Dicle University
Journal of Engineering
https://dergipark.org.tr/tr/pub/dumf
duje.dicle.edu.tr

Research Article

Workflow Scheduling for Cloud Computing Using Evolutionary Algorithm

Mehmet KAYA[1], Betül BOZ[2*]

[1] Marmara University, Computer Engineering Department, mehmet.kaya@marmara.edu.tr, Orcid No: 0009-0003-7393-4226
[2] Marmara University, Computer Engineering Department, betul.demiroz@marmara.edu.tr, Orcid No 0000-0001-7819-347X

ARTICLE INFO

Article history:

Received 1 August 2023
Received in revised form 25 November 2023
Accepted 25 November 2023
Available online 31 December 2023

Keywords:

Workflow Scheduling, Cloud Computing, Evolutionary Algorithm

Doi: 10.24012/dumf.1335981

* Corresponding author

ABSTRACT

Cloud computing provides powerful, highly scalable, flexible resources for real world applications. It also reduces the cost and operation expenses. Workflow scheduling is important for getting higher performance, reducing cost and using resources more efficiently in cloud computing. Workflow scheduling in cloud systems assigns tasks to resources available in the system and aims to utilize cloud resources by decreasing makespan of the workflow. In this study, an evolutionary algorithm is proposed to solve workflow scheduling problem. The main objective of this work is to minimize the makespan of the schedule. To achieve this goal, problem specific crossover operator and mutation operators are proposed in the evolutionary algorithm. The crossover operator will combine the problem-specific information stored in both parents to create a new individual. The mutation operators will explore neighbor solutions using some intelligent search mechanisms. This unique design of the operators increases the diversity of the search space and the quality of the solutions. As a result, the workflow schedules obtained from the evolutionary algorithm decreases the makespan of the workflow in the cloud system. The performance of the proposed study is measured using well-known scientific workflows and is compared with the algorithms from the literature. The proposed study outperforms all related algorithms in 67% of the test cases and obtains the same results in the remaining test cases.

Introduction

Cloud computing aims to provide computing resources to real world applications dynamically. The performance of cloud services is mostly dependent on the scheduling of tasks to the available computing resources in the cloud. If the scheduling is not properly done, then the computing resources in the system may be underutilized or overutilized. Improper scheduling may result as an increase in the execution time of the tasks, waste of system resources and increase in cost for usage of resources [1]. Since there are limited number of resources in the cloud and the total number of requests for using these resources are increasing day by day, efficient scheduling of tasks in cloud computing is very crucial.

Scheduling problem is known to be NP-complete [2], so meta-heuristic algorithms are proposed to find optimal or sub-optimal solutions. There are many meta-heuristic algorithms that successfully provide schedules [3] but they are not specifically designed for cloud systems [4]. Min-Min [5], Max-Min [5], First Come First Serve (FCFS) [6], Heterogenous Earliest Finish Time (HEFT) [7], Minimum Completion Time (MCT) [8] are heuristic algorithms that are very widely used in cloud computing studies. Min-Min algorithm mainly focuses on the task that can be executed in the minimum completion time, whereas in Max-Min the task with the longest execution time is first selected and assigned to the VM that will execute the task fastest. FCFS assigns the tasks to VMs according to their arrival time. HEFT is a list-based scheduling algorithm which includes a task priority list. Each task has an estimated completion time and decisions about scheduling are made according to this value. MCT uses expected minimum execution time of tasks. There are also algorithms that consider multi-objectives such as makespan and cost using task duplication [9], genetic algorithm [10], evolutionary algorithm [11] and neural network based dynamic workflow scheduling [12].

The main motivation behind this study is to design an evolutionary algorithm considering the properties of cloud systems. Cloud systems offer multiple computing resources to the users so in our algorithm the individuals are designed as a two-dimensional array. The first dimension refers to computing resources and the second dimension holds the tasks assigned to these computing resources. The crossover operator is designed to carry the information included in the parents to the offspring. 3 different mutation operators and a hybrid mutation operator using these 3 different mutation operators in a probabilistic manner, are proposed in this study. The

mutation operators are designed to explore neighbor solutions using some intelligent search mechanisms. Since the main performance issue of cloud computing is the total execution time of the workload, our main objective is to decrease makespan of the workflow. The performance of the proposed algorithm is promising and can be extended to support other QoS (Quality of Service) requirements such as makespan, cost, reliability using multi-objectives.

The rest of the paper is organized as follows. In the next section, workflows and their representation are explained, and then the details of the proposed algorithm is given. In Section 3, the simulation environment used for workflows in cloud and scientific workflows used in the experiments are provided. Then the performance of the algorithm is given with a comparison of well-known scheduling algorithms. Finally the conclusions and future directions for the proposed study are provided.

