American Journal of Computer Science and Technology
2018; 1(1): 24-30
http://www.sciencepublishinggroup.com/j/ajcst
doi: 10.11648/j.ajcst.20180101.14

An Efficient Algorithm for Workflow Scheduling in the Clouds Based on Differential Evolution Method

Toan Phan Thanh[1], Loc Nguyen The[2], Said Elnaffar[3]

[1]Faculty of Technology Education, Hanoi National University of Education, Ha Noi, Viet Nam

[2]Faculty of Information Technology, Hanoi National University of Education, Ha Noi, Viet Nam

[3]School of Engineering, Computer Science Department, American University of RAK, Ras al Khaimah, UAE

Email address:
pttoan@hnue.edu.vn (T. P. Thanh), locnt@hnue.edu.vn (L. N. The), said.elnaffar@gmail.com (S. Elnaffar)

To cite this article:
Toan Phan Thanh, Loc Nguyen The, Said Elnaffar. An Efficient Algorithm for Workflow Scheduling in the Clouds Based on Differential Evolution Method. *American Journal of Computer Science and Technology.* Vol. 1, No. 1, 2018, pp. 24-30.
doi: 10.11648/j.ajcst.20180101.14

Received: October 27, 2017; **Accepted**: December 4, 2017; **Published**: January 2, 2018

Abstract: The Cloud is a computing platform that provides on-demand access to a shared pool of configurable resources such as networks, servers, storage that can be rapidly provisioned and released with minimal management effort from clients. At its core, Cloud computing focuses on manimizing the effectiveness of the shared resources. Therefore, workflow scheduling is one of the challenges that the Cloud must tackle especially if a large number of tasks are executed on geographically distributed servers. The Cloud is comprised of computational and storage servers that aim to provision efficient access to remote and geographically distributed resources. To that end, many challenges, specifically workflow scheduling, are yet to be solved such. Despite it has been the focus of many researchers, a handful efficient solutions have been proposed for Cloud computing. In this work, we propose a novel algorithm for workflow scheduling that is derived from the Opposition-based Differential Evolution method, MODE. This algorithm not only ensures fast convergence but also averts getting trapped in local extrema. Our simulation experiments Cloud Sim show that MODE is superior to its predecessors. Moreover, the deviation of its solution from the optimal one is negligible.

Keywords: Workflow Scheduling, Opposition-Based Differential Evolution, Cloud Computing, Differential Evolution

1. Introduction

The Cloud is a computing platform that provides convenient, on-demand access to a shared pool of configurable computing resources such as networks, servers, storage that can be rapidly provisioned and released with minimal management effort on clients. At its core, Cloud computing focuses on maximizing the effectiveness of the shared resources. Therefore, workflow scheduling is one of the challenges that the Cloud must tackle especially if a large number of tasks are executed on geographically distributed servers. This requires a reasonable scheduling algorithm in order to minimize a task completion time (*makespan*).

Our work described in this paper is a new attempt to introduce a more efficient scheduling algorithm. Specifically, we introduce a Modified version of the Opposition-based Differential Evolution (ODE) method [19], which we call MODE. Via experiments, we demonstrate its effectiveness and performance in comparison with other algorithms.

The rest of the paper is organized as follows. Section II reviews some of the related works germane to workflow scheduling algorithms. Section III describes the Opposition-based Differential Evolution (ODE) method. Section IV describes the computation and communication model on which Cloud tasks operate. Based on this model, Section V presents our proposed scheduling algorithm MODE (Modified ODE). Section VI describes the experiments we conducted using the simulation tool CloudSim [1] in order to evaluate the proposed algorithm. Section VII concludes our paper and sketches future work.

2. Related Work

A workflow is a sequence of connected tasks. Workflow

scheduling in Clouds is a challenge because each task needs to be mapped to an appropriate server while allowing that task to satisfy some performance constraints. In general, the scheduling problem, i.e. the mapping of tasks to the computation resources such as servers, is an NP-complete problem [4]. Hence, past works mainly banked on heuristic-based or metaheuristic-based solutions for scheduling workflows.

Some researchers [13] proposed an algorithm for task scheduling based on two-level load balancing in the cloud environment. This algorithm not only meets user's requirements but also provides high resource utilization. The authors also introduced the implementation of an efficient Quality of Service (QoS) based Meta-heuristic method.

