Resource Allocation in Cloud Computing Using Imperialist Competitive Algorithm with Reliability Approach

Maryam Fayazi
Department of Computer, Ahvaz Branch, Islamic Azad University
Department of Computer, Khouzestan Science and Research Branch, Islamic Azad University
Ahvaz, Iran

MohammadReza Noorimehr
Department of Computer, Ahvaz Branch, Islamic Azad University
Department of Computer, Khouzestan Science and Research Branch, Islamic Azad University
Ahvaz, Iran

Sayed Enayatollah Alavi
Department of Computer Engineering, Faculty Engineering, Shahid Chamran University of Ahvaz
Ahvaz, Iran

***Abstract*—Cloud computing has become a universal trend now. So, for users, the reliability is an effective factor to use this technology. In addition, users prefer to implement and get their work done quickly. This paper takes into account these two parameters for resource allocation due to their importance In this method, the Imperialist Competitive algorithm with the addition of a cross layer of cloud architecture to reliability evaluation is used. In this cross layer, initial reliability is considered for all the resources and the implementation of their tasks and due to the success or failure of implementation, reliability of resources is increased or reduced. Reliability and makespan are used as a cost function in ICA for resource allocation. Results show that the proposed method can search the problem space in a better manner and give a better performance when compared to other methods.**

Keywords*—Imperialist Competitive algorithm; Reliability; makespan; Cloud Computing

I. INTRODUCTION

Research on cloud computing is growing rapidly. Cloud computing means developing and applying computer technology on the Internet. Cloud computing is a network for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction[1]. In cloud computing, the end users have unlimited access to the resources and only pay for the resources they consume. Cloud computing basics is that the user data is not stored locally, but is stored in the internet data center. Companies that provide cloud computing services can manage and maintain the data centers [2, 3].

To provide services to end users, the cloud computing environment needs to be reliable and also well managed in such a way that it gives throughput in the lowest time. So, reliability and task scheduling are two important parameters. In this paper, considering reliability, the allocation of makespan resources was done using Imperialist Competitive algorithm and a cross layer in cloud architecture.

This paper is organized as follows. Section 2 explains some related works. Section 3 is dedicated to problem description and some meta-heuristic algorithms related to problems such as genetic algorithm and imperialist Competitive algorithm. Section 4 is a discussion about proposed approach that uses imperialist Competitive algorithm in it. Section 5 shows simulation evaluations and comparison to genetic algorithm. Section 6 is the conclusion.

II. RELATED WORK

Many heuristic and meta-heuristic methods are prevented by different researchers for scheduling and resource allocation to tasks in cloud. The heuristic approach uses the concept of prioritization to schedule tasks. Algorithms such as max-min and min-min are part of the heuristic-based approach that divide scheduling to two steps: prioritizing the task and source selection. In priority task, each task is assigned a rating based on its priorities. In step source selection, a high-priority task is selected and is scheduled on the optimized resources that have completed the previous tasks. The meta-heuristic approach includes scheduling algorithms based on repeatable method to find the optimal solution. They provide an efficient way of moving quickly towards a very good solution. Many meta-heuristic approaches have been applied for solving workflow scheduling problems, including Genetic Algorithms and Imperialist Competitive Algorithm (ICA) [4].

One of important problems in cloud computing is makespan constraint. Most scheduling algorithms focus on makespan. In [5]-[10], Genetic Algorithm is used to reduce makespan of tasks allocation to resources. Yue Miao [11] by using firefly algorithm based on chaos algorithm tried to reduce the average time spent by subtasks in processing request tasks, and thus improve the efficiency of task processing and achieve a rational allocation of resources. Mizan et al. [12] proposed a modified task scheduling algorithm based on the concept of Bees life algorithm and greedy algorithm to gain optimistic value of service in hybrid cloud. The main idea of the system is to achieve an affirmative response from the end users and utilize the resources in a very transient manner.

Seidgar et al. [13] describe an approach incorporating simulation with imperialist competitive algorithm for the scheduling purpose having machine breakdowns and preventive maintenance activities. The objective is to minimize the makespan.

In addition of makespan, reliability function is a very important function for users [14]-[20].