Material and Method

Workflow Representation

Workflow is representing the input tasks and their dependencies using a Directed Acyclic Graph (DAG). DAG is denoted by W = {V, E} where V is the set of vertices, and E is the set of edges between these vertices.

Tasks are denoted as vertices V={t_0,..,t_n} where t_0 to t_n represent n tasks in the workflow. Data dependency between these tasks are denoted as edges between related vertices in the graph as E={$t_i \rightarrow t_j \mid t_i, t_j \in V$} denoting that there is a data dependency from t_i to t_j. The size of data that needs to be transferred in between two tasks may vary, therefore weights are added to the edges to denote the transfer amount. An example workflow with 8 tasks is shown in Figure 1. The dependencies between the tasks are represented with the edges having different weight values. In this example, the highest amount of data transfer is between t_6 and t_7.

For a task $t_i \in T$, $P(t_i)$ denotes the set of precursors of t_i, $S(t_i)$ denotes the set of successors of t_i. If a task does not have any successors, it is an exit task. There can be more than one exit task. All exit tasks can be routed to one exit task which is denoted as t_e. Finishing time of the task $t_i \in T$, can be denoted as $FT(t_i)$ and starting time of the task $t_i \in T$, can be denoted as $ST(t_i)$. Makespan of a workflow is calculated as follows:

$$Makespan(W) = FT(te) \quad (1)$$

Evolutionary Algorithm

Evolutionary algorithm is a metaheuristic method that is commonly used in optimization problems. Since evolutionary algorithms are inspired by biological evolution, they use some mechanisms like reproduction, mutation, recombination, and selection. Population of the algorithm includes candidate solutions to the optimization problem, where each solution is represented as individuals in the population. Individual representation may change depending on the problem considered. Fitness function determines the quality of a solution. An Evolutionary Algorithm generally has the following steps: initialization, selection, crossover, mutation and termination.

Initial population is created in the initialization step. Some individuals in the initial population can be created using a heuristic approach to speed up the optimization process and others are created randomly to generate diversity. Once the initial population is created, the fitness of each individual is calculated. Both crossover and mutation operations are applied to the individuals of the population and new individuals which are referred as offsprings, are created. If the offspring has a better fitness value, then it will be replaced with its parents. This process is repeated until the termination criteria is met.

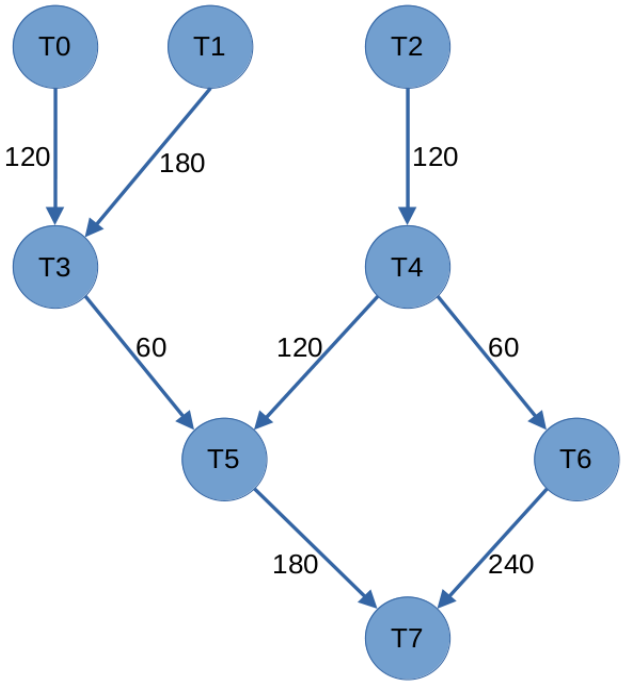

Figure 1. An example workflow represented as a DAG

Proposed Algorithm

The proposed algorithm takes the workflow as a DAG and number of resources available in the cloud referred as Virtual Machines (VMs) as an input. It also takes some algorithm specific inputs such as the generation size, population size and mutation rate. The details of our proposed algorithm are given in the following subsections.

Individual representation and initial population generation

Individual S is represented as a two-dimensional array. $S[i]$ includes an array list of tasks that are assigned to VM_i and $S[i][j]$ denotes the ID of the task that is assigned to

VM_i. The order of the tasks assigned to a VM does not represent in which order they will be executed.

VM_0	T_1	T_3	T_6	T_7
VM_1	T_2			
VM_2	T_0	T_4	T_5	

Figure 2. An example individual representation

An example individual representation is given in Figure 2. Individual S ={{1, 3, 6, 7 }, { 2 }, {0, 4, 5 }} includes 3 task sets, so all tasks are assigned to 3 different VMs. The first set {1, 3, 6, 7} shows that t_1, t_3, t_6 and t_7 are assigned to VM_0. The second set {2} contains a single task t_2 which is assigned to VM_1. Finally, the last set denotes that t_0, t_4 and t_5 will be executed in VM_2.

