Indian Journal of Science and Technology, *Vol 9(45), DOI: 10.17485/ijst/2016/v9i45/106354, December 2016*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Memetic Algorithm for Multi-objective Workflow Scheduling In Cloud

K. Padmaveni and John Aravindhar

Department of Computer Science and Engineering, Hindustan Institute of Technology and Science, Rajiv Gandhi Salai, Old Mahabalipuram Road, Padur – 603103, Kelambakam, Chennai, Tamil Nadu, India; kpadmaveni@hindustanuniv.ac.in, jaravindhar@hindustanuniv.ac.in

Abstract

Objectives: Cloud computing is a service delivery over the internet where users pay based on the usage and the Quality of service (Qos). The cloud environment supports high performance computing based on protocols, which allow sharing of computation and storage. Scheduling in a cloud is the process of scheduling the virtual machines (VM) to meet the customer's request. **Methods/Statistical Analysis:** The proposed evolutionary algorithm called Memetic Algorithm (MA) takes makespan and total cost as two objectives and gives an optimal workflow schedule of jobs. **Findings:** The algorithm is testing with different IaaS parameters from Amazon. Results show that MA gives significantly better solution than other algorithms like Genetic Algorithm (GA) and Iasi Cloud Partial Critical Path (IC-PCP). The schedule generated by MA gives more stability on most of the workflow instances. **Application/Improvements:** The proposed model applied to schedule the VMs in a cloud in an effective way.

Keywords: Cloud Computing, Genetic Algorithm, IAAS Cloud Partial Critical Path, Memetic Algorithm, Optimal Workflow Schedule

1. Introduction

In the recent years, cloud computing has emerged as a computing paradigm that supports computing services to enormous remote users with heterogeneous requirement. The services are provided under the Service Level Agreement (SLA). The Virtual Machines (VMs) are the instances in SaaS. Using the VMs, the customer can almost get unlimited access to resources and also can lower the Total Ownership Cost (TOC).

Workflow model can describe any application that has jobs and flow of data among jobs. The workflow scheduling problem is NP-complete and it is a problem of assigning jobs to processors in multiprocessor environment[1-3]. It can be represented as a Directed Acyclic Graph (DAG) where the nodes are processes and the edges are workflow. The edges show the data dependencies among jobs.

Figure 1 shows hybrid IaaS cloud having resources of private cloud and public IaaS cloud.

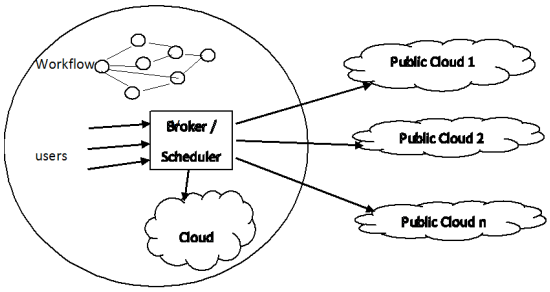

Figure 1. Hybrid IaaS cloud.

The work of service provider completed in two levels[4]:

- The infrastructure providers who rent resources.
- The Service providers, who charge resources.

Usually the algorithm use QoS constraints for converting the problem in the form of single objective optimization problem. The algorithm LOSS and GAIN[5] use a schedule

*Author for correspondence

and reassigns each job to other processor until it matches to budget. Few algorithms like NSPO[6,7] use the Pareto Swarm Optimization algorithm (PSO) to generate trade off among cost and make span. The Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT) extends the workflow in Amazon EC2.

Most of the above algorithms are tough to apply for cloud as they based on traditional heterogeneous environment like Grid. In this paper, we will see an optimization algorithm for cloud workflow scheduling problem. This algorithm generates a set of schedules having different trade-off between time and cost. The proposed algorithm is an evolutionary algorithm called memetic algorithm that has genetic algorithm as its ancestor. Memetic algorithm uses real world pay-per-use pricing strategies and it is based on IaaS instances. The memetic operators like encoding, evaluation function, initial population; crossover, mutation and reproduction are used.

The paper highlights the commonly scheduling algorithms and challenges on IaaS platforms in section 2. This is followed by scheduling problem definition and the pricing models in section 3. The memetic algorithm is explained in section 4. The paper is concluded in section 6.