Others [15] presented an optimized algorithm for task scheduling based on Hybrid Genetic Algorithms. They considered the QoS requirements like completion time, bandwidth, cost, distance, reliability of different type tasks. They used annealing in their implementation after the selection, crossover and mutation, to improve local search ability of genetic algorithm.

In [16], the authors presented a model for task scheduling in cloud computing to minimize the overall time of execution and transmission. They proposed a particle swarm optimization algorithm to solve task scheduling that is based on small position value rule (SPVPSO). The researchers compared the SPVPSO algorithm with the CR-PSO and L-PSO and found that the SPVPSO algorithm had attained the optimal solution and converged faster in large tasks than the other two algorithms.

Another work [17] proposed a hierarchical scheduling algorithm which satisfied the Service Level Agreement with quick response from the service provider. The authors used the response time as a Quality of Service (QoS) metric to prioritize the execution of jobs by estimating their completion times.

Another paper [18] presented an optimized algorithm for task scheduling in cloud computing based on Activity-Based Costing (ABC). This algorithm assigns a priority level for each task and uses cost drivers. The priority is estimated based on four major factors: time, space, resources and profit.

Pandey [9] presented a scheduling algorithm (PSO_H) to minimize the cost of the execution. However, instead of finding the schedule which has a minimum cost, PSO_H looked for the schedule that minimizes the execution cost at the nearest server. The author compared the PSO_H algorithm with the Random and Round Robin algorithm and showed that the optimal solution of PSO_H algorithm is better than two matching algorithms.

3. Opposition-Based Differential Evolution (ODE)

Opposition-based Differential Evolution (ODE) [19] is an evolutionary optimization technique that consists of two main steps: population initialization and producing new generations by genetic operations such as selection, crossover and mutation. ODE enhances these two steps by considering the opposite point of individuals in the population.

3.1. Opposition-Based Learning

The concept of Opposition-Based Learning (OBL) was original introduced by Tizhoosh [20]. The main idea behind OBL is the simultaneous consideration of an estimate and its corresponding opposite estimate in order to achieve a better approximation for the current candidate solution.

Definition 1 (Opposite Number): Let $x \in [a, b]$ be a real number, then the opposite number $\bar{x}$ is defined as

$$\bar{x} = a + b - x \tag{1}$$

Definition 2 (Opposite Point in n-Dimensional Space): Similarly, the above definition can be extended for higher dimensions as follows:

Let $P(x_1, x_2,\ldots,x_D)$ be an D-dimemsional vector, where $x_i \in [a_i, b_i]$; i=1,2,…,D. The opposite of P is defined by: $\bar{P} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_D)$ where

$$\bar{x}_i = a_i + b_i - x_i \tag{2}$$

The opposition-based optimization can be defined as follows:

Let $P=(x_1, x_2,..,x_D)$ be a point in D-dimensional space, and f(.) is the fitness function which is used to measure the candidate's fitness. According to the definition of opposite point, $\bar{P} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_D)$ is the opposite of $P=(x_1, x_2,..,x_D)$. Now if $f(\bar{P}) \leq f(P)$, then point P can be replaced by $\bar{P}$, otherwise we continue with P.

3.2. Opposition-Based Population Initialization

Using the definition of opposite points, the opposition-based initialization for the ODE can be described as follows:

Procedure: Opposition-Based Population Initialization

1. Random initialization of population $P(x_1, x_2,\ldots,x_{PopSize})$
2. Calculate opposite population by equal(2)
3. OP ← Opposite (P)
4. Selecting the PopSize fittest individuals from {P ∪ OP} as initial population.

3.3. Opposition-Based Differential Evolution Algorithm

By combining the Opposition-Based Learning method and the Differential Evolution algorithm, the ODE algorithm can be described as follows:

Algorithm:Opposition-Based Differential Evolution

1. Call Procedure: Opposition-Based Population Initialization
2. for i=1 to PopSize do
3. $f_i \leftarrow$ fitness(x_i)
4. while (criteria is not satisfied) do
5. for i=1 to PopSize do
6. r1, r2 ← Random(1, PopSize)
7. F ← Random(0,1)
8. $v_i(t) \leftarrow$ pbest + F×(x_{r1}- x_{r2})
9. for j=0 to D do