Initial population is generated randomly. Each individual in the population contains the assignment of tasks to VMs. While creating an individual, all tasks from t_0 to t_n are selected and assigned to a random VM. The makespan of the solution represented by each individual is calculated and assigned as the fitness value of that individual.

Algorithm 1: Crossover Operation

Input: Two individuals in the population as parents ($parent_1$, $parent_2$),
Output: An offspring

1 Find the mean value of sizes of the task sets in the chromosomes $size_{mean}$
2 Set the crossover point as floor($size_{mean}/2$)
3 Generate empty chromosomes $offspring_1$ and $offspring_2$
4 Copy the first floor ($size_{mean}/2$) elements in the task sets from $parent_1$ to $offspring_1$
5 Copy all the elements those are not included in $offspring_1$ from $parent_2$ to $offspring_1$
6 Copy the first floor ($size_{mean}/2$) elements in the task sets from $parent_2$ to $offspring_2$
7 Copy all the elements those are not included in $offspring_2$ from $parent_1$ to $offspring_2$
8 Randomly choose one of the offsprings and return it

Crossover operator

Crossover operator is performed on two individuals selected from the population referred as $parent_1$ and $parent_2$, and at the end of the crossover one offspring is generated as shown in Algorithm 1. Tournament selection is used while selecting the parents. In tournament selection, 3 individuals from the population are selected randomly and the individual with the best fitness value is assigned as $parent_1$. The same procedure is applied for selecting $parent_2$.

While generating the offspring, some part of the solution is taken from $parent_1$ and the other part is taken from $parent_2$, so a crossover point should be found. In this work, one point crossover is used. Each parent's task set sizes for each VMs are calculated. Half of the mean value of the task set sizes determines the crossover point.

The first offspring's task set elements come from the first parent until the crossover point is reached. The remaining part of the offspring's task set elements come from the second parent. The second parents task sets are checked from beginning to the end to determine whether they already exist in the offspring or not. If they are not available in the offspring, then they will be added to the task sets.

For the second offspring, the same operations will be performed by switching the order of the parents. Two offsprings are created and one of them is selected randomly and mutation is applied to this offspring.

To clarify the execution of the crossover operation, an illustrative example is provided in Figure 3. The size of the task sets for $parent_1$ is 4, 2 and 3. The size of task sets for $parent_2$ is 4, 4 and 1. The mean value of all the task set sizes is 3. Half of this value, which is 1, selected as the crossover point. All the elements before the crossover point are copied from $parent_1$ to $offspring_1$ and $parent_2$ to $offspring_2$. The remaining part is added from the other parent if it is not already included in the solution.

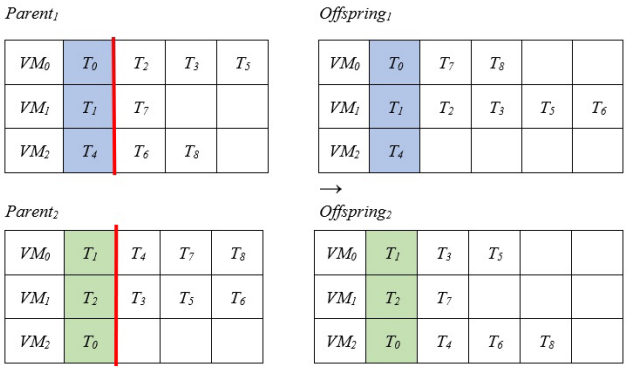

Figure 3. Crossover applied to parents to generate offsprings.

Mutation

Mutation is applied to the offspring generated at the end of the crossover operator. In this study, three different mutation techniques are proposed.

The first mutation operator is named as *Random Swap Mutation Operator (RSM)*. This operator randomly selects

two tasks from different VMs and swaps these tasks. As an example, Task 6 from VM_0 and Task 4 from VM_2 are selected randomly and are swapped in the offspring. In the final solution represented by the offspring, Task 6 is executed on VM_2 and Task 4 is executed on VM_0 as shown in Figure 4.

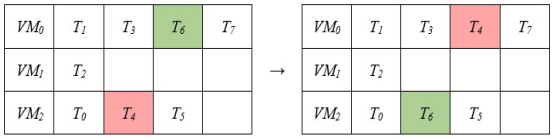

Figure 4. Random swap mutation operator

The second mutation operator is named as *Random Move Mutation Operator (RMM)*. This mutation operator randomly selects a VM and a position, removes the task in that position from the VM. It then assigns this task to a new VM where its new position is randomly set.

