A Hybrid Approach of Memetic and Bees Life Algorithm for Multiobjective Workflow Scheduling in Cloud

Padmaveni Krishnan*  John Aravindhar

*Hindustan Institute of Technology and Science, India
* Corresponding author’s Email: kpadmaveni@hindustanuniv.ac.in

Abstract: Cloud computing is a paradigm that provides platforms for scientific applications. In clouds, the users pay based on the usage and the quality of service (QoS). There are many workflow scheduling algorithms in the heterogeneous computing environment. But all these cannot be applied in cloud due to the service based resource managing method in cloud. This paper discusses a workflow scheduling problem which gives an optimized solution considering make span and deadline as two objectives. A novel hybrid algorithm of Memetic algorithm and Bee swarm optimization algorithm called Memetic Bee Optimization algorithm (MBOA) is proposed. Experiments on randomly generated workflow and real time workflows show that the schedules generated by MBOA gives more stability on most of workflow instances. The results also show that the algorithm gives significantly better solutions than existing QoS optimization algorithms.

Keywords: Cloud computing, Memetic algorithm, Bee swarm optimization algorithm, Memetic bee optimization algorithm.

1. Introduction

Cloud computing has emerged as a computing paradigm that aims at providing computing services as scalable and virtualized resources to enormous remote users having heterogeneous requirement. Cloud provides computing services like networks, storage and services allocated from the resource pool with minimal management. The services are provided so that they follow the service level agreement. The instances in SaaS are virtual machines(VMs). The VMs help the customer in getting almost unlimited access to the resources and also plays a major role in reducing the Total Ownership Cost.

Any application that has jobs and flow of data among jobs can be described by a Workflow model [1]. Workflow scheduling problem is a problem of assigning jobs to processors in multiprocessor environment [2]. It is NP-complete and it can be represented as a Directed Acyclic Graph(DAG) in which the nodes are processes and edges are workflow among processors [3]. The edges show the data dependencies among jobs and they are directed [4]. Different workflows of Montage and Laser Interferometer Gravitational wave observation (LIGO) are explained by Luiz et al [5]. Montage application creates an image like sky that has about 17 hierarchical workflows and 900 sub workflows. LIGO represents a workflow that involves Tera bytes of data for producing the results.

Fig. 1 shows hybrid IaaS cloud having resources of private cloud and public IaaS cloud. The scheduling algorithm is run by the broker when the user submits a workflow. The broker determines the following:

i. The resources those will be used
ii. The part of workflow that will run on each cloud provider

Generally algorithms use QoS constraints and solves the problem by treating as single objective optimization problem. LOSS and GAIN [6] are algorithms that use a schedule and reassigns each job to another processor till it matches to budget. Algorithms like NSPO[7] use the Pareto Swarm
2. The common scheduling algorithms and challenges in IaaS platform

The workflow scheduling problem considered here assumes that the amount charged to a user is purely based on the resource utilized. POSH is an algorithm that has exponentially correlated the CPU cycles to cost. Our pricing model is based on two assumptions:

i. The total cost of task is the sum of cost of subtask.

ii. The cost cannot be changed when the service is under run.

The existing scheduling algorithms have challenges on the limitation of resource in the resource pool. List based heuristic algorithm finds the best assignment by traversing all available processors in the selection step of every task. But this cannot be applied every time in cloud scheduling since the resources are enormous and it is not possible to do such traversals every time. One of the well-known existing algorithm is Particle Swarm optimization based algorithm which defines the particle positions and velocities as matrices. Here the order is the no of tasks(n) by no of resources(m),ie.,m x n. The disadvantage here is the number of resources ‘m’ may be too large to handle.

Few genetic operators represent the mapping of task to resources by strings. But, the existing genetic approaches might not be suitable to the cloud environment always because the VM instances are not permanent since they may be allocated and deallocated anytime during execution.

Durillo et al.,[11] proposed a list based heuristics that can be used in cloud. This algorithm constructs an instance pool of limited size and hosts out the possible schedules in advance.