Zhao et al. [14] proposed a DRR (Deadline, Reliability, Resource-aware) scheduling algorithm, which schedules the tasks such that all the jobs can be completed before the deadline, ensuring the Reliability and minimization of resources. Gartner [15] used a formal approach to define important terms like fault, fault tolerance, and redundancy. This leads to four distinct forms of fault tolerance and two main phases in achieving them: detection and correction. It shows that this can help to reveal inherently fundamental structures that contribute to understanding and unifying methods and terminology. By doing this, it surveys many existing methodologies and discuss their relations. The underlying system model is the close-to-reality asynchronous message-passing model of distributed computing. Zhang et al. [16] present BFT Cloud (Byzantine Fault Tolerant Cloud), a Byzantine fault tolerance framework for building robust systems involuntary-resource cloud environments. BFT Cloud guarantees robustness of systems when up to f of totally 3f+1 resource providers are faulty, including crash faults, arbitrary behaviors faults, etc. BFT Cloud is evaluated in a large-scale real-world experiment which consists of 257 voluntary-resource providers located in 26 countries. The experimental results shows that BFT Cloud guarantees high reliability of systems built on the top of voluntary-resource cloud infrastructure and ensures good performance of these systems. In [17], the different techniques of fault tolerance are presented. The main focus is on types of faults occurring in the system, fault detection, and recovery techniques. In [18], the existing fault tolerance techniques in cloud computing are discussed based on their policies, tools used, and research challenges. Cloud virtualized system architecture has been proposed. In the proposed system, autonomic fault tolerance has been implemented. The experimental results demonstrate that the proposed system can deal with various software faults for server applications in a cloud virtualized environment. Jhawar et al. [19] introduced an innovative, system-level, modular perspective on creating and managing fault tolerance in Clouds. They proposed a comprehensive high-level approach to shading the implementation details of the fault tolerance techniques to application developers and users by means of a dedicated service layer. In particular, the service layer allows the user to specify and apply the desired level of fault tolerance, and does not require knowledge about the fault tolerance techniques that are available in the envisioned Cloud and their implementations. In [20], a fault tolerance model for cloud computing is given and the Paper describes a model for Fault Tolerance in Cloud computing (FTMC). FTMC model tolerates the faults on the basis of reliability of each computing node. A Computing node is selected for computation on the basis of its reliability and can be removed, if it does not perform well for applications.

To increase the Quality of Service in cloud, it is necessary to implement a scheduling algorithm that in addition to makespan, considers the reliability of cloud resources during resource allocation. Recently in [21], makespan and reliability were discussed. A genetic algorithm has been proposed that schedules workflow applications in unreliable cloud environment and meets user defined QoS constraints. A budget constrained time minimization genetic algorithm has been proposed which reduces the failure rate and makespan of workflow applications. It allocates those resources to workflow application which are reliable. So, here with this motivation, work is done that reduces makespan with Imperialist Competitive algorithm to provide reliable machines for implementation of tasks.

III. BACKGROUND

A. Problem Description

Earlier, the Cloud data center was composed of thousands of servers and hundreds of switches connecting the servers. Each server can host for tens of virtual machines [22]. From scheduling algorithms, it is expected to find a plan for each task in set T with desired quality of service constraints for users. The schedule should be one that reduces the failure rate and makespan.

B. Imperialist Competitive Algorithm

Imperialist Competitive Algorithm is defined as the optimization strategy based on social and political evolution of humans. The Main Basics of this algorithm are Assimilation, Imperialistic Competitive, and Revolution. For more accurate algorithm inspired social and political phenomenon of colonialism [23]. The Pseudo code for the algorithm is as follows:

1) Select some random points and initialize the empires.

2) Move the colonies toward their relevant imperialist (Assimilating).

3) If there is a colony in an empire which has lower cost than that of imperialist, exchange the Positions of that imperialist and the colony.

4) Compute the total cost of an empire (Related to the power of both imperialist and its colonies).

5) Pick the weakest colony from the weakest empire and give it to the empire that has the most likelihood to possess it (Imperialistic Competitive).

6) Eliminate the powerless empires.

7) If there is just one empire, stop if not go to 2.

In step 1, randomly some points are selected and some Empires are formed. Then, according to equations (1), (2), and considering N.C. in equation (3), for each empire, the same number of initial colonial countries are selected, randomly and given to n^{th} Empire.

$$C_n = \max_i\{c_i\} - c_n \quad (1)$$

C_n : N^{th} Empire normalized cost

c_n : N^{th} Empire cost

$\max_i\{c_i\}$: Max cost between Empires

$$P_n = \left| \frac{c_n}{\sum_{i=1}^{N_{imp}} c_i} \right| \quad (2)$$

P_n : N^{th} Empire normalized power

N_{imp} : Number of Empires

N.C (Number of colonies of each Empire) = round (P_n * N_{col}) (3)

N_{col} : total number of colonies

In step 2, Empires with the follow-up policy of assimilation efforts are selected to attract their colonies. In line with this policy, the colonial country moves the size of x units to the Empire and get a new position. X is a random number with uniform distribution (or any other suitable distribution).

In step 3, it is possible that the colony moves to the empire get better position than the empire (lower cost). In this case, empire and colony exchange and the algorithm continues with the empire country in a new position.