The final mutation operator is named as *Intelligent Move Mutation Operator (IMM)*. This operator again selects a task randomly and removes it from the VM it is assigned to. It then searches for the successors and predecessors of the task. The removed task is placed to a VM right after its predecessor or right before its successor to minimize communication overhead.

Fitness calculation and population update

Since the objective of this study is to decrease the makespan of the workflow, the fitness value is equal to the makespan of the solution present in the individuals. The individual with the smallest fitness value is reported as the best solution once the algorithm is completed.

For population update, elitism is used. Best individual in the population is added to the next generation. Crossover and mutation operations are performed to add new individuals to the population. All the new individuals are kept separately from the current population. Current population is replaced by the new population after all the new population members are generated.

Workflow simulator and scientific workflows

WorkflowSim [13] is a well-known framework for simulating workflows in the cloud environment. It is implemented using Java programming language. We used WorkflowSim as our workflow simulator in the experiments.

CyberShake, Montage, Epigenomics are realistic scientific workflows that are used to test the performance of our algorithm. These workflows are generated by Pegasus Workflow Generator [14]. The structure of these workflows is given in Figure 5. The CyberShake workflow is used to characterize earthquake hazards. The Epigenomics workflow is used to automate various operations in genome sequence processing. The Montage application stitches together multiple input images to create custom mosaics of the sky.

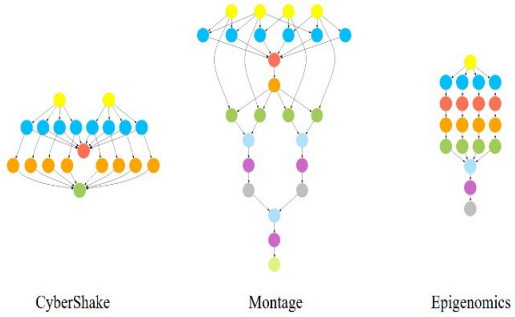

Figure 5. The structure of scientific workflows [15]

Results

The proposed evolutionary algorithm has some parameters which are population size, generation size and mutation rate. The first set of experiments show the performance of the algorithm for varying population size, the number of offsprings generated and the mutation operators. The proposed algorithm updates the whole population in each iteration; therefore, it generates new offsprings where the number of offsprings are equal to the population size in each iteration. The performance of the algorithm for varying population size with respect to number of offsprings generated is shown in Figure 6, Figure 7 and Figure 8. In each test, the population size varies between 25, 50, 100 and the experiments are performed on Montage scientific workflow for 25, 50 and 100 tasks. These tests are also performed on CyberShake and Epigenomics scientific workflows, and the results show similar trends.

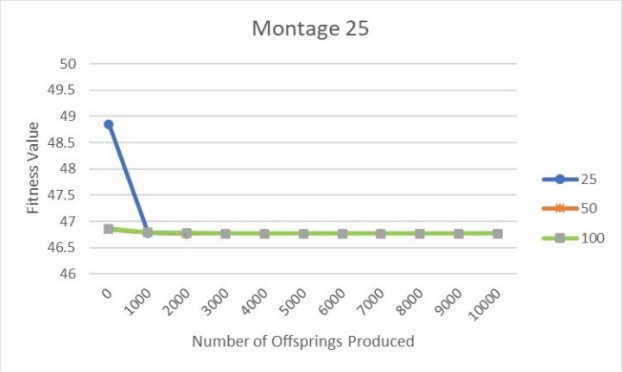

Figure 6. The effect of population size on the performance of the algorithm for Montage Scientific Workload with 25 tasks.

The fitness value of the initial population is higher when the population size is set to 25 for 25 tasks, and 25 and 50

for 50 and 100 tasks as compared to a population size of 100. This shows that as the number of individuals in the population increases, the chance to obtain better solutions also increases, which in turn leads to smaller fitness values. But as the algorithm continues to iterate, the fitness values of the population tend to decrease and finally reach near equal values. The time it takes to reach these results varies depending on the number of tasks considered. Therefore, the population size of the algorithm is set to 50 and the generation size of the algorithm is set to 2000 for smaller workflows which has 25 to 60 tasks and 5000 for larger workflows which has 100 tasks.

