**Proposed Algorithm**

***Abstract Of Proposed Algorithm*:**

The approach is to minimize the overall completion time of all the tasks. To do so, different approaches are used, and their results are compared. The basis of all these heuristic and meta-heuristic techniques are same, **minimization of completion time**. Heuristic approaches used here are, Min-Min cloud scheduling algorithm and Max-Min cloud scheduling algorithm. Meta-heuristics include, use of genetic algorithm, with complete random population, then a population with Min-Min algorithm’s result as part of population, then another with Max-Min algorithm’s result as part of the population, and lastly with a population consisting of result of both Min-Min and Max-Min algorithm’s result.

The basis of comparison, i.e, completion time, is taken in terms of the maximum ready time from the given set of virtual machines for the particular instance. Also, the overall decline in completion time using different populations is also compared.

***Proposed Algorithm:***

The algorithm works by combining the heuristic principle of Min-Min algorithm for Cloud Computing and Max-Min algorithm for Cloud Computing and meta-heuristic technique of genetic computing. For all tasks, using both the heuristic approaches, all the cloudlets in the ETC Matrix are allocated to different machines. These tasks are allocated and scheduled keeping in mind, optimal consumption of time of completion. Then from these schedules, the allocation of each task is extracted. These schedules act as an input to genetic algorithm. The genetic algorithm is designed in a way that it considers the schedules with minimum completion time as the fittest individuals. Breeding technique used is Single Point Crossover (SPC), so as to get the better genes of all the individuals to get added to the overall population. to Generation after generation, there takes place mutation of all the individuals, this increases the possibility of getting close to an optimized schedule for given ETC. Hyper parameters of the genetic algorithm, when set optimally, result in a better optimization of the schedule. And setting these hyper-parameters depend mainly on the following: number of cloudlets to assign and number of machines to assign on. Aside that, the number of generations to run the algorithm for, depends on only one, that is, the saturation of minimization of completion time.

|  |  |
| --- | --- |
| **Notation** | **Definition** |
| ET(i,j) | Expected time to compute cloudlet i on machine j. |
| CT(i,j) | Completion time of cloudlet i on machine j. |
| T | Total number of cloudlets. |
| M | Total number of machines. |
| readyTime(j) | Ready time of machine j. |

The workflow of **Genetic Algorithm** is as follows:

* An **Initial population** is generated with a set of individuals.
* The population is tested for **Fitness**, the fitness parameter here being the completion time of all the tasks, so lower the fitness score, better the individual.
* On the basis of fitness score, and set survival rate, **Selection** of best individuals from the population is done.
* These selected individuals, then **Crossover** to form offspring. The technique used here is, Single Point Crossover.
* Based on the **Mutation** ratio, particular number of genes at random locations in the individual get modified.
* This is repeated over a number of generations until we get satisfactory results.

The workflow of **Min-Min** algorithm is as follows:

* If ready time isn’t given, initialize ready time of all VMs to 0.
* Compute the initial **Completion Time** of all the tasks in all the VMs.
* Assign a unique Task Id to all the tasks.
* For all the tasks in CT Matrix, find the task with minimum completion time, and the VM on which it does.
* Assign that task to the VM on which it takes minimum completion time.
* Update the ready time for that VM.
* Remove the task from CT Matrix.

The workflow of Max-Min algorithm is as follows:

* If ready time isn’t given, initialize ready time of all VMs to 0.
* Compute the initial **Completion Time** of all the tasks in all the VMs.
* Assign a unique Task Id to all the tasks.
* For all the tasks in CT Matrix, find the VM which gives minimum completion time for it.
* From all the tasks, find the task which takes maximum time to complete.
* Assign that task to the VM on which it takes minimum completion time.
* Update the ready time for that VM.
* Remove the task from CT Matrix.

**Algorithm:**

* obtain the output schedule using Min-Min and Max-Min algorithm.
* generate a completely random population.
* to that population, add output of Min-Min and Max-Min algorithm as individuals.
* **for** generation ← 1 to totalGenerations:

Calculate **Fitness** of population[generation]

**Select** parents for next generation

**Crossover** to get offspring

**Mutate** the offspring

**Update** the population with offspring depending on **survivalRatio**.

**end for**

**Pseudo Code:**

**Min-Min Algorithm:**

***if*** readyTime is not passed

readyTime = np.zeros(M)

***end if***

out = np.empty((ET.shape[0],3)

CT(i,j) = ET(i,j) + readyTime(j)

***for*** task in 1,2,…..T

TId = CTmin[0]

MId = CTmin[1]

TId\_ET = CT(TId,-1)

readyTime(MId) += ET(TId, MId)

out(task,0) = CT(TId, -1)

out(task,1) = Mid

out(task,2) = ET(TId\_ET, MId)

***for*** i in 1,2,….T

CT(i, j) += ET(TId\_ET, MId)

***end for***

Remove task TId\_ET from CT

***end for***

**Max-Min Algorithm:**

***if*** readyTime is not passed

readyTime = np.zeros(M)

***end if***

out = np.empty((ET.shape[0],3)

CT(i,j) = ET(i,j) + readyTime(j)

***for*** task in 1,2,…..T

TId = (CT[(:)max)min[0]

MId = (CT(:)max)min[1]

TId\_ET = CT(TId, -1)

readyTime(MId) += ET(TId, MId)

out(task,0) = CT(TId, -1)

out(task,1) = MId

out(task,2) = ET(TId\_ET, MId)

***for*** i in 1,2,….T

CT(i, j) += ET(TId\_ET, MId)

***end for***

Remove task TId\_ET from CT

***end for***

***Genetic Algorithm:***

***for*** generation in 1,2,…..G

fit = Call ***calc\_fitness***(pop, ET)

best\_results.append(fitmin)

parents = Call ***selection***(pop, fit, survivalRate)

children = Call ***crossover***(parents, 1-survivalRate)

***for*** i in 1,2,…..parents.shape[0]

pop(i, :) = parents(i,:)

***end for***

***for*** i in parents.shape[0]+1,…….pop.shape[0]

pop(i, :) = parents(i,: )

***end for***

overall\_fitness.append(fitness[0])