In step 4, according to formula 4, the power of an empire is equal to the central government plus a small percentage of the power of its colonies.

$$T.C_n. = \text{Cost (empire}_n) + \xi \text{ mean } \{\text{Cost (colonies of empire}_n)\} \quad (4)$$

In equation (4), $T.C_n.$ is the total cost of n^{th} Empire and ξ is a positive number that is considered usually between zero and one, and close to zero. Little consideration of ξ, makes the total cost of an empire, almost equal to the cost of the central government (the empire) and also increasing ξ increases the effect of the colonies cost of that empire in determining the total cost.

The total cost of empire is obtained by the following equation (5):

$$N.T.C_n. = \max_i\{T.C_i\} - T.C_n. \quad (5)$$

In equation (5), $T.C_n.$ is total cost of n^{th} empire and $N.T.C_n.$ is total normalized cost of that empire. Each empire with less $T.C_n.$, will be more $N.T.C_n.$ In fact, the empire with min cost has max power.

In step 5, there is a competition between empires on taking over the weakest colony of the weakest empire.

By using the total normalized cost, the possibility (power) of takeover of Competitive colony by each empire, is calculated as in equation (6):

$$P_{p_n} = \left| \frac{N.T.C_n}{\sum_{i=1}^{N_{imp}} N.T.c_i} \right| \quad (6)$$

With the possibility of taking of the empire, for dividing these colonies between empires randomly but with probability of related to possibility of taking of empire; vector p is formed from the above probability values (equation (7)):

$$\mathbf{P} = \left[p_{p_1}, p_{p_2}, p_{p_3}, ..., p_{p_{N_{imp}}} \right] \quad (7)$$

Vector p in equation (7) has a size of $1*N_{imp}$ and is made from possibility takeover empires values. Then, the random vector R, is formed in the same size of vector p. arrays of this vector are random numbers with uniform distribution in the interval [0, 1]. Then, vector D is made from the following equation (8):

$$\mathbf{D} = \mathbf{P} - \mathbf{R} = \left[D_1, D_2, D_3, ..., D_{N_{imp}} \right] = \left[p_{p_1} - r_1, p_{p_2} - r_2, p_{p_3} - r_3, ..., p_{p_{N_{imp}}} - r_{N_{imp}} \right] \quad (8)$$

By using vector D, those colonies where the index in vector D is larger than others are given to the empire. In step 6, the empire that doesn't have any colony, is eliminated to be a colony.

IV. Proposed Method

This section discusses the proposed approach that uses Imperialist Competitive algorithm and cross layer for evaluating the reliablity of virtual machines.

A. Evaluation Reliability Algorithm

In [20] a method is used to increase fault tolerance and this method is applied as cross layer in cloud architecture to evaluate the relaiability of virtual machines and then the VM with a high reliability is selected. Reliability evaluation algorithm is applied on each VM. At first, each VM reliability will be set to 1. There is an adjustment factor N that controls reliability evaluation. N value is always greater than 0. The algorithm gets the RF factor from input. RF is a reliability factor that increases or decreases resource reliability. RF value depends on user decision. Reliability should be between 0, 1. If task implementation by resource is done before defined deadline, it is successful and it's reliability will be increased. Otherwise it is a failure and it's reliability will be reduced. Pseudo-code of reliability evaluation algorithm is shown in Figure 1. Table I gives a sample of VM reliability evaluation.

```
1 Start
2 Initialized Reliability: =1, N: =1;
3 Input RF;
4 Input processing node status;
5 If processing node Status =Pass then
6 Reliability: = Reliability + (Reliability * RF);
7 If N > 1 then
8 N: = N-1;
9 If Reliability > 1 then
10 Reliability: = 1;
11 Else if processing node Status = Fail then
12 Reliability: = Reliability – (Reliability * RF * N);
13 N: = N+1;
14 If Reliability < 0 then
15 Reliability: = 0;
16 End
```

Fig. 1. Pseudo-code of reliability evaluation algorithm

TABLE I. SAMPLE OF VM RELIABILITY EVALUATION

Time	VM1	Reliability
1	Fail	0.95
2	Fail	0.9
3	Success	0.945
4	Success	0.99
5	Fail	0.94

B. Imperialist Competitive algorithm

To achieve the best allocation of resources in cloud, the Imperialist Competitive algorithm as follows:

1) Problem modeling: in this method, n different tasks and m virtual machines are considered and also Ti means i^{th} task and VMj means j^{th} virtual machines. It is done by using identification number of each virtual machine and the task. In Figure 1, there are 10 tasks and each task has a unique identification number from 0 to 5 and there are 3 virtual machines each having unique identification number from 0 to 2. In Imperialist Competitive algorithm, each country represents a mapping of tasks to available resources on the issue of scheduling tasks. We are looking for the best possible sequence of tasks to allocate to resources. In ICA, position of each country represents a solution for a problem that shows mapping tasks on resources. Figure 2 shows sample of mapping tasks to resources by using identification number of tasks and virtual machines.