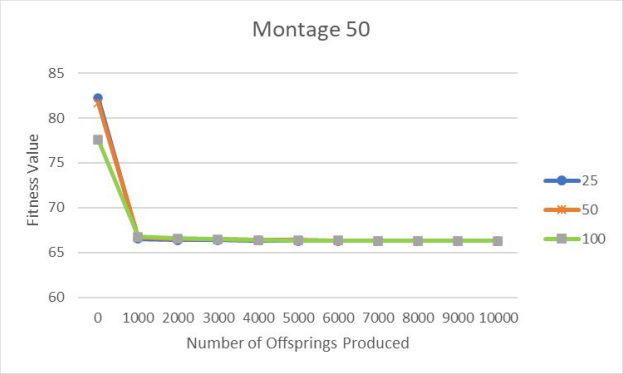

Figure 7. The effect of population size on the performance of the algorithm for Montage Scientific Workload with 50 tasks.

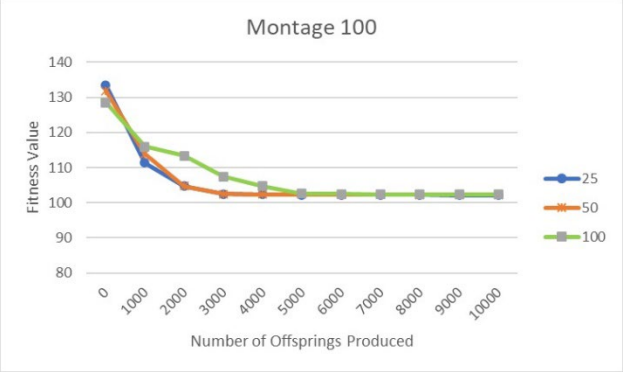

Figure 8. The effect of population size on the performance of the algorithm for Montage Scientific Workload with 100 tasks.

Next the performance of our algorithm is explored for different mutation strategies. Mutation rate is chosen as 0.2, 0.5 and 0.8. Depending on the previous studies from the literature, the number of VMs is selected as 20. The algorithm is executed for 10 runs and the best, average fitness values with their standard deviation are reported for different mutation operators with varying mutation rates.

The performance of the algorithm is first evaluated using 3 different mutation strategies. Once the performance of these mutation operators is evaluated, it is observed that depending on the structure of the workflow, different mutation operators can perform better in different cases, so hybrid mutation operators are added to the algorithm. Hybrid mutation operator chooses one of the mutation strategies depending on a given probability. Hybrid Mutation Operator 1 (HM1) uses Random Swap Mutation (RSM), Random Move Mutation (RMM) and Intelligent Move Mutation (IMM) with a probability of 35%, 35% and 30%, respectively. Whereas the probabilities of RSM, RMM and IMM are set to 45%, 45% and 10%, respectively for Hybrid Mutation Operator 2 (HM2).

The performance of these mutation strategies for different mutation rates using Montage scientific workflow are shown in Table 1. When the number of tasks considered is 25, all operators show the same performance and obtain the same results in all runs, and the standard deviation is 0. As the number of tasks are increased to 50, all operators except IMM show similar performance in most of the runs and standard deviation is low. When 100 tasks are used in the experiments, the RMM and HM2 operators show the best performance, whereas the performance of IMM is the worst. When the performance of RMM and HM2 operators on varying mutation rates is explored, they give the best average results when the mutation rate is chosen as 0.5. The performance of CyberShake scientific workflow has similar trend as can be seen from Table 2. Finally, the performance of the algorithm using Epigenomics scientific workflow is given in Table 3. All operators except IMM obtain the best results for 24, 48 and 100 tasks. When the average values are compared, RMM and HM2 operators with mutation rates 0.5 and 0.8 obtain the best results in all runs, therefore their standard deviation is 0.

From all these tests reported in Table 1, Table 2 and Table 3, we can conclude the following for mutation operators:

- IMM gives the worst result in all of the cases.
- Hybrid mutation strategies give the best result in most of the cases due to the reason that they use multiple strategies so that they explore the neighbor solutions.
- HM2 with mutation rate 0.5 gives the best result in most of the test cases.

Table 4 shows the performance of the algorithm for 3 different scenarios. “EA-WM” denotes the performance of the algorithm when only crossover is applied to the parents. EA-HM2 denotes the performance of the algorithm when HM2 with mutation rate 0.5 is used, whereas EA-Best shows the best result obtained from 5 different mutation operators with 3 different mutation rates. As can be seen from the table, selecting HM2 with a mutation rate of 0.5 gives the best result in 66% of the test cases, and gives good results in the remaining test cases. So instead of executing the algorithm 15 times for each mutation operator – mutation rate couple, HM2 with mutation rate 0.5 can be selected as the mutation operator in the proposed study.

