of_Vists)
      Change the status of this active bee to be inactive
      Selected randomly inactive bee is to be active
Return
```

Fig. 2. Pseudo code of ProcessActiveBee()

The active bee in ProcessActiveBee() module, first obtains a neighbor solution relative to its current solution. A key concept in ABC algorithms is the idea that each virtual food source that represents a solution has some sort of neighbor. In the case of the cloud task scheduling problem, where a solution can be represented as an array of VM's IDs representing the order of VMs (the first task will be mapped to the first VM in the array of VM's IDs and so on), a natural neighbor solution relative to a current solution is a permutation of the current solution where two adjacent VM's IDs have been exchanged. The prob variable has a value between 0.0 and 1.0 and will be compared against the *prob_Mistake* field value to determine if the bee makes a mistake in evaluating the neighbor solution—that is, rejects a better neighbor solution or accepts a worse neighbor solution. Quality of solution here is related to the Makespan of the solution. The more quality means low Makespan time of solution and less quality means high Makespan time of the solution. The expected Makespan of the solution is computed by (1).

$$EM = \arg\max_{j \in J} \{sum_{i \in IJ}(d_{ij})\} \quad (1)$$

Where, *EM* is the expected Makespan, *IJ* is the set of tasks that assigned to the VM_j and d_{ij} which expresses the expected execution time and transfer time of the $task_i$ on VM_j can be computed with (2).

$$d_{ij} = \frac{TL\_Task_i}{Pe\_num_j * Pe\_mips_j} + \frac{InputFileSize}{VM\_bw_j} \quad (2)$$

Where TL_Task_i is the total length of the task that has been submitted to VM_j, Pe_num_j is the number of VM_j processors, Pe_mips_j is the MIPS of each processor of VM_j, *InputFileSize* is the length of the task before execution and VM_bw_j is the communication bandwidth ability of the VM_j.

If the current active bee finds a better neighbor solution, the algorithm determines if the bee makes a mistake and rejects the better neighbor or the bee accepts the better neighbor (update its current solution to equal the neighbor solution). Similarly, if the current bee did not find a better neighbor solution, the algorithm determines whether the bee makes a mistake and accepts the worse neighbor solution or does not make a mistake and rejects the neighbor. Notice that there are two different types of mistakes possible, but that both types of mistakes have the same probability, *prob_Mistake* = 0.01. If this active bee solution was updated,

the new solution is checked to see if it will be a new *Global_Best_Solution* or not and DoWaggleDance () module is called to simulate the bee returning to the hive and conveying information about the new food source to the inactive bees.

Max_number_of_Vists is a threshold value used to prevent a bee from staying too long at a particular solution. In every iteration, if a bee does not find a neighbor food source with better quality, the bee's *numberOfVisits* counter is incremented. After the current active bee accepts or rejects the neighbor solution, the algorithm checks if the *numberOfVisits* counter in a Bee object exceeds the *Max_number_of_Vists* threshold value. If so, the current bee's status is converted to inactive and a randomly selected inactive bee is converted to be active. The DoWaggleDance() module simulates an active or scout bee returning to the hive and then performing a waggle dance to inactive bees in order to convey information about the location and quality of a food source.

```
DoWaggleDance ()
  For i :=1 to Number_of_Inactive
     If (Dancing bee solution quality > current inactive bee solution quality)
        Generate a random number between zero to one in prob2
       If (Prob_Presuasion> prob2)
         Current inactive bee accepts the solution of Dancing bee
       End If
     End If
Return
```

Fig. 3. Pseudo code of DoWaggleDance ()

The DoWaggleDance() module is presented in Fig. 3. The quality of the current bee's solution (dancing bee) is compared against the quality of each inactive bee solution. If the current bee's solution is better and the current inactive bee is persuaded (with probability probPersuasion = 0.90), the current bee's solution is copied over the inactive bee's solution (The inactive bee updates its solution to equal dancing bee's solution).

The ProcessScoutBee() module used by the solve module simulates the action of a scout bee randomly searching for appealing food sources. The ProcessScoutBee() module is presented in Fig. 4. A scout bee generates a random solution, checks if the random solution is better than its solution. If so, random solution is copied over the scout bee's solution. Recalls that high quality values (smaller Makespan time) are better. If the scout bee has found a better solution, the algorithm checks to see if the new solution is a global best solution. Note that unlike active bees, scout bees never make mistakes evaluating the quality of a food source. In ABC algorithm inactive bees are exactly that inactive so the ProcessInactiveBee() module is empty.