2. Common Scheduling Algorithms and Challenges in IaaS Platform

The workflow-scheduling problem assumes that the amount charged to a user is based on the resource utilized by them. POSH assumes that the number of CPU cycles utilized by a user can be exponentially correlated to the cost. Two assumptions are made for our pricing model.

- The sum of cost of subtask is the total cost of task.
- The cost is fixed when the service is under run.

The challenge in the existing scheduling algorithms is, the resource pool is usually limited. The list based heuristic algorithm traverses all available processors in the selection step of every task and finds the best assignment. But this cannot be applied every time in cloud scheduling as the resources are enormous and it is impossible to do such traversals. One of the existing algorithm is Particle Swarm optimization based algorithm. This algorithm defines the particle positions and velocities as matrices. The order is no of tasks (n) by no of resources(m). ie. mxn. The disadvantage here is 'm' may be too large.

There are few genetic operators, which represent the mapping of task to resources by strings. The existing genetic approaches may not be suitable to cloud environment every time because the VM instances are not permanent and may be allocated and deal located anytime.

In[8] has proposed a list based heuristics used in cloud. This constructs a instance pool of limited size and hosts out the possible schedules in advance.

In this paper, we model the workflow-scheduling problem in cloud. The pricing criterion is considered when the fitness evaluation is made. The algorithm is designing such that it does not depend upon a fixed pricing scheme.

3. Workflow Scheduling Problem

3.1 Definition of Workflow

A general method to represent workflow is by means of Direct Acyclic Graph (DAG). Here Work flow WF=(J, D) where J is the set of 'n' jobs. $J=\{J_0, J_1,\ldots,J_n\}$. D is the set of edges or control dependencies. $D=\{(J_i, J_j)/ J_{i'jj} \in J\}$ The weight assigned to the edges of the graph is the quantity of data transferred between jobs. The vertices (Jobs) have the weights, which will be the execution time of jobs in that processor.

The execution time of every job is denoted as Exec time (J_i). The data transferred from J_i to J_j is denoted as Data(J_i,J_j). Every job J_i apart from the source has its predecessor J_k if there is an edge from J_k to J_i ie., $Pred(J_i)=\{J_k/ (J_k, J_i) \in D\}$. The source job will not have any predecessor. So $Pred(J_{source})=\Phi$

The DAG we consider has a single entry and single exit.

3.2 Cloud Resource Management

The infrastructure as service platform provides computational resource through virtual machines. The virtual machine that is running is called as an instance. The IaaS platform provides wide range of instance types having different execution time of jobs and bandwidths. Our assumption is that a customer is allowed to get any number of instances. So the set of instances $I=\{I_0,I_1,\ldots,\}$ is infinite. The set $T=\{T_0,T_1,\ldots T_m\}$ is the type of instances offered and it is fixed. Any job will fit on one instance from the available type in T.

$CPU(T_i)$represents the features of the instance type T_i. We assume that parallel execution of the jobs is also

possible. The time taken to execute any job is half if the CPU instance is doubled. The running time of Job J_i on instance type T_j is

$$Time(ji) = \frac{Unit\ time(ji)}{CPU(Tj)} \quad (1)$$

Where *Unit time*(*Ji*) is the time taken for executing job Ji for unit CPU time.

Any instance type Ti will have a bandwidth bw(Ti). When communicating among different instances, the minimum among bandwidth is considered so that the worst case for the time of communication can be taken.

$$Time(Ji, Jj) = \begin{cases} \dfrac{(Ji, Jj)data}{\min\{bw(TR), bw(TS)\}\ R \neq S} \\ 0, \text{otherwise} \end{cases} \quad (2)$$

Where TR and TS are different instances, to which Ji and Jj are scheduled.