Table 1. Performance of different mutation strategies using Montage Workflow

Mutation		# of Tasks								
		25			50			100		
Type	Probability	Best	Average	Std. Dev.	Best	Average	Std. Dev.	Best	Average	Std. Dev.
RSM	0.2	46.77	46.77	0.00	66.20	66.22	0.03	101.67	101.74	0.10
	0.5	46.77	46.77	0.00	66.20	66.21	0.01	101.66	101.69	0.02
	0.8	46.77	46.77	0.00	66.20	66.23	0.03	101.68	101.74	0.04
RMM	0.2	46.77	46.77	0.00	66.20	66.25	0.05	101.91	102.00	0.10
	0.5	46.77	46.77	0.00	66.20	66.23	0.02	101.86	101.90	0.03
	0.8	46.77	46.77	0.00	66.20	66.22	0.02	101.91	102.00	0.05
IMM	0.2	46.77	46.77	0.00	66.38	66.51	0.06	106.05	112.48	2.39
	0.5	46.77	46.77	0.00	66.33	66.47	0.08	102.42	108.72	5.31
	0.8	46.77	46.77	0.00	66.37	66.45	0.05	102.32	106.59	5.36
HM1	0.2	46.77	46.77	0.00	66.20	66.24	0.04	101.68	101.78	0.19
	0.5	46.77	46.77	0.00	66.20	66.21	0.01	101.68	101.75	0.08
	0.8	46.77	46.77	0.00	66.20	66.21	0.02	101.73	101.80	0.06
HM2	0.2	46.77	46.77	0.00	66.20	66.23	0.03	101.67	101.80	0.22
	0.5	46.77	46.77	0.00	66.20	66.21	0.02	101.67	101.69	0.02
	0.8	46.77	46.77	0.00	66.20	66.23	0.02	101.68	101.80	0.10

Table 2. Performance of different mutation strategies using CyberShake Workflow

Mutation		# of Tasks								
		30			50			100		
Type	Probability	Best	Average	Std. Dev.	Best	Average	Std. Dev.	Best	Average	Std. Dev.
RSM	0.2	262.20	262.20	0.00	283.25	283.25	0.00	303.56	303.99	0.15
	0.5	262.20	262.20	0.00	283.25	283.25	0.00	303.56	303.80	0.25
	0.8	262.20	262.20	0.00	283.25	283.25	0.00	303.56	303.66	0.20
RMM	0.2	262.20	262.32	0.26	283.25	283.25	0.00	303.56	306.26	2.71
	0.5	262.20	262.20	0.00	283.25	283.25	0.00	303.56	304.33	1.19
	0.8	262.20	262.20	0.00	283.25	283.25	0.00	303.56	303.85	0.24
IMM	0.2	262.20	262.32	0.26	283.25	283.56	0.27	327.06	340.75	8.34
	0.5	262.20	262.26	0.20	283.25	283.62	0.25	311.32	328.22	13.64
	0.8	262.20	262.26	0.20	283.25	283.41	0.25	308.57	314.54	3.67
HM1	0.2	262.20	262.20	0.00	283.25	283.25	0.00	303.56	303.75	0.24
	0.5	262.20	262.20	0.00	283.25	283.25	0.00	303.56	303.75	0.24
	0.8	262.20	262.20	0.00	283.25	283.25	0.00	303.56	303.75	0.24
HM2	0.2	262.20	262.20	0.00	283.25	283.31	0.16	303.56	303.80	0.25
	0.5	262.20	262.20	0.00	283.25	283.25	0.00	303.56	303.85	0.24
	0.8	262.20	262.20	0.00	283.25	283.25	0.00	303.56	303.61	0.15

Table 3. Performance of different mutation strategies using Epigenomics Workflow

Mutation		# of Tasks								
		24			48			100		
Type	Probability	Best	Average	Std. Dev.	Best	Average	Std. Dev.	Best	Average	Std. Dev.
RSM	0.2	5584.32	5584.33	0.01	7731.38	7731.41	0.02	32821.92	33038.76	322.91
	0.5	5584.31	5584.33	0.01	7731.37	7731.38	0.02	32821.92	32821.92	0.00
	0.8	5584.31	5584.33	0.01	7731.37	7731.37	0.00	32821.92	32821.92	0.00
RMM	0.2	5584.29	5584.29	0.00	7731.38	7731.38	0.00	32821.92	33223.44	250.52
	0.5	5584.29	5584.29	0.01	7731.37	7731.38	0.00	32821.92	33168.75	332.75
	0.8	5584.29	5584.29	0.00	7731.37	7731.38	0.01	32821.92	32923.61	101.99
IMM	0.2	5584.29	5584.37	0.05	7731.40	7778.41	148.47	41117.57	44919.36	2113.30
	0.5	5584.29	5584.34	0.05	7731.38	7731.45	0.07	41328.05	43611.06	1943.12
	0.8	5584.29	5584.35	0.04	7731.40	7731.45	0.05	43161.06	44015.77	783.13
HM1	0.2	5584.29	5584.29	0.00	7731.37	7731.38	0.01	32821.92	32822.15	0.48
	0.5	5584.29	5584.29	0.01	7731.37	7731.37	0.00	32821.92	32876.18	123.06
	0.8	5584.29	5584.29	0.00	7731.37	7731.38	0.01	32821.92	32822.03	0.35
HM2	0.2	5584.29	5584.29	0.00	7731.37	7731.38	0.00	32821.92	32885.28	200.35
	0.5	5584.29	5584.29	0.00	7731.37	7731.37	0.00	32821.92	32821.92	0.00
	0.8	5584.29	5584.29	0.00	7731.37	7731.38	0.00	32821.92	32821.92	0.00