```
ProcessScoutBee()
  Generate a random Solution.
  If (random solution quality > scout bee solution quality)
        Scout bee accepts the random solution
        Update Current_Best_Solution
        Perform DoWaggleDance()
   End If
Return
```

Fig. 4. Pseudo code of ProcessScoutBee()

IV. COMPUTATION COMPLEXITY ANALYSIS

The computational complexity of implementing ABC as a scheduling function of the number of tasks and the number of VMs based on the pseudo code presented in Fig. 1, is O(Block1 + Block2).

Block1 represents the initialization phase of bees from steps 1 to 3 in Fig. 1. It has a computational complexity of O(b), where b is number of bees.

Block2 is the iterative phase from steps 4 to 6 and has a computational complexity of O($t_{max} \times b \times n \times w$), where tmax is the number of iterations. b is number of bees, n is the number of batch tasks and w is the number of inactive bees.

Finally, the overall complexity is O($b + t_{max} \times b \times n \times w$) and can be simplified to O($t_{max} \times b \times n \times w$), ignoring the constants and lower order terms.

V. IMPLEMENTATION & EXPERIMENTAL RESULTS

A. Parameters Setting of Cloud Simulator

The experiment is implemented with 10 Datacenters with 50 VMs and 50-1000 tasks under the simulation platform. The length of the task is from 1000 MI (Million Instructions) to 20000 MI. The parameters setting of cloud simulator are shown in Table 1.

TABLE I. PARAMETERS SETTING OF CLOUDSIM

Entity Type	Parameters	Value
Task (cloudlet)	Length of task	1000-20000
	Total number of task	100-1000
Virtual Machine	Total number of VMs	50
	MIPS	500-2000
	VM memory(RAM)	128-2048
	Bandwidth	500-1000
	cloudlet Scheduler	Space_shared and Time_shared
	Number of PEs requirement	1-4
Datacenter	Number of Datacenter	10
	Number of Host	2-6
	VmScheduler	Space_shared and Time_shared

B. ABC Parameters evalution and setting

Sensitive parameters that must be fine-tuned for each problem include the number of each type of bee, the maximum number of visits before a food source is exhausted, inactive bee persuasion probability, active bee mistake probability and t_{max}. Table 2 shows the selected best parameters of ABC.

TABLE I. SELECTED PARAMETERS OF ABC

Parameter	Value
Number_of_Bees	100
Number_of_active	75
Number_of_Scout	15
Number_of_Inactive	10
Max_number_of_Vists	70
Prob_Mistake	.01
Prob_Presuasion	.90
t_{max}	100

The cloud task scheduling algorithms to be compared in the experiments include ACO in [7], FPLTF algorithm in [8], FCFS in [9] and the proposed algorithm. Table 3 shows the selected best parameters of ACO algorithm are as in [7].

TABLE II. SELECTED PARAMETERS OF ACO

Parameter	Value
α	.3
β	1
ρ	.4
Q	100
m	10
t_{max}	100

C. Implementation results of ABC, ACO, FPLTF and RR

The following experiments, we compared the average makespan with different tasks set. The average makespan of the ABC, ACO, FPLTF and FCFS algorithms are shown in Fig.5. It can be seen that, with the increase of the quantity task, ABC takes the time less than FPLTF and FCFS algorithms. Although ABC performance is very near from ACO, ABC outperformed ACO in all instances of tasks. This indicates that the proposed algorithm take less time to execute than other methods because it combines intelligently different concepts for exploring the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions. That is the reason why proposed algorithm takes less time to execute jobs.