Tasks	T0	T1	T2	T3	T4	T5
Virtual Machines	VM1	VM2	VM0	VM2	VM0	VM1

Fig. 2. Countries as resource allocation

2) Initial Population: In initial population, countries are generated randomly. In this problem, each country is an array with $2*N_t$ length that first row is tasks and second row is resources and The task t is selected from set T that has not been allocated to any virtual machine. Then a virtual machine is selected randomly from set VM. This process continues until countries are generated. The number of countries are generated in initial population according to the size of initial population. Then, number of initial empires and colonies is calculated from follow equations (9), (10):

$$\text{Initial empires} = \text{Countries}* n \quad (9)$$

$$\text{All Colonies} = \text{Countries} - \text{Initial empires} \quad (10)$$

In equation (10), n is a percentage of total number of countries that in the proposed algorithm is considered to be 0.1. In this paper, initial population size and number of empires are 100, 10, respectively.

3) Evaluation: in this step, best country is found. So, fitness value is calculated based on makespan and reliability. A country having low makespan and high reliability will have low cost value compared to other countries. In the proposed method, makespan must be minimized. Makespan and task execution time are calculated respectively in equations (11), (12).

$$\text{Makespan M (I)} = \sum_{i=0}^{n} T_{et_i} \quad (11)$$

$$T_{et} = (\text{length of task}) / (\text{MIPS of virtual machine}) \quad (12)$$

In this method, in addition to reduced makespan, reliability also will increase. So, the solution giving a low makespan and high reliability will be selected. Reliability is obtained from cross layer.

So, cost function in this model is calculated from the following equation (13):

$$\text{Cost} = c1*(1-R) + c2* M \quad (13)$$

$$\text{Reliability R (I)} = \text{mean (R (VMt1), R (VMt2)} \ldots \text{R (VMtn))} \quad (14)$$

In the equation (13), M is makespan of countries and R is mean reliability of resources in countries obtained from equation (14). In equation (14), R (VMt1) is the reliability of vm where first task is to be executed, R (VMt2) is the reliability of vm where second task is to be executed, and so on. R (VMtn) is the reliability of vm where nth task is to be executed. c1 and c2 values are coefficients to prioritize the order of importance makespan or reliability factors and sum of these values must be 1. These values are determined by the user.

4) Initial empires generation: some countries with highest power are considered as empire and other countries as colonies. For this, countries are ordered as ascending on basis of their cost value. So, countries in the beginning of the array will have lower cost. Normalized cost and power of empires and number of their colonies is obtained from equations (1), (2), (3).

5) Assimilation: In this step, assimilation policy is done. In fact, distance of empire and colony is calculated from equation (15) and then colony new position is calculated from equation (16).

$$d = \text{Imperialist position} - \text{Colony position} \quad (15)$$

$$X = \text{round (Colony Position} + \alpha * ß * \text{int (Rnd (2))} * d) \quad (16)$$

In equation (16), ß is a number greater than one and close to 2. ß is made to colony country during move to empire country, close to it from different sides. α is a desired parameter which increases increase search around empire and decreases to make colonies move close to Vector Empire to colony. In this model, α is considered 0.5.

6) Revolution: Revolution means sudden changes in the position of a country that values change with a defined rate randomly. In fact, in this step, resources with random number are allocated to tasks that will a new country appear. Revolution rate in this model is 0.1. Figure 3 shows a sample of this change.

7) Exchange colony and empire: as explained before, if a colony gets lower cost while moving to empire then the colony and empire exchange positions.

8) Empires Competitive and Eliminate Week Empire: Total power of empire is obtained from equation (4). To takeover colonies of other empires, first according equation (5), from total cost of empire, is defined as it's normalized total cost and then, probability of takeover each empire (depend on the power of the empire), with considering total cost of empire is calculated from equation (6).

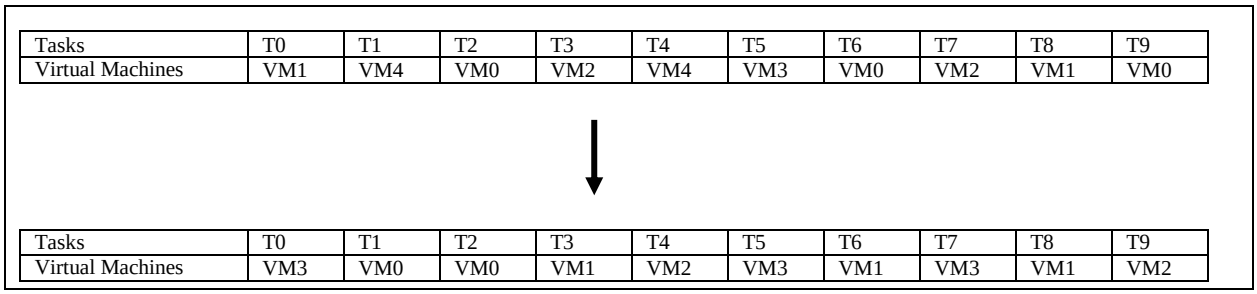

Fig. 3. Revolution

9) No need to calculate CDF1, make this mechanism to perform faster than roulette wheel in genetic algorithm. This step is ended when one of the empires takes over a colony, then after a few iteration, weakest empire will be without any colonies. In this case that empire will be removed from the empire list.