Table 4. Performance comparison of the algorithm with/without mutation

Workflow	# of Tasks	EA - WM	EA - HM2	EA - Best
CyberShake	30	**262.199**	**262.199**	**262.199**
	50	**283.254**	**283.254**	**283.254**
	100	341.137	**303.562**	**303.562**
Epigenomics	24	5584.420	**5584.287**	**5584.287**
	48	7731.619	**7731.374**	**7731.374**
	100	42887.049	32821.916	**32821.915**
Montage	25	46.789	**46.765**	**46.765**
	50	66.743	**66.196**	**66.196**
	100	113.412	101.666	**101.658**

Table 5. Performance comparison of the proposed algorithm with algorithms from the literature

Workflow	# of Tasks	HEFT	Data Aware Scheduling	FCFS	Max-Min	MCT	Min-Min	Round Robin	EA-HM2
CyberShake	30	262.199	262.822	262.822	262.199	262.822	262.822	262.822	262.199
	50	283.254	283.771	283.771	283.254	283.771	283.771	283.771	283.254
	100	603.669	323.438	323.438	304.275	323.438	326.14	323.438	**303.562**
Epigenomics	24	5584.289	5584.37	5584.374	5584.289	5584.374	5584.374	5584.374	**5584.287**
	48	7731.41	7731.477	7731.459	7731.459	7731.459	7731.449	7731.459	**7731.374**
	100	32824.09	34963.04	34963.04	36947.03	34963.04	43018.52	40010.52	**32821.97**
Montage	25	46.765	46.773	46.773	46.765	46.773	46.773	46.773	46.765
	50	66.338	66.398	66.398	66.314	66.398	66.404	66.398	**66.196**
	100	102.296	102.146	102.252	102.052	102.252	102.274	102.156	**101.666**

Table 6. Comparison of the proposed algorithm with algorithms from the literature using t-test

Workflow	# of Tasks	HEFT	Data Aware Scheduling	FCFS	MAX-MIN	MCT	MIN-MIN	Round Robin
CyberShake	30	1	1.82E-233	1.8E-233	1	1.8E-233	1.82E-233	1.82E-233
	50	1	2.68E-229	2.7E-229	1	2.7E-229	2.68E-229	2.68E-229
	100	9.191E-55	1.943E-33	1.94E-33	2.829E-05	1.94E-33	1.9E-34	1.943E-33
Epigenomics	24	4.404E-08	1.232E-37	1.23E-37	4.404E-08	1.23E-37	1.232E-37	1.232E-37
	48	2.24E-181	1.98E-191	6.3E-190	6.27E-190	6.3E-190	5.96E-189	6.27E-190
	100	1.316E-49	1.73E-103	1.7E-103	1.3E-108	1.7E-103	1.09E-115	5.9E-113
Montage	25	1	5.75E-213	5.8E-213	1	5.8E-213	5.75E-213	5.75E-213
	50	4.121E-16	5.353E-19	5.35E-19	1.391E-14	5.35E-19	3.092E-19	5.353E-19
	100	1.577E-24	2.704E-22	6.17E-24	1.761E-20	6.17E-24	3.079E-24	1.826E-22

Finally the performance of the proposed study is compared with the algorithms from the literature as given in Table 5. The proposed algorithm outperforms the other algorithms in 6 of the 9 test cases, and gives the same performance in the remaining test cases. Our algorithm is especially better than the other algorithms when the number of tasks in the cloud computing system increases.

The statistical analysis of the performance comparison of the algorithms using t-test is shown in Table 6. In all scientific workloads when number of tasks reach 100, the proposed algorithm significantly outperforms all related studies.

Discussion

In this study 3 different mutation operators RSM, RMM and IMM are proposed. Since IMM uses the information of the predecessor and successor of the task that is moved from one VM to the other, it is assumed that it would be intelligent, would decrease the computation time and would perform the best among all the mutation strategies, but the experimental results showed the opposite where IMM gave the worst performance. These results denote that random movements instead of intelligent strategies perform better. When performance of the mutation operators that include randomness are explored, in some test cases RMM outperformed RSM, whereas in others it is the opposite. So, hybrid mutation strategies are proposed which show the best performance.