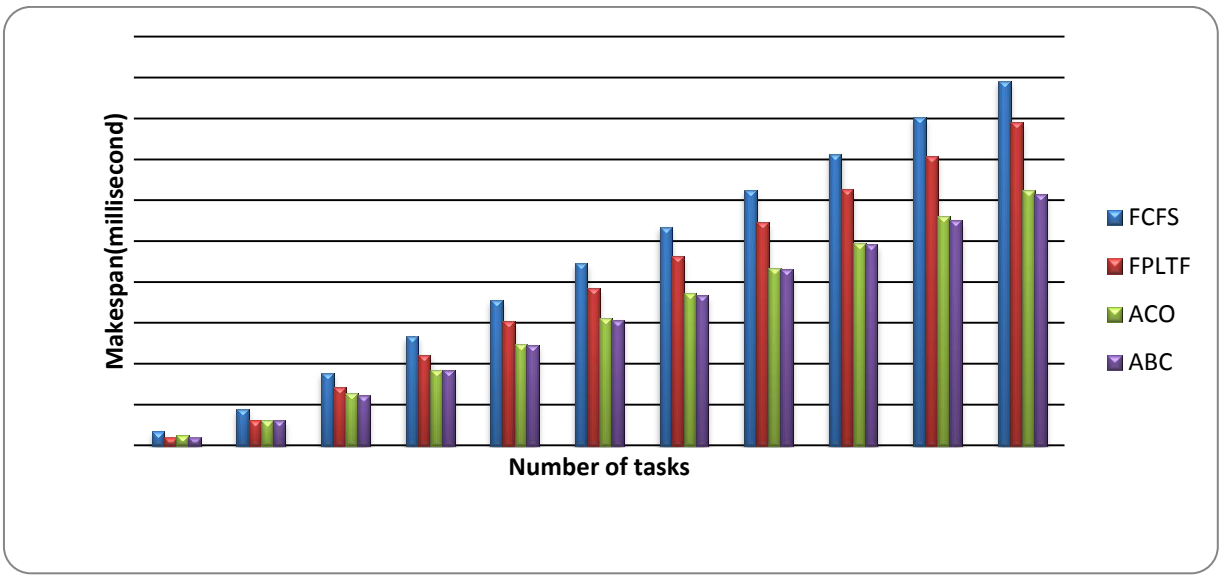

Fig. 5. Average makespan of ABC, ACO, FPLTF and FCFS

The Degree of Imbalance (DI) measures the imbalance among VMs, which is computed by (3) and (4).

$$T_i = \frac{TL\_Tasks}{Pe\_num_j \times Pe\_mips_j} \tag{3}$$

Where, *TL_Tasks* is the total length of tasks which are submitted to the VM_i.

$$DI = \frac{T_{\max} - T_{\min}}{T_{avg}} \tag{4}$$

Where T_{max}, T_{min} and T_{avg} are the maximum, minimum and average T_i respectively among all VMs [7]. The small value of DI tells that the load of the system is more balanced. The average degree of imbalance of each algorithm with the number of tasks varying from 100 to 1000 is shown in Fig.6. It can be seen from Fig. 6 that the ABC and ACO can achieve better system load balance than FPLTF and FCFS algorithms.

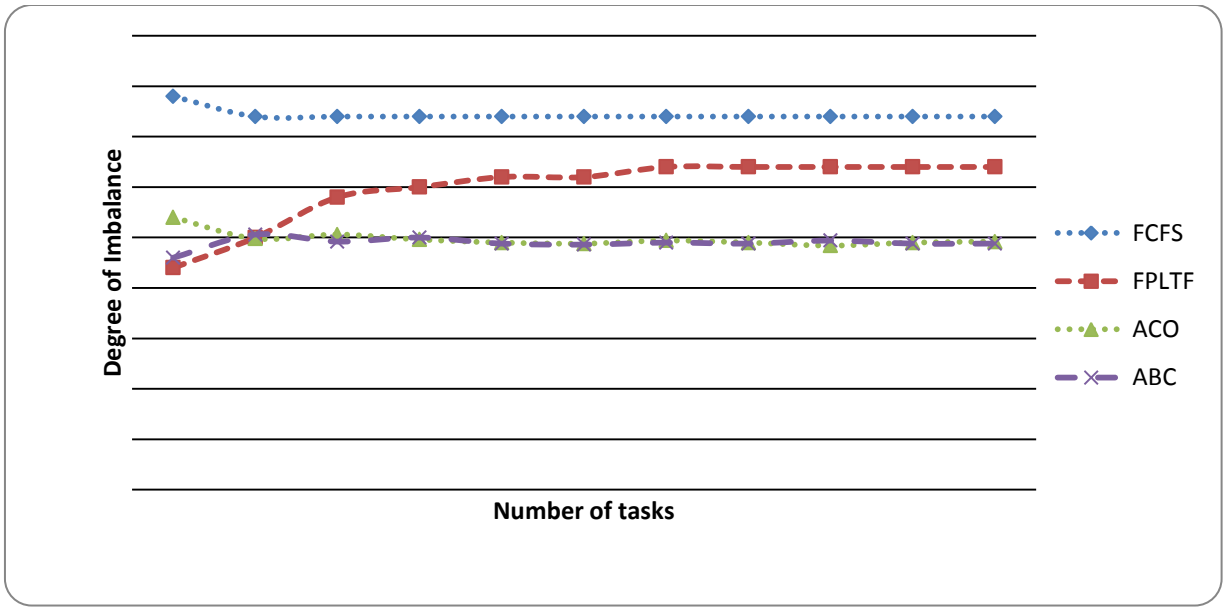

Fig. 6. Average degree of imbalance of ABC, ACO, FPLTF and FCFS

The degree of imbalance can be measured also using standard deviation that is given by (5).