In the algorithm proposed, the pricing criteria is considered when the fitness evaluation is made and the algorithm is designed such that it does not depend upon any fixed pricing schemes.

3. The workflow scheduling Problem

3.1 Workflow definition

A workflow can be generally represented by means of Direct Acyclic Graph (DAG). Here Work flow WFLOW=(J,C) where J is the set of ‘n’ vertices or jobs. \( J = \{ J_0, J_1, \ldots, J_n \} \) and C is the set of edges or control dependencies. \( C = \{ (J_a, J_b) | J_a, J_b \in J \} \). Each control dependency is assigned with the weight that represents the quantity of data transferred among jobs. The weight assigned to the vertices are the execution
time of jobs in a processor. It is denoted by 
\( \text{ExecTime}(J_i) \).

Data \((J_i, J_j)\) denoted the transfer of data from job \( J_i \) to job \( J_j \). The data transfer is always unidirectional. Excluding the source job, all other jobs have predecessor. Every job \( J_i \) apart from the source has its predecessor \( J_k \) provided there is an edge from \( J_k \) to \( J_i \) ie., \( \text{Pred}(J_i)={J_j/(J_j,J_i)Є C} \). Since the source job will not have any predecessor, \( \text{Pred}(J_{source})=Φ \)

### 3.2 Cloud resource model

The infrastructure as service platform delivers computational resource through virtual machines. A virtual machine that is running is called as an instance. There are different range of instance types having different execution time of jobs and bandwidths in IaaS platform. It is assumed that a customer can get any number of instances. Hence the set of instances \( I=\{I_0, I_1, ..., J\} \) is infinite whereas, the set \( T=\{T_0, T_1, ..., T_n\} \) is the type of instances offered and it is fixed. Each job is mapped to one instance from the available type in \( T \).

Every workflow has a user defined deadline associated with it. This deadline is the maximum time limit to complete the execution in cloud environment. If the user doesn’t give the deadline, it is obtained by assigning all jobs sequentially to the fastest instance and calculate minimum execution time of the workflow.

Each instance type \( T_n \) has its own features represented by CPU Type \( CPU(T_n) \), Memory \( M(T_n) \) and Cost per time interval. It is also assumed that parallel execution of the jobs is also possible. The CPU instance plays a major role in the time taken. It specifies the configuration so that if the time taken to execute any job is half if the CPU instance is doubled. The time to run a of Job \( J_i \) on instance type \( T_j \) is represented by Eq. (1).

\[
\text{Time}(J_i) = \frac{\text{Unitime}(J_i)}{\text{CPU}(T_j)}
\]  

Where \( \text{Unitime}(J_i) \) is the time taken to execute the job \( J_i \) for unit CPU time.

Any instance type \( T_j \) will have its own bandwidth represented by \( bw(T_j) \). When the communication is among different instances, the bandwidth of both may be different. Here, the minimum among both bandwidth is considered to calculate the time so that the worst case time of communication can be calculated.

\[
\text{Time communication}(J_i, J_j) = \begin{cases} 
(\text{Unitime}(J_i) \text{data} & R \neq S \\
\min[bw(\text{InsR}),bw(\text{InsS})] & 0, \text{otherwise}
\end{cases}
\]  

Where \( \text{InsR} \) and \( \text{InsS} \) are different instances to which \( J_i \) and \( J_j \) are scheduled.

### 3.3 Scheduling problem

The leading cloud providers like Amazon EC2, IBM, Microsoft Azure, etc have different pricing schemes allowed. The algorithm designed is a generic one that flexible to fit for any pricing model. It is assumed that there are ‘\( k \)’ different pricing models. \( P=\{P_0, P_1, ..., P_k\} \). The function \( \text{COST}(I, P, M, T) \) will calculate the cost for the instance ‘\( I \)’ for the pricing model ‘\( P \)’ having the instance type ‘\( M \)’. The IaaS model is represented in general as \( S=\{I,P,M\} \) where \( M \) stands for instance type.