10. $u_{i,j} = \begin{cases} v_{i,j} \text{ nếu } rand_{i,j} \leq CR \text{ hoặc } i = I_{rand} \\ p_{i,j} \text{ nếu } rand_{i,j} \geq CR \text{ hoặc } i \neq I_{rand} \end{cases}$
11. end for
12. if($f(u_i(t)) \leq f(x_i(t))$)
13. $x_i(t+1) = u_i(t)$
14. end if
15. end for
16. rand ← Random(0,1)
17. if(rand < J_r)
18. Calculating Opposite Population OP(PopSize)
19. Fitness Evaluation
20. Selecting PopSize Fittest Individuals from {Current Population, OP}
21. end if
22. end while

4. Problem Formulation

We denote the workflow as a Directed Acyclic Graph (DAG) represented by G=(V, E), where:

1. V is set of vertex, each vertex represents a task
2. T={T_1, T_2,…,T_M } is the set of tasks, M is the number of tasks
3. E represents the data dependencies between these tasks. The edge (T_i, T_j) ∈ E means the task T_i is the father of the task T_j, the data produced by T_i will be consumed by the task T_j.
4. The Cloud's computation resources are a set of servers S = {S_1, S_2,….,S_N}. N is the number of servers.
5. Each task T_i can be executed by any server S_j∈S, and S_i has to handle the whole the workload of T_i
6. The computation of task T_i denoted by W_i (flop-floating point operations)
7. $P(S_i)$: the computation power of the server S_i (MI/s : million instructions/second)
8. The bandwidth $B(S_i, S_j)$ between server S_i and server S_j is represented by the function B(): S×S → R^{+}. We assume that $B(S_i, S_i) = \infty$ and $B(S_i, S_j) = B(S_j, S_i)$
9. D_{ij}: data produced by task T_i and consumed by task T_j.

Each scheduling plan can be represented by the function *f*(): T→S where $f(T_i)$ is the server that handles the task T_i

Based on the above assumptions we have:

10. The execution time of the task T_i is

$$\frac{W_i}{P(f(T_i))} \quad (3)$$

11. The communication time between the task T_i and T_j is

$$\frac{D_{ij}}{B(f(T_i), f(T_j))} \quad (4)$$

Formally, we need to minimize the execution time, called *makespan*, of the workflow:

makespan → min

where *makespan* is the time difference between the start and finish of a sequence of workflow's tasks.

5. Proposed Algorithm

5.1. Particle Representation

In the proposed scheduling algorithm, the solution is represented as a vector of length equal to the number of tasks. The value corresponding to each position *i* in the vector represents the server to which task *i* was executed.

Example 1

Consider a workflow with a set of tasks T={T_1, T_2, T_3, T_4, T_5}, a set of servers S = {S_1, S_2, S_3}. So the particle x_i^k = [1; 2; 1; 3; 2] gives us the following scheduling plan:

T_1	T_2	T_3	T_4	T_5
S_1	S_2	S_1	S_3	S_2

In that scheduling plan, tasks T1 and T3 will be executed by the server S1, tasks T2 and T5 are assigned to the server S2 and task T4 is handled by server S3.

5.2. Method to Calculate the Opposite of Individuals

The design of ODE entails calculating the opposite of individuals in the population, which can be carried out as follows:

Let a = Max{$P(S_i)$}; ∀i=1,2,..,N

b = Min{$P(S_i)$}; ∀i=1,2,..,N

Assuming that the particle $x_i = (S_{i\pi(1)}, S_{i\pi(2)},\ldots,S_{i\pi(M)})$; $S_{i\pi(j)} \in S$, ∀j=1,2,..,M; the opposite of x_i, denoted by $\bar{x_i}$, will be calculated as follows:

$$\bar{x_i} = (\bar{S}_{i\pi(1)}, \bar{S}_{i\pi(2)}, \ldots, \bar{S}_{i\pi(M)}) \quad (5)$$

where:

$$\bar{S}_{i\pi(j)} = a + b - S_{i\pi(j)};\ \forall j = 1,2,..,M$$

We we subsequently assign the value corresponding to each position *j* of vector $\bar{x_i}$ by identifying the server which has a computation power closer to $\bar{S}_{i\pi(j)}$ than any other server

$$\overline{x_{ij}} \leftarrow k \text{ nếu } \left|P(S_k) - \bar{S}_{i\pi(j)}\right| \leq \left|P(S_r) - S_{i\pi(j)}\right| \forall S_r \quad (6)$$