10)Termination: The algorithm continues until a converge

11)Condition or reach to total number of iteration reached.

The pseudo code of proposed ICA is given in Figure 4.

The flowchart of proposed method is given in Figure 5.

```
1 Initialize Tasks and virtual machines.
2 While (No. of countries < initial population size) {
3 Set the status of all tasks in set T to unscheduled to generate new country
4 While (Set T has unscheduled tasks) {
5 Select a task t from set T of tasks which has not been allocated to any virtual
    machine.
6 Select a virtual machine vm randomly from set VM of virtual machines.
7 Schedule task t on virtual machine vm.}
8 Store the new generated country in population P.}
9 Evaluate cost of all countries in population P by calling reliability
    evaluation algorithm.
10 Sort the countries in ascending order according to their cost value.
11 Generate initial empires from top list of countries and colonies as other
    countries.
12 While (total number of iteration < iteration threshold or No. of empire > 1)
    {
13 Calculate distance of empire and colony and move colony.
14 If (Revolution rate > Revolution threshold) then
15 Apply Revolution.
16 If (empire cost > colony cost) then
17 Exchange Empire and colony position.
18 Calculate total power of empire.
19 Calculate normalized total power of empire.
20 Make vector D
21 Get max index of vector D
22 Give a colony of weakest empire to highest power empire.
23 If weakest empire (No. of colonies) <=1 then
24 Eliminate Week Empire and change to colony.}
25 Return Best Country with min Cost
```