The goal is to find a schedule to execute a workflow on cloud computing environment such that total execution cost and makespan are minimized by meeting the deadline constraints. Hence the two main objectives in workflow scheduling are

i. Total Cost Minimization
ii. Makespan

#### 3.3.1. Total cost Minimization

For a workflow \( \text{WFLOW}=(J, C) \) and an IaaS \( S=(I, P, M) \), the idea is to produce more scheduling choice having instance, type and order. The order is the vector containing the scheduling order of jobs. The problem is considered with the constraint that the pricing scheme once chosen remains unchanged till the usage is completed. Hence the goal for the scheduler is to provide a schedule that has minimal cost among all possible schedules.

\[
\text{Total Cost} = \sum_{\forall T_i \in I} \text{cost}(I, P, T_i)
\]  

Here \( I \) is the instances used by the user, \( p \) is the pricing option which remains unchanged.

#### 3.3.2. Makespan

Makespan of a workflow is the total time elapsed from the start of the first job till the completion of the last job. Makespan will be less if more parallel execution is done.

Any schedule has Start time(\( \text{ST} \)) and completion time(\( \text{CT} \)). Start time of any job will depend upon the finish time of its previous job. If the job ‘\( J_i \) belongs
to instance $I_j$, it can be said that $Ins(J_i)=I_j$. The instance available time is mentioned in the vector $Avail(J)$. Initially it is 0 for all the available instances. When in execution, $Avail(J)$ will have the time when the instance will become free for next execution.

\[ ST(J_{first})=0 \]  
(4)

\[ ST(J_i)=\max\{Avail(Ins(J_i)), \max(CT(Time(J_i))+Time\ communication(J_j,J_i))^n \} \]  
(5)

If $J_{start}$ is the first job of execution and $J_{exit}$ is the last job to complete the execution in the workflow, $Makespan=CT(J_{exit})-ST(J_{start})$  
(6)

4. MBOA for optimization

4.1 Memetic algorithm

The memetic algorithm can be used to optimize several conflicting objectives by combining them as a single objective. The memetic algorithm is an evolutionary algorithm that simulates the natural evolution and is successful in the past. For the workflow scheduling problem, the memetic algorithm generates a set of schedules selected by the user constraints. The schedules are refined using the genetic operators and the solution is reached. The solution is generated in polynomial time and could be optimal or near optimal solution.

4.2 Bees algorithm

The Bees algorithm is also an optimization algorithm which is based on the life of Bees. The bee colony generally has a Queen bee ‘Q’, lots of male bees called Drones ‘D’ and several thousands of sterile females called workers ‘W’. [12]. The best fit schedule in any iteration is treated as queen ‘Q’. The crossover operation in any iteration is done with queen and Drones. The worker bees are those which are worst fit. These schedules are not fit for the next iteration.

4.3 Memetic bee optimization algorithm (MBOA)

The MBOA is a hybrid version of Memetic algorithm by blending the concepts of Bees algorithm also. The flow diagram of the hybrid version is in Fig. 2.

The constraints considered in this algorithm are
Figure 3 Example DAG workflow

i. The percentage of Drones and Workers need to be changed towards the end of solution. Towards the end, select more drones for quicker convergence to solution.

ii. New queen generation is identified by checking if the new generation has any schedule with <1% closer solution as initial queen.

iii. The termination criteria is that, if one region couldn’t give better solution continuously for N/5 iterations, stop proceeding with that region.

4.4 Fitness function

The fitness of any possible solution is the major factor for selecting the solution. Fitness function for this problem is a combination of both objectives mentioned in section 3.3.

\[ \text{Fitness} = \sum_{i=1}^{2}(w_i \cdot \text{Obj}_i) \]  

(7)

Where \( \text{Obj}_1 \) and \( \text{Obj}_2 \) are Total cost and Makespan respectively and \( 0 \leq w_i \leq 1 \).