Algorithm: OP_Algorithm

Input: population p = (x_1, x_2,…,$x_{PopSize}$)

Output: opposite of population OP

1. a ← Max{$P(S_i)$}; ∀i=1,2,…,N
2. b ← Min{$P(S_i)$}; ∀i=1,2,…,N
3. for i=1 to PopSize do
4. $\bar{x_i} \leftarrow opposite\ (x_i); by\ equation\ (5)$
5. assign the identity of the server to each position j of vector $\bar{x_i}$ by equation (6)
6. end for

return OP

5.3. Rank-Based Roulette Wheel Selection

Rank-based roulette wheel selection is the selection strategy where the probability of a particle being selected is based on its fitness rank relative to the entire population. Rank-based selection schemes first sort individuals in the

population according to their fitness and then computes the selection probabilities according to their ranks rather than the fitness values. Rank-based selection uses a function to map the indices of individuals in the sorted list to their selection probabilities. The rank for an individual may be scaled linearly using the following formula:

$$rank(pos) = 2 - SP + \left(2 \times (SP - 1) \times \frac{pos-1}{PopSize-1}\right); \text{ where } 1.0 \leq SP \leq 2.0 \quad (7)$$

The algorithm to selected individuals from population by Rank-based Roulette Wheel Selection can be described as follows:

Algorithm: RBRWS algorithm
Input: population p = (x_1, x_2,…,$x_{PopSize}$)
Output: particle p_s
Begin
1. SP ← [1.0, 2.0]
2. for i=1 to PopSize do
3. f_i ← fitness(x_i)
4. sort p by ascending of f_i
5. for i=1 to PopSize do
6. pos[i] ← PopSize – 1
7. for i=1 to PopSize do
8. calculate $rank_i$ by equation (7)
9. rand ← Random(0,SP)
10. s← PopSize
11. while(rank[s] < rand && s>0)s= s-1
12. return x_s

End.

5.4. The MODE Algorithm

The MODE algorithm can be described as follows:

Algorithm MODE ( )
Input: T, S, size of workload W[1×*M*], P[1×*N*], B[*N*×*N*], D[*M*×*M*], the number of particle *NoP*
Output: the best position *gbest*
1. Call procedure: Opposition-Based Population Initialization
2. while(criteria is not satisfied)do
3. for i=1 to PopSize do
4. selecting p_1 from population by RBRWS algorithm
5. selecting p_2 from population by RBRWS algorithm
6. F ← Random(1,0)
7. v_i ← pbest + F×(p_1 – p_2)
8. assign the identity of the server to each position j of vector v_i by equation (4)
9. $rand_{i,j}$ ← Random(0,1)
10. I_{rand} ← random(1,M)
11. $u_{i,j} = \begin{cases} v_{i,j} \text{ nếu } rand_{i,j} \leq CR \text{ hoặc } i = I_{rand} \\ x_{i,j} \text{ nếu } rand_{i,j} \geq CR \text{ hoặc } i \neq I_{rand} \end{cases}$
12. if (makespan(u_i) < makespan(x_i))
13. x_i ← u_i
14. if(rand < J_r)
15. Calculating Opposite Population OP by OP_Algorithm
16. Fitness Evaluation
17. Selecting PopSize Fittest Individuals from {Current Population, OP}
18. End while

Return *gbest;*

In the Initialization step, we randomly generate a population of PopSize individuals, and calculate the opposite of individuals in the population using equation (5) and (6), then select the Popsize fittest individuals from current population and their counterpart opposites.

In each iteration, MODE selects two individuals from population according to the Rank-based Roulette Wheel Selection method. According to the mutation operator, a mutant vector is generated by adding the weighted difference between two selected vector of the target population individuals to a third one as follows :

$$v_i \leftarrow pbest + F \times (p_1 - p_2)$$

where p1, p2 are two individuals selected by rank-based roulette wheel selection method, and pbest is the best individual in the population.

After the mutation phase, the crossover operator is applied to obtain the trail vector u_i using the following equation :

$$u_{i,j} = \begin{cases} v_{i,j} \; if \; rand_{i,j} \leq CR \; or \; i = I_{rand} \\ x_{i,j} \; if \; rand_{i,j} \geq CR \; or \; i \neq I_{rand} \end{cases}$$

Where $rand_{i,j}$ is the j^{th} independent random number uniformly distributed in the range of [0, 1]. Also I_{rand} is a random number in the range of [1, M]. CR is a user defined crossover factor in the range [0, 1].