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \text{for all i} \in \text{VM list} \qquad (5)$$

Where σ is the standard deviation with the same unit as load (%), N is the number of virtual machines and x_i is the finishing time of VM_i. if the standard deviation value of a method is small, it means that the difference of each load is small. The standard deviations for each algorithm are shown in Fig. 7. It can be seen from the Fig. 7 that the proposed ABC algorithm can achieve good system load balance.

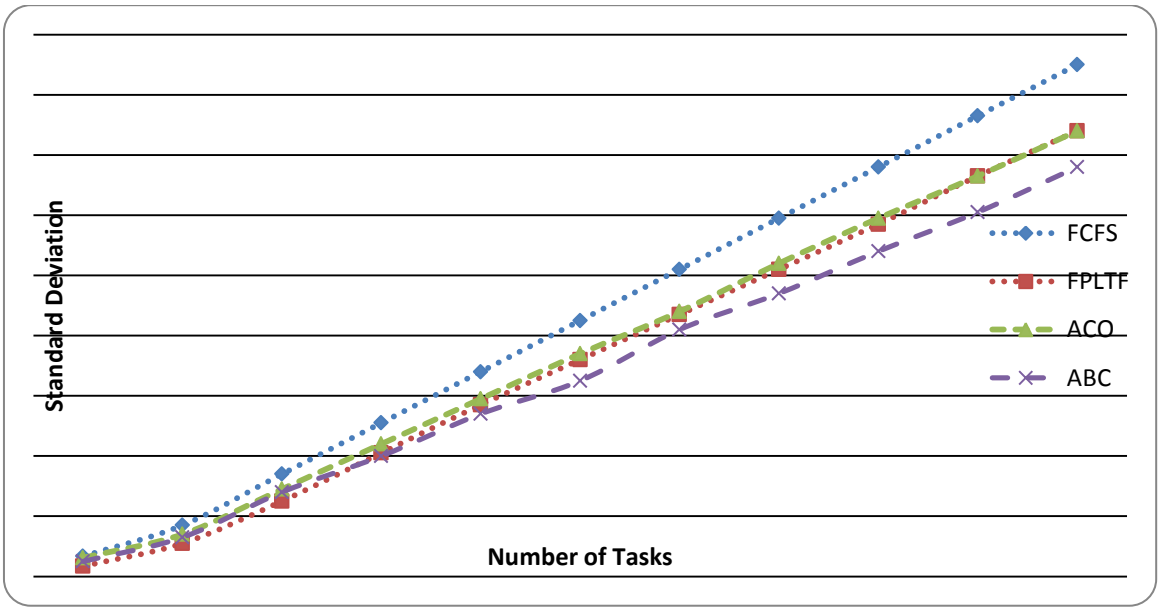

Fig. 7. Standard deviation of ABC, ACO, FPLTF and FCFS

VI. CONCLUSIONS AND FUTURE WORK

In this paper ABC algorithm for achieving cloud computing tasks scheduling has been proposed. Firstly the best values of parameters for ABC algorithm, experimentally determined. Then the ABC algorithm in applications with the number of tasks varying from 50 to 1000 evaluated. Simulation results demonstrate that ABC algorithm outperforms ACO, FPLTF and FCFS algorithms. In future work the effect load balancing will be considered. Also the comparison between the proposed approach and other meta-heuristics approaches will be performed.

REFERENCES

[1] Paul, M., Sanyal, G., "Survey and analysis of optimal scheduling strategies in cloud environment", IEEE International Conference on Information and Communication Technologies (WICT), pp. 789 – 792, 2012

[2] Kun Li, Gaochao Xu, Guangyu Zhao, Yushuang Dong, Dan Wang, "Cloud Task scheduling based on Load Balancing Ant Colony Optimization", Chinagrid Conference (ChinaGrid), pp.3– 9, 2011

[3] F. Chang, J. Ren, and R. Viswanathan, "Optimal Resource Allocation in Clouds" in 2010 IEEE 3rd International Conference on Cloud Computing, pp.418-425, 2010.