4.5 Encoding procedure

The encoding procedure is used to map the cloud workflow as DAG. The problem encoding is done as follows. Sequence of job indices represents the Order of schedule. This can be like \( O_1, \ldots, O_n \) where \( O_{i+1} \) will start execution only after the completion of \( O_i \). The job_instance is an array of size ‘n’ where the \( i^{th} \) element represents the instance of \( i^{th} \) job. The instance_type is an array of size ‘m’ where the \( i^{th} \) element has the instance type of \( i^{th} \) instance. Fig. 3 is an example of DAG workflow.

Topological ordering using the following procedure:

i. Find the indegree of all vertices
ii. The first node to schedule is the node with indedge 0
iii. Remove that node and the edges corresponding to that and find the new node with indedge 0.
iv. Repeat the above step until all the nodes are ordered.

5. Memetic and bees algorithm operators

5.1 Crossover

The scheduling for jobs here is precedence constrained. The order should always be maintained in the dependencies between jobs. If \( J_i \) has to get the output of \( J_j \), then \( J_i \) should precede \( J_k \) in all the possible orders generated. The crossover used here is Partially Matched Crossover (PMX) shown in Algorithm 1. Given two schedules ‘A’ and ‘B’ the PMX randomly picks two crossover points.

This crossover point is used for the construction of next generation schedule. Here substring is a subsequence of job in the given schedule referred with starting and ending position. The crossover is performed by considering the following facts.

i. Same job should not repeat.
ii. The dependencies among job are maintained.

The procedure for Crossover operation is given in Algorithm 1:

i. Procedure Partially_Matched (Schedule A, Schedule B)
ii. \( n \) is the number of jobs
iii. Select a random value \( P1 \) between 0 and \( n-1 \)
iv. Select second random value \( P2 \) between 0 and \( n-1 \) where \( P1 \neq P2 \)
v. If \( P1 > P2 \), swap \( P1 \) and \( P2 \)
vi. From the Schedule A, copy the substring from position \( P1 \) to \( P2 \) as \( O_1 \).

vii. From the Schedule B, copy the substring from position \( P1 \) to \( P2 \) as \( O_2 \).

viii. For all alleles(Jobs) in substring \( O_1 \) and \( O_2 \)

ix. If the alleles in substring \( (A,0,P1) \) and substring \( (A,P1+1,n-1) \) does not contain entries from substring \( (A,P1,P2) \) and substring \( (B,P1,P2) \), then replace substring \( O_1 \) by \( O_2 \).

x. else repeat steps 3 to 9.

xi. For string \( O_1 \) and \( O_2 \)

xii. Find the remaining alleles in \( O_1 \) and \( O_2 \).

xiii. Place them from left to right.

xiv. Check if the dependencies among jobs are satisfied. If not repeat steps 3 to 12.
Algorithm 1: Partially matched crossover

5.2 Mutation

The mutation is a memetic operator that maintains alteration in one or more gene values shown in algorithm 2. It is an occasional random alteration of a value in a schedule with small probability.

i. Procedure Mutation( Schedule A)
ii. n is the number of jobs
iii. Select a random value P1 between 0 and n-2
iv. Select a random value P1 between 1 and n-1 and P2 > P1
v. Swap the alleles in positions P1 and P2 and generate new schedule .
vii. If not repeat step 3 to 6
viii. Check the fitness of each schedule. The best fit is the queen.
ix. End procedure

Algorithm 2: Mutation

5.3 Initial population

In any scheduling algorithm, the solution space is massive. The generation of initial population plays a major role in the convergence to the solution. For a problem size of ‘n’ jobs, the initial population of ‘n’ possible schedules are generated. Out of the ‘n’ schedules, first four schedule are based on

i. Topological ordering
ii. Earliest Finish Time algorithm
iii. Shortest Job First
iv. Minimal Cost Ordering algorithm

The remaining n-4 schedules are generated on random ordering. Any schedule generated will be considered only if the dependency is maintained. This is depicted in algorithm 3.