Now, we need to decide whether the trail vector u_i should be a member of the population of the next generation, it is compared with the target individual x_i. Finally, the selection is based on the survival of the fitness as follows :

$$x_i = \begin{cases} u_i, if \; makespan(u_i) < makespan(x_i) \\ x_i, otherwise \end{cases}$$

After the crossover phase, we calculate the opposite of individuals in the population by equation (5) and (6). The fitness of individuals is evaluated, then the PopSize fittest individuals are selected for the next generation.

6. Evaluation

We conducted some experiments in order to compare the performance of the MODE against the PSO_H [9] and Random [12]. Our experimental setup consists of a computer with Intel Core i5 2.2 GHz, RAM 4GB, and Windows 7 Ultimate. The experiments were carried out using the simulation tool CloudSim [14] and the packet library Jswarm [1] and Java.

6.1. Data

We use both random and real world instances in our experiments using the following data sets:

The computation power of the servers and the bandwidth of connections between servers are collected from a Cloud

provider such as Amazon [2] and its Web site (exp. http://aws.amazon.com/ec2/pricing)

The sets of working data are collected from Montage project [3]. The instances are divided into 6 groups based on the number of servers N and the number of tasks M:

Group 1: M=10, N=3; Group 2: M=10, N=5; Group 3: M=20, N=5; Group 4: M=20, N=8;

Group 5: M=25, N=8; Group 6: M=50, N=8;

We denote the ratio of the number of edges and the number of vertexes of graph G by:

$$\alpha = \frac{|E|}{M \times (M-1)/2}$$

6.2. Configuration Parameters

The Cloud's configuration parameters are chosen as follows:

1. Server's computation power: from 1 to 250 (million instructions/s)
2. Connection bandwidth B: from 10 to 100 (Mbps)
3. Communication data D: from 1 to 10000 (MB)

The MODE's configuration parameters are:

1. Number of particles *NoP*=100; α: from 0.2 to 0.7;
2. Differential amplification factor F=0.5;
3. Crossover probability constant CR=0.9;
4. Jumping rate constant $J_r = 0.3$.

6.3. Results

Each problem instance was executed 30 times continuously. The results summarized in Table 1 show that the mean value (column *Mean*) and standard deviation value (column *STD*) of MODE are better than those of PSO_H [9] and Random [12] in most of the cases. When the number of servers (*N*) and the number of tasks (*M*) are relatively large (i.e. larger scale cloud), for example *M*=20 and *N*=8; *M*=25, *N*=8; *M*=50, *N*=8, MODE is better than PSO_H and Random with respect to all metrics: mean, standard deviation and best value (column *Best*).

Figures 1-6 depict the performance of the three algorithms: proposed algorithm MODE, PSO_H [9], and Random [12] where the vertical axis represents the makespan (seconds) of the schedule. For each instance, we compare the best position vector (column BEST), the mean value (column MEAN) and standard deviation value (column STD). At the first instance, MODE even found the optimal solution.

Table 1. *Comparison the makespan of MODE with other algorithms.*

M	N	α	MODE			PSO_H			RANDOM		
			Best	Mean	STD	Best	Mean	STD	Best	Mean	STD
10	3	0.26	16.9	17.3	0.4	16.4	20.4	2.4	21.4	28.6	3.2
10	5	0.26	75.5	78.2	0.9	86.0	107.5	13.2	123.3	184.1	42.4
20	5	0.15	27.5	29.3	1.4	29.6	41.0	5.0	45.8	59.0	6.8
20	3	0.31	32.5	33.9	0.8	33.2	41.9	4.6	47.4	65.6	7.8
20	8	0.31	29.9	31.7	1.2	37.1	44.7	6.1	51.6	67.6	6.8
50	8	0.3	9.1	9.7	0.5	12.1	14.0	0.9	13.9	87.1	25.2