The proposed algorithm is compared with related studies from the literature and when the task size increases in the workflow, it is observed that the performance improvement of the proposed algorithm increases. Since in cloud computing systems, large number of tasks are considered, the proposed algorithm can provide a better solution. One disadvantage of our proposed study is its execution time. It runs much slower as compared to the studies used for performance comparison.

Conclusion

In this study, an evolutionary algorithm for workflow scheduling in cloud computing systems is proposed. The properties of the cloud resources and workflows are used to design the individual representation, crossover operator and mutation techniques. The main power of our algorithm comes from the selection of individual representation with problem specific information, the crossover operator that explores the search space successfully and mutation techniques that explores the neighbor solutions. The experimental study shows that the proposed algorithm outperforms related studies from literature and can provide better solutions for cloud computing.

The objective of the proposed study is to decrease the makespan of the workflow, as a future work the proposed algorithm can be extended to work with multiple objectives. The VMs considered in our study are homogenous, the computing power of the VMs can be heterogenous. Since in the individual representation, unique sets are defined for each VM, heterogeneity can easily be added to the algorithm. Finally, the proposed mutation operators and hybrid strategies can be used by other scheduling algorithms to increase their performance.

Ethics committee approval and conflict of interest statement

There is no need to obtain permission from the ethics committee for the article prepared.

There is no conflict of interest with any person / institution in the article prepared.

Authors' Contributions

-Study conception and design: Mehmet Kaya and Betül Boz

-Analysis and interpretation of data: Mehmet Kaya

-Drafting of manuscript: Mehmet Kaya and Betül Boz

-Critical revision: Betül Boz

References

[1] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task Scheduling in Cloud Computing based on Meta-heuristics: Review, Taxonomy, Open Challenges, and Future Trends", *Swarm and Evolutionary Computation*, 2021, 62.

[2] M. R. Garey and D. S. Johnson, "A guide to the theory of np-completeness", *Computers and intractability*, 1979, pp. 641–650.

[3] R. Zarrouk, I. E. Bennour, and A. Jemai, "A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem", *Swarm Intelligence*, 2019, pp. 1–24.

[4] N. Sadashiv, and S. D. Kumar, "Cluster, grid and cloud computing: A detailed comparison", *2011 6th International Conference on Computer Science & Education (ICCSE)*, 2011, pp. 477–482.

[5] S. H. H Madni, Latiff, M. S. A. Abdullahi, M., Abdulhamid, and M. Usman, "Performance comparison of heuristic algorithms for task scheduling in iaas cloud computing environment", *PLoS ONE*, 2017, 12: 5.

[6] A. Brandwajn, and T. Begin, "First-come-first-served queues with multiple servers and customer classes", *Performance Evaluation*, 2019; 130, pp. 51–63.

[7] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing", *IEEE transactions on parallel and distributed systems*, 2002, 13(3), pp. 260-274.

[8] B. Li, L. Niu, X. Huang, H. Wu, and Y. Pei, "Minimum completion time offloading algorithm for mobile edge computing", *IEEE 4th International Conference on Computer and Communications (ICCC)*, IEEE, 2018, pp. 1929–1933.

[9] F. Yao, C. Pu, and Z. Zhang, "Task Duplication-Based Scheduling Algorithm for Budget-Constrained Workflows in Cloud Computing", *IEEE Access*, 2021, 9, pp. 37262-37272.

[10] H. Aziza and S. Krichen, "A hybrid genetic algorithm for scientific workflow scheduling in cloud environment", *Neural Computing & Applications*, 2020, 32(18).

[11] M. Zhang, H. Li, L. Liu and R. Buyya, "An adaptive multi-objective evolutionary algorithm for constrained workflow scheduling in Clouds", *Distributed and Parallel Databases*, 2018, 36(2), pp. 339-368.

[12] G. Ismayilov and H. Topcuoglu, "Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing", *Future Generation computer systems*, 2020, 102, pp. 307-322.

[13] W. Chen and E. Deelman, "WorkflowSim: A toolkit for simulating scientific workflows in distributed environments", *2012 IEEE 8th International Conference on E-Science, Chicago, IL, USA*, pp. 1-8. doi: 10.1109/eScience.2012.6404430.

[14] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation", *Future Generation Computer Systems*, 2015, 46, pp. 17-35.

[15] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. -H. Su and K. Vahi, "Characterization of scientific workflows", *2008 Third Workshop on Workflows in Support of Large-Scale Science*, Austin, TX, USA, 2008, pp. 1-10, doi: 10.1109/WORKS.2008.4723958.