The four algorithms included in generation of initial population helps in quicker convergence towards the optimal solution. For every schedule generated, the instance-type and job to instance are mapped accordingly.

i. Procedure initial population
ii. n is the number of jobs
iii. m is the number of instance types
iv. Generate the first schedule by topological ordering based on indegree of any vertex.
v. Generate the second schedule by heterogeneous earliest finish time algorithm.
vi. Generate the third schedule by Shortest job first algorithm
vii. Generate the fourth schedule based on the descending order of cost.
viii. For i=1 to n-4
ix. Generate a random schedule ‘I’ so that the dependency is maintained.
x. Find the fitness of each schedule. The best fit is the queen.
xi. End procedure

Algorithm 3: Initial population

5.4 Complexity analysis

The complexity of memetic operators crossover and mutation are O(n^2) and O(n) respectively for n jobs. The checking of dependency and fitness is O(n). The time complexity of evaluation and generation each generation is O(n^2). For a graph of ‘n’ vertices, a maximum of n^2 edges exist. For ‘g’ iterations, the overall complexity is O(g n^2 ). Excluding the evolution, four schedules in initial population are generated based on algorithms. The complexity of these also needs to be included. The HEFT, Topological ordering, SJF and MCO have O(n^2) complexity each. The overall complexity is O(4n^2) + O(gn^2). Which is O(gn^2) in general.

6. Testing

6.1 Experimental workflow

The MBOA algorithm is assessed using standard workflows defined in Pegasus workflow management systems[13]. The workflow models used are Montage, Epigenomics, Cybershake, Sipht and Inspiral. [14]. It is shown in Fig. 4

6.2 Algorithms compared

MBOA is compared with two algorithms namely Particle Swarm optimization(PSO)[15] and Genetic algorithm(GA) [16].

PSO is an evolutionary algorithm based on particles (ie.,bird or fish) [17]. The particles have the ability to move around the solution space and reach to a solution [18].

GA is a meta-heuristic algorithm based on evolutionary ideas of natural selection and genetics.
It provides solution to optimization problems with its initialization, selection, and generic operators [20].

6.3 Experimental setup

The algorithm is implemented in Cloudsim 3.0. We assume only six types of instances taken from Amazon as sample. The processing time for each job is estimated based on their processing capacity and the instance.

The average bandwidth to transfer intermediate data between instances is comparable to the average bandwidth provided by Amazon elastic block store (20 kbps). Our experiment has considered a billing period of 10 minutes and the acquisition delay is estimated as 1 minute.

We have considered six sizes of scientific workflow in our experiments small (approx. 20 Jobs), medium (approx. 50 Jobs), large (approx. 100 Jobs), xlarge (approx. 200 Jobs), 2xlarge (approx. 500 Jobs) and 4xlarge (approx. 1000 Jobs). We have conducted each experiment 10 times.

To evaluate the performance of the MBOA, we need the deadline of each workflow. The deadline is obtained from any of the following

i. The user

ii. By assigning all jobs sequentially to the fastest instance and calculate minimum execution time of the workflow.

The deadline is also recorded as the Fastest time. Assigning the jobs sequentially to the slowest instance gives the Slowest Time. Testing was done using four different deadline intervals of 1, 2, 3 and 4. Deadline interval is calculated by finding the difference between fastest time and the slowest time and then dividing it by five. Say, if the slowest time is 50 ns and the fastest time is 100 ns, the difference is 50 ns. Here the interval is 10 ns. The deadline interval 1 is 60 ns, deadline interval 2 is 70 ns and so on.

The IaaS parameters used for different virtual machines are mentioned in the table 1 (source www.aws.amazon.com). Here L, M and H represents the Low, Moderate and High in Network Preference.

7. Results and analysis

Graphs are plotted to show the deadlines met. Comparison graph is generated based on the percentage of fitness given by each algorithm.