[4] Arabi E. keshk, Ashraf El-Sisi, Medhat A. Tawfeek, F. A. Torkey, " Intelligent Strategy of Task Scheduling in Cloud Computing for Load Balancing ", International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), Vol. 3, pp.12-23,2013

[5] C. S. Chong, M. Y. H. Low, A. I. Sivakumar, and K. L. Gay, "A bee colony optimization algorithm to job shop scheduling," in Proceedings of the Winter Simulation Conference, 2006. pp. 1954-1961, 2006.

[6] Dhinesh Babu L.D , P. Venkata Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments", Applied Soft Computing , Vol. 13, pp. 2292–2303, 2013

[7] Medhat A. Tawfeek, Ashraf El-Sisi, Arabi E. keshk and Fawzy A. Torkey, " Cloud Task Scheduling Based on Ant Colony Optimization", on the 8th International Conference on Computer Engineering & Systems ICCES, , pp.64-68, 2013.

[8] D. Saha, D. Menasce, S. Porto, Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures, Journal of Parallel and Distributed Computing, Vol.28, PP. 1-18, 1995

[9] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, 7th edition, JohnWiley & Sons, 2005.

[10] Y. Gao et al., "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing" J. Comput. System Sci. (2013) http://dx.doi.org/10.1016/j.jcss.2013.02.004

[11] Christian Blum , Andrea Roli, "Metaheuristics in combinatorial optimization: overview and conceptual comparison," ACM Computing Surveys journal Vol. 35, No. 3, pp. 268-308, 2003.

[12] C. R. Reeves, editor. "Modern Heuristic Techniques for Combinatorial Problems" Blackwell Scientific Publishing, Oxford, England, 1993.

[13] G. L. Nemhauser and A. L. Wolsey. "Integer and Combinatorial Optimization" John Wiley & Sons, New York, 1988.

[14] Buyya, R., Ranjan, R., Calheiros, R.N., "Modeling and simulation of scalable cloud computing environments and the cloudSim toolkit: challenges and opportunities," in Proceedings of the 7th High Performance Computing and Simulation Conference, Leipzig, Germany, 2009.

[15] Ghalem, B., Fatima Zohra, T., and Wieme, Z. "Approaches to Improve the Resources Management in the Simulator CloudSim" in ICICA 2010, LNCS 6377, pp. 189–196, 2010.

[16] Chenhong Zhao, Shanshan Zhang, Qingfeng Liu, Jian Xie, Jicheng Hu, "Independent Tasks Scheduling Based on Genetic Algorithm in Cloud Computing", IEEE International Conference on Wireless Communications, Networking and Mobile Computing, PP. 1 – 4, 2009

[17] Kai Zhu, Huaguang Song, Lijing Liu, Jinzhu Gao, Guojian Cheng, "Hybrid Genetic Algorithm for Cloud Computing Applications", IEEE International Conference on Asia-Pacific Services Computing Conference (APSCC), PP. 182 – 187, 2011

[18] T.D. Seeley, The Wisdom of the Hive, Harvard University Press, Cambridge, MA, 1995.

[19] N. Karaboga, M.B.C. etinkaya, A novel and efficient algorithm for adaptive filtering: artificial bee colony algorithm, Turkish Journal of Electrical Engineering & Computer Sciences, Vol. 19,pp. 175–190, 2011.

[20] E. Cuevas, D. Zaldívar, M. Pérez-Cisneros, H. Sossa, V. Osuna, Block matching algorithm for motion estimation based on Artificial Bee Colony (ABC), Applied Soft Computing, Vol. 13, no. 6, Pages 3047–3059, 2013.

[21] S. Nakrani and C. Tovey, "On honey bees and dynamic server allocation in Internet hosting centers," Adaptive Behavior, Vol. 12, no. 3-4, pp.223-240, 2004.

[22] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey, 2005.

[23] D. Karaboga, B. Basturk, "the performance of artificial bee colony (ABC) algorithm, " Applied Soft Computing, Vol. 8, pp. 687–697 , 2008.

[24] Li-Pei Wong, Low, M.Y.H., Chin Soon Chong, "Bee Colony Optimization with local search for traveling salesman problem, " Industrial Informatics, pp. 1019 – 1025, 2008.