Fig. 4. The pseudo code of proposed ICA

[1] Cumulative distribution function

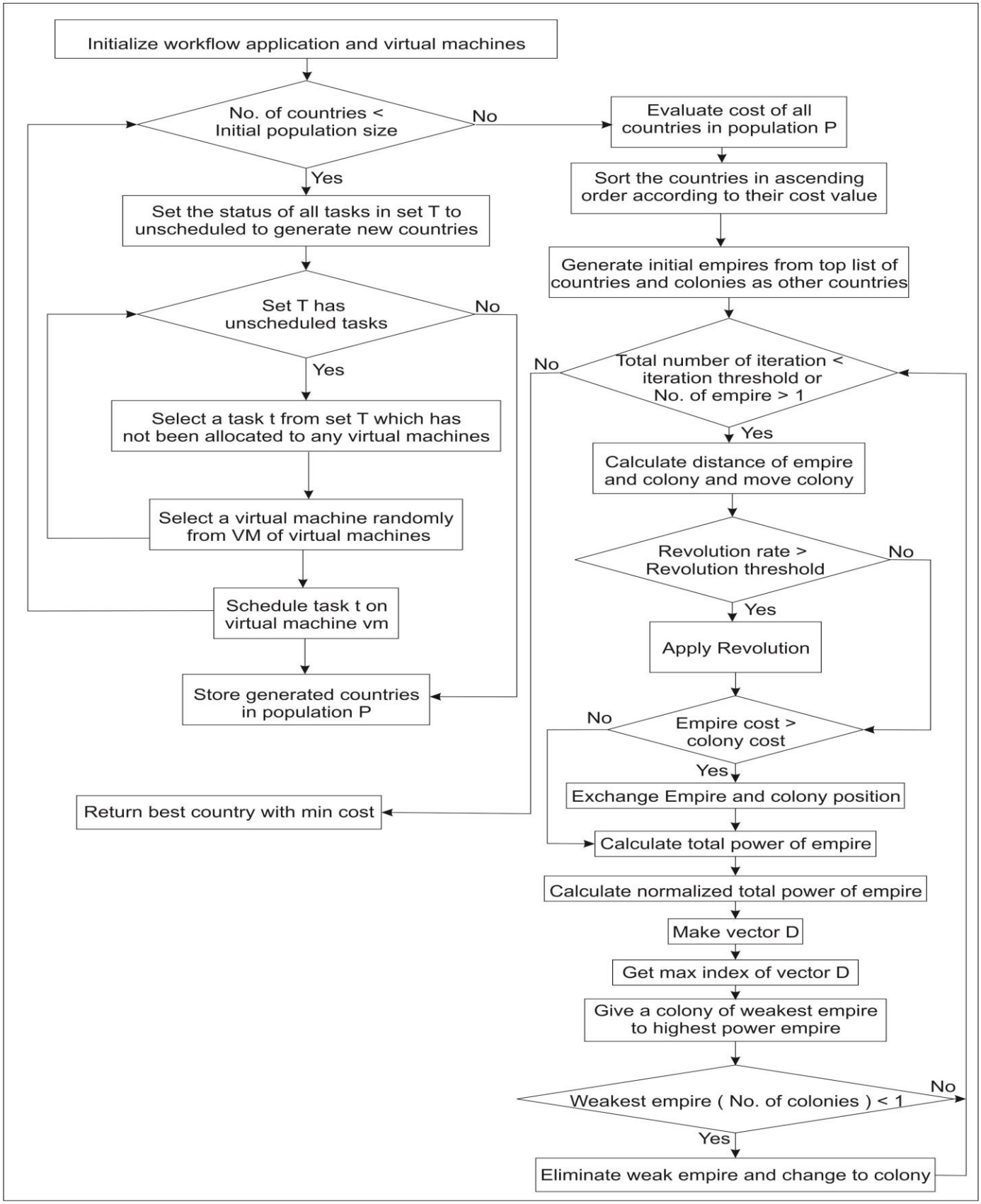

Fig. 5. The flowchart of proposed method

V. Performance Evaluation

To ensure proper functioning of the proposed method, results of the testing should be compared to other methods. Until now, different papers are provided in this field that often are handed makespan problem. In [21], the makespan and reliability are considered by using genetic algorithm. So, in this paper proposed algorithm is compared to genetic algorithm. CloudSim is used for implementation of the algorithm [24]. Table II gives information about simulation.

TABLE II. Simulation Condition

Parameters	Value
Number of Datacenters	2
Number of Hosts	2
MIPS Rate of Host	1000000
Ram of Host	6144
Number of VMs on each Host	2,3
MIPS Rate of VMs	2600,3200
Ram of VM	512
Mode of VMs on Host	Space-Shared
Total Number of VMs	5,10
Number of Tasks	50-250
Task size (No. of Instructions)	1000-2000

The experiment is performed on a computer with a dual-core, 2.4 GHz processor and 2GB of RAM. The test was done once on 5 virtual machines and another one on 10 virtual machines in an unreliable cloud environment. The scheduling algorithm estimates execution time of tasks by using MIPS rate of virtual machines. MIPS rates of some virtual machines are considered as 2600 and 3200. Instruction numbers of tasks are random numbers between 1000 and 2000. Table III and Table IV show the parameters in ICA and GA, respectively.

TABLE III. ICA Parameters

Initial Population Generation	Randomly
Initial Population Size	100
Number of Iterations	100
Revolution Rate	0.1
Assimilation Coefficient	2
Assimilation Angle Coefficient	0.5
zeta (ξ)	0.02
RF Value	0.05

TABLE IV. GA Parameters

Initial Population Generation	Randomly
Initial Population Size	100
Number of Iterations	100
Mutation operation	swapping
Mutation rate	0.01
Crossover operation	Single point
Crossover rate	0.5

Experiments are done for both algorithms with different number of tasks and resources. Table V shows average results of the experiment after running the algorithm 20 times for parameters such as makespan and reliability.

TABLE V. Comparison of Algorithms with Different Tasks and Resources

		GA		ICA	
Number of Tasks	Number of Recourses	Makespan	Reliability	Makespan	Reliability
100	5	13.42	0.96	12.1	0.98
200	10	14.01	0.9	12.71	0.93

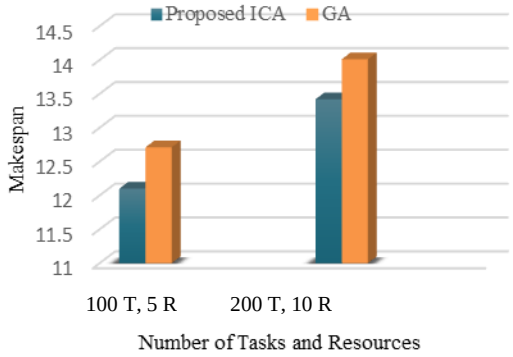

Fig. 6. comparison of makespan in both algorithm with increasing number of tasks and resources

The values of two results are compared together. The bar diagram in fFgure 6 shows makespan on resources in both algorithms for 100 tasks with 5 resources and 200 tasks with 10 resources. As shown, the proposed method reduces makespan rather than genetic algorithm by 5 percent in the first experiment and 4.4 percent in the second experiment. The ICA algorithm has useful operations such as revolution and assimilation and help to better search the problem space but in genetic algorithm mutation is used to avoid local optimum. Hence, ICA can find optimum solution than GA