<table>
<thead>
<tr>
<th>Inst type</th>
<th>CPU</th>
<th>Memory (GiB)</th>
<th>PIOP Opzd</th>
<th>N/W Pref</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Db.m1. small</td>
<td>1</td>
<td>1.7</td>
<td>-</td>
<td>L</td>
<td>$0.044</td>
</tr>
<tr>
<td>Db.m1. medium</td>
<td>1</td>
<td>3.75</td>
<td>-</td>
<td>M</td>
<td>$0.087</td>
</tr>
<tr>
<td>Db.m1. large</td>
<td>2</td>
<td>7.5</td>
<td>Yes</td>
<td>M</td>
<td>$0.175</td>
</tr>
<tr>
<td>Db.m1. xlarge</td>
<td>4</td>
<td>15</td>
<td>Yes</td>
<td>H</td>
<td>$0.350</td>
</tr>
<tr>
<td>Db.m2. 2xlarge</td>
<td>4</td>
<td>34.2</td>
<td>Yes</td>
<td>M</td>
<td>$0.420</td>
</tr>
<tr>
<td>Db.m2. 4xlarge</td>
<td>8</td>
<td>68.4</td>
<td>Yes</td>
<td>H</td>
<td>$0.840</td>
</tr>
</tbody>
</table>
The fitness is compared among three algorithms namely

- Genetic Algorithm (GA)
- Particle swarm Optimization (PSO)
- MBOA

For the testing purpose it is assumed with 0.5 for $w_i$ in Eq. (7).

In case of the Montage workflow, MBOA gave a better result than GA and PSO. The GA gave better result for deadline interval 3 and 4. MBOA achieved 100% result in deadline interval 3 and 4. It gave 92% and 96% result in deadline interval 1 and 2 respectively. Comparing the results for Epigenomics, it is inferred that MBOA gives a better result than the other two algorithms.

In the Cybershake workflow model, it is found that all the three algorithms showed a decently better performance in deadline 1 and 2. MBOA gave 100% performance for deadline 3 and deadline 4.

While comparing the performance in the Sipht workflow, PSO gave a better performance than GA. But MBOA outperformed both.

In the Inspiral workflow model, MBOA showed the best of 92% for interval 1 & 2 and 100% for interval 3 & 4. GA came closer to MBOA having 80% accuracy in interval 1 & 2 and 100% for interval 3 & 4.

Comparing the values and graph in Fig. 5, the other two algorithms totally outperform GA. Out of the other two algorithms MBOA has proven better than PSO.

8. Further analysis

The performance was analysed for different instances. Inference says that the solution was generated with lower makespan most of the time. As future work, analysis can be done with hybrid algorithms of ACO or Modified PSO algorithm with genetic algorithm. An important point is the impact on the selection of initial population. The initial four algorithms selected has a major difference in time for the convergence to the solution. It helps in faster convergence towards solution.

When the algorithm was checked with the input condition of one VM of every type in each job. It was found that the algorithm takes higher execution time for longer jobs when compared to shorter jobs. When seeing the computational complexity, MBOA algorithm is better than the other algorithms like PSO.
9. Conclusion and future work

Many scheduling algorithms are available for multiprocessor architectures in cloud environment. But most of these have difficulty when directly applied in cloud. The algorithm developed overcomes the issues in the real-world cloud computing models.

Cloud computing provides high performance computing resources on demand for solving large scale scientific problems. To execute the large scale scientific application, cloud provider needs to have appropriate provisioning algorithm which helps the cloud provider in minimizing the cost and makespan. Towards this, MBOA is proposed.

For giving a solution to the multi-objective cloud scheduling problem, an encoding scheme that represents the scheduling criteria having the different instances of jobs and their types are modelled. The memetic operators like evaluation function, crossover and mutation are used along with the bees algorithm. The experiment is checked with the actual pricing model in Amazon EC2 and the results are promising.

The future work can be a hybrid algorithm of PSO or any other optimization algorithm using more than one pricing scheme. The issues like termination delay of virtual Machines can be considered. Further work can be done in execution of workflow deployed in different regions and the data transfer cost among different data centre can be considered.

References