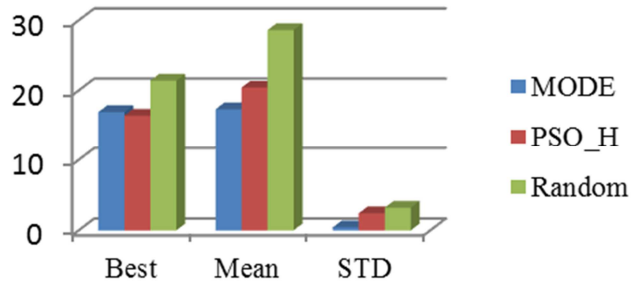

Figure 1. *M=10, N=3.*

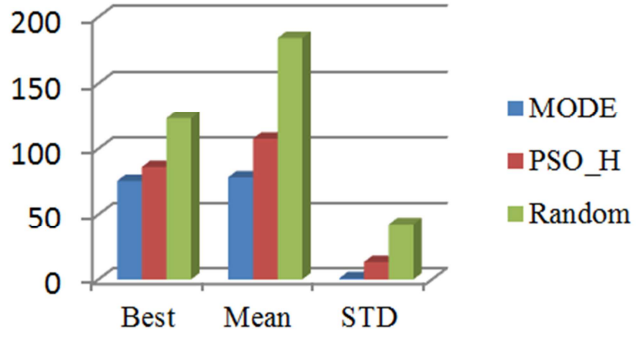

Figure 2. *M=10, N=5.*

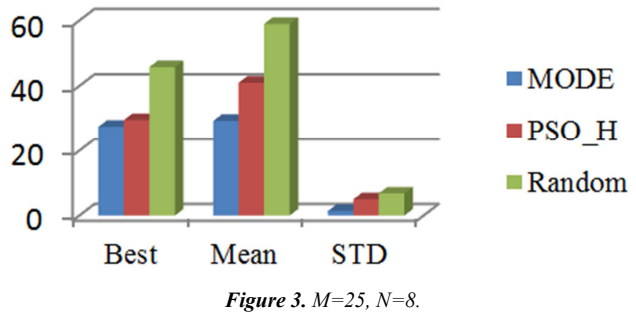

Figure 3. *M=25, N=8.*

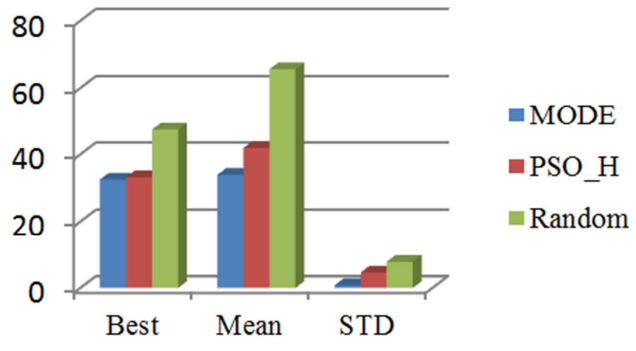

Figure 4. *M=20, N=3.*

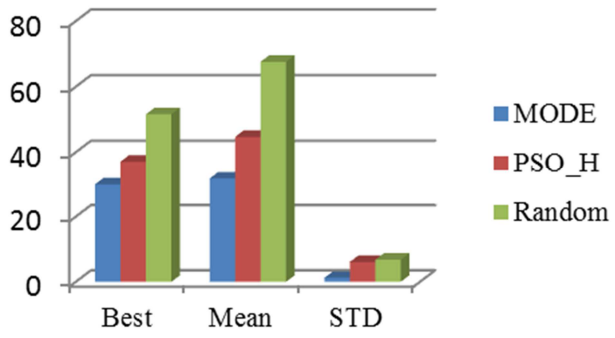

Figure 5. *M=25, N=8.*

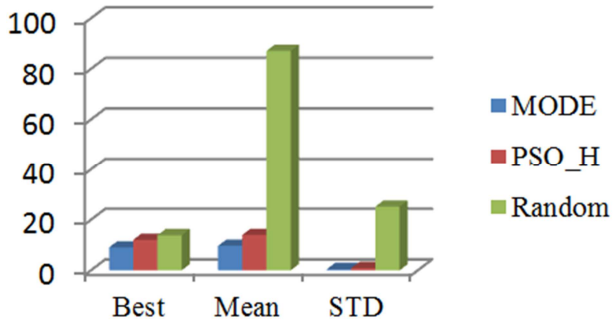

Figure 6. *M=50, N=8.*

7. Conclusion

The ultimate goal of any scheduling algorithm is to minimize the execution time. Far from that goal, our proposed algorithm also avoid being trapped on local extrema. The contributions of our paper are:

1. Building a novel approach, represented by the formula to calculate the opposite of individuals in the population.
2. Proposing a new scheduling algorithm named MODE by incorporating the ODE (Opposition-Based Differential Evolution) strategy and Rank-based Roulette Wheel Selection method.