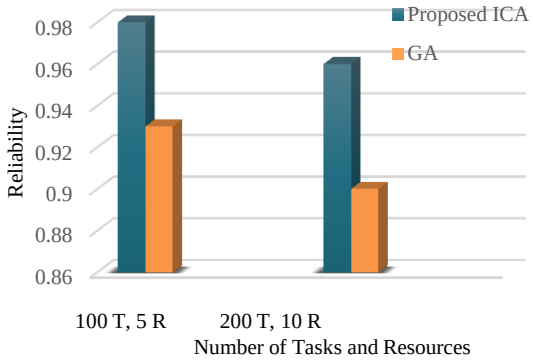

Fig. 7. compare reliability of both algorithm with increasing number of tasks and resources

Figure 7 shows reliability on resources in both algorithm for 100 tasks with 5 resources and 200 tasks with 10 resources. As shown, the proposed method increases reliability rather than genetic algorithm (in first experiment 5.4 percent and in second experiment 6.7 percent). Therefore,

according to this Figure and Figure 6, ICA proves to be able to better search the problem space rather than GA. In the other hands selected cost function here in linear condition has fitting to two objects separately.

For more studies, we use different environment scale. The results of makespan of both methods with different tasks and 5 resources are in accordance with Figure 8. In GA, during the initial stage of population, virtual machines with high reliability are selected. In this algorithm, makespan is less important. In addition, in imperialist competitive using vector D is used instead of roulette wheel that is quicker. So as shown in Figure 8, the proposed algorithm has lower makespan in comparison to GA.

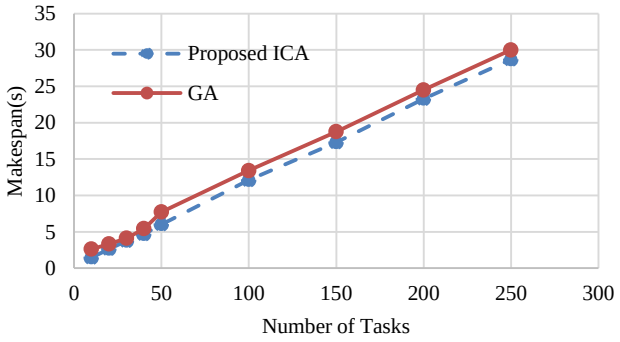

Fig. 8. comparison of makespan

Reliability evaluation results of VMs are shown in Figure 9. As shown, in low number of tasks, reliability of both algorithms is near to each other but by increasing number of tasks, reliability of proposed ICA will be higher than GA.

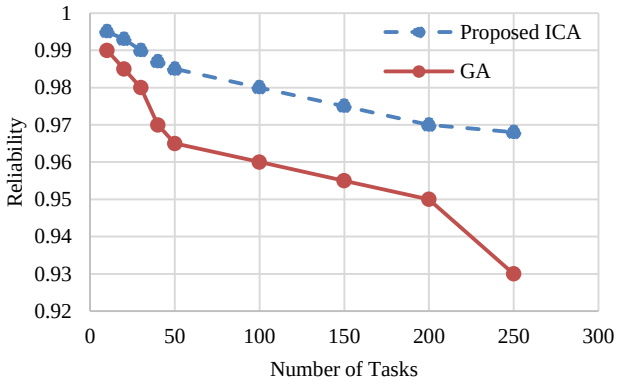

Fig. 9. comparison of reliability

VI. Conclusion

In this paper, for resource allocation, makespan and reliability were used as a fitness function. In fact, two parameters were used in ICA as a cost function. In cross layer, initial reliability of all resources is equal to 1 and after they implemented tasks, the reliability increased or decreased according to successful or failure implementation. Then, by using imperialist Competitive algorithm different solutions are checked and their cost function is calculated by makespan and reliability values. In comparison with other methods, results show that the proposed algorithm improves makespan and reliability rather than GA. In fact, the proposed method reduces makespan rather than genetic algorithm in experiment 100 tasks and 5 resources 5 percent and in in experiment 200 tasks and 10 resources 4.4 percent while is increase reliability in experiment 100 tasks and 5 resources 5.4 percent and in experiment 200 tasks and 10 resources 6.7 percent. Results show that ICA can search problem space better than GA.

References

[1] Peter Mell, Timothy Grance: The NIST Definition of Cloud Computing, National Institute of Standards and Technology, Special Publication 800-145 (2011)

[2] Zhao,L., Sakr, S., Liu, A., Bouguettaya, A.: Cloud Data Management, XIX, 202 p. 86 illus., 50 illus., Springer, 2014

[3] Chunye Gong, Jie Liu, Qiang Zhang, Haitao Chen and Zhenghu Gong.: The Characteristics of Cloud Computing, 39th International Conference on Parallel Processing Workshops, 2010.