The experimental results show that MODE is superior to its predecessor especially when MODE works in a larger scale Cloud, i.e. the number of servers and tasks are fairly large. In the future, we are planning to improve the MODE algorithm in order to solve bigger instances in a reasonable makespan.

References

[1] R. N. Calheiros, R. Ranjan, A. Beloglazov, Cesar A. F. De Rose, and R. Buyya, CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software: Practice and Experience, volume 41, Number 1, Pages: 23-50, Wiley Press, USA, 2011

[2] J. V. Vliet, F. Paganelli, Programming Amazon EC2, O'Reilly Media, ISBN 1449393683, 2011

[3] http://montage.ipac.caltech.edu

[4] J. D. Ullman, NP-complete scheduling problems, Journal of Computer and System Sciences, pages 384-393, volume 10, issue 3, 1975

[5] S. Parsa, R. E. Maleki, RASA: A New Task Scheduling Algorithm in Grid Environment, International Journal of Digital Content Technology and its Applications, volume 3, No. 4, 2009

[6] A. Agarwal, S. Jain, Efficient Optimal Algorithm of Task Scheduling in Cloud Computing Environment, International Journal of Computer Trends and Technology Volume 9, 2014

[7] J. Huang, The Workflow Task Scheduling Algorithm Based on the GA Model in the Cloud Computing Environment, Journal of software, volume 9, 2014

[8] H. Liu, A. Abraham, C. Grosan, A Novel Variable Neighborhood Particle Swarm Optimization for Multi-objective Flexible Job-Shop Scheduling Problems, Proc. of 2nd International Conference on Digital Information Management (ICDIM '07), Volume 1, pages 138-145, 2007.

[9] S. Pandey, L. Wu1, S. M. Guru, R. Buyya1, A Particle Swarm Optimization (PSO)-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments, Proc. of 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), pages 400-407, 2010

[10] J. Kennedy, R. C. Eberhart, Particle swarm optimization, Proc. of IEEE International Conference on Neural Networks. pages 1942–1948, 1995

[11] A. E. M. Zavala, EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation IIA Comparison, A Comparison Study of PSO Neighborhoods, pages 251-295, Springer-Verlag Berlin Heideberg, 2013

[12] M. Mitzenmacher, E. Upfal, Probability and Computing: Randomized Algorithms and Probabilistic Analysis, Cambridge University Press (2005)

[13] Dr. Sudha Sadhasivam, R. Jayarani, Dr. N. Nagaveni, R. Vasanth Ram, Design and Implementation of an efficient Twolevel Scheduler for Cloud Computing Environment, In Proceedings of International Conference on Advances in Recent Technologies in Communication and Computing, 2009

[14] R. Buyya, R. Calheiros, Modeling and Simulation of Scalable Cloud Environment and the Cloud Sim Toolkit: Challenges and Opportunities, IEEE publication 2009, pp1-11

[15] G. Guo-Ning and H. Ting-Lei, Genetic Simulated Annealing Algorithm for Task Scheduling based on Cloud Computing Environment, In Proceedings of International Conference on Intelligent Computing and Integrated Systems, 2010, pp. 60-63

[16] L. Guo, Task Scheduling Optimization in Cloud Computing Based on Heuristic Algorithm, Journal of networks, v.7, No.3, 2012, pp. 547-552

[17] R. Rajkumar, T. Mala, Achieving Service Level Agreement in Cloud Environment using Job Prioritization in Hierarchical Scheduling, Proceeding of International Conference on Information System Design and Intelligent Application, 2012, vol 132, pp 547-554

[18] Q. Cao, W. Gong and Z. Wei, An Optimized Algorithm for Task Scheduling Based On Activity Based Costing in Cloud Computing, In Proceedings of Third International Conference on Bioinformatics and Biomedical Engineering, 2009, pp.1-3

[19] R. Storn and K. Price, Differential Evolution-A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, Journal of Global Optimization, 1997, pp. 341-359

[20] H. R. Tizhoosh, Opposition-based learning: A new scheme for machine intelligence, International Conference on Computational Intelligence for Modelling, 2005, pp. 695-701