[4] Yu, J., Buyya, R., Kotagiri, A.: Workflow Scheduling Algorithms for Grid Computing, vol. 146, pp. 173–214. Springer, Heidelberg (2008)

[5] Wang, P.C., Korfhage, W.: Process Scheduling using Genetic Algorithm. In: Parallel and Distributed Proceeding Seventh IEEE Symposium, pp. 638–641 (1995).

[6] Page, A.J., Naughton, T.J.: Dynamic Task Scheduling using Genetic Algorithm for Heterogeneous Distributed Computing. In: Proceedings 19th IEEE Conference on Parallel and Distributed Processing Symposium (2005).

[7] Moattar, E.Z., Rahmani, A.M., Derakhshi, M.R.F.: Job Scheduling in Multiprocessor Architecture using Genetic Algorithm. In: 4th IEEE Conference on Innovations in Information Technology, pp. 248–251 (2007).

[8] Mocanu, E.M., Florea, M., Ionut, M.: Cloud Computing Task Scheduling Based on Genetic Algorithm. In: System IEEE Conference, pp. 1–6 (2012).

[9] Shekhar Singh, Mala Kalra, Task scheduling optimization of independent tasks in cloud computing using enhanced genetic algorithm- International journal of application or innovation in engineering & management (ijaiem)- volume 3, issue 7, july 2014.

[10] Van den Bossche, R., Vanmechelen, K., Broeckhove, J.: —Cost Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. In: 3rd IEEE International Conference on Cloud Computing, Miami (July 2010).

[11] Tasquia Mizan, Shah Murtaza Rashid Al Masud, Rohaya Latip.: Modified Bees Life Algorithm for Job Scheduling in Hybrid Cloud, International Journal of Engineering and Technology Volume 2 No. 6, June, 2012.

[12] Yue Miao: Resource Scheduling Simulation Design of Firefly Algorithm Based on Chaos Optimization in Cloud Computing, International journal of Grid Distribution Computing, Vol.7, No.6 (2014), pp.221- 228.

[13] Hany Seidgar, Mostafa Zandieh, Hamed Fazlollahtabar, Iraj Mahdavi.: Simulated imperialist competitive algorithm in two-stage assembly flow shop with machine breakdowns and preventive maintenance, Journal of Engineering Manufacture February 18, 2015.

[14] Zhao, L., Ren, Y., Sakurai, K.: —A Resource Minimizing Scheduling Algorithm with Ensuring the Deadline and Reliability in Heterogeneous Systems‖. In: International Conference on Advance Information Networking and Applications, AINA. (IEEE 2011).

[15] Felix C. Gärtner, "Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments", ACM Computing Surveys, Vol. 31, No. 1, March 1999.

[16] Yilei Zhang, Zibin Zhengand and Michael R. Lyu, "BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-Resource Cloud Computing", 4th International Conference on Cloud Computing, IEEE, 2011.

[17] Arvind Kumar, Rama Shankar Yadav, Ran vijay and Anjali Jain "Fault Tolerance in Real Time DistributedSystem", International Journal on

Computer Science and Engineering (IJCSE),Vol. 3,ISSN: 0975-3397, No. 2 ,Feb 2011.

[18] AnjuBala, InderveerChana, "Fault Tolerance- Challenges, Techniques and Implementation in Cloud Computing", International Journal of Computer Science (IJCSI) Issues, Vol. 9, Issue 1, No 1, January 2012.

[19] Ravi Jhawar, Vincenzo Piuri and Marco Santambrogio, Member of IEEE, "Fault Tolerance Management in Cloud Computing: A System-Level Perspective ", IEEE, 2012.

[20] Anjali D.Meshram, A.S.Sambare, S.D.Zade: Fault Tolerance Model for Reliable Cloud Computing, International Journal on Recent and Innovation Trends in Computing and Communication,600-603 (2013).

[21] Lovejit Singh and Sarbjeet Singh: A Genetic Algorithm for Scheduling Workflow Applications in Unreliable Cloud Environment, Springer-Verlag Berlin Heidelberg, 139-150 (2014).

[22] Rajkumar Buyya, Rodrigo N. Calheiros, Jungmin Son, Amir Vahid Dastjerdi, and Young Yoon : Software-Defined Cloud Computing: Architectural Elements and Open Challenges, Cloud Computing and Distributed Systems (CLOUDS) Laboratory Department of Computing and Information Systems, arXiv:1408.6891v2 [cs.DC] 19 Feb 2015.

[23] E. Atashpaz-Gargari and C. Lucas: Imperialist Competitive Algorithm An algorithm for optimization inspired by imperialistic competition, IEEE Congress on Evolutionary Computation, 2007.

[24] Calheiros, R.N., Ranjan, R., De Rose, C.A.F., Buyya, R.K.: CloudSim: A Novel Framework for Modelling and Simulation of Cloud Computing Infrastructures and Services. GRIDS Laboratory. The University of Melbourne, Australia (2009).