The current cloud providers like Amazon EC2, Microsoft Azure, IBM etc have different pricing schemes. So the algorithm we have designed is a generic one that can fit for any pricing model. We assume that there are 'k' pricing models. P={P_0, P_1,….P_k}. Hence, the function cost (I_l, P_m, T_n) will calculate the cost for the instance 'l'using the pricing model 'm' having the instance type 'n'. So the IaaS model can be in general said as S=(I, P, M)

3.3 The Scheduling Problem

For a workflow WF=(J, D) and an IaaS S=(J, P, M), the solution is to produce more scheduling choice having instance, type and order. Here order is the vector containing the scheduling order of tasks. We consider that the user opts only one pricing scheme and does not change till the usage is complete. So the goal for the scheduler is to provide a schedule that has minimized cost where $\text{Total Cost} = \sum_{Tiel^*}^{n} \text{cost}(I_i, P, T_i)$ where I^* is the instances used by the user, p is the pricing option which will not vary[9].

4. Memetic Algorithm for Optimization

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 optimal solution is reached.

4.1 Fitness Function

The fitness of any possible solution is the important factor for selecting the solution. The two main objectives of our problem are Make span and Total cost. We should have the Start time(ST) and completion time(CT) for any schedule. Start time of any job will depend upon the finish time of its previous job. If the job 'Ji belongs to instance Ij, Ins(Ji)=Ij.

$$ST(J_{first})=0 \quad (3)$$

$$ST(Ji)=\max\{Ins(Ji), \max_{(Jj \in Preced(Ji))} (CT(Jj + Time\ communication(Jj, Ji))\} \quad (4)$$

4.2 Encoding

The problem is encoded as follows. Order is the sequence of job indices. This can be like O_1 , …..O_n where O_{i+1} will start execution after the completion of O_i. The job instance is an array of size 'n' where the ith element shows the instance of ith job. The instance type is an array of size 'm' where the ith element shows the instance type of ith instance. Consider the DAG in Figure 2.

The topological ordering using the in degree of every vertices of the above graph gives the Figure 3.

4.3 Memetic Operators

4.3.1 Crossover

The scheduling order should always maintain the constraints in the dependencies between jobs. If J_k has to get the output of J_i , then J_i should precede J_k in all the possible

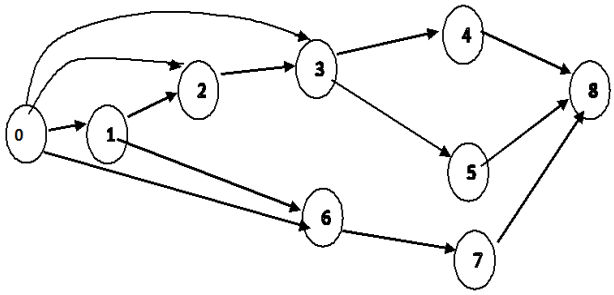

Figure 2. Example DAG workflow.

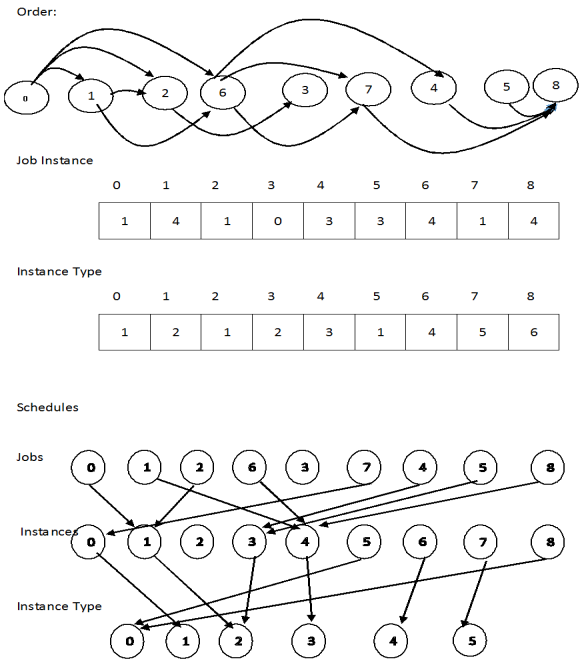

Figure 3. Encoding and scheduling for DAG example in Figure 2.

orders we generate. The crossover applied here is Partially Matched Crossover (PMX) shown in Algorithm 1. Given two schedules 'S1' and 'S2' the PMX will randomly pick two crossover points. This crossover point is used for the construction of child schedule. The crossover is done by considering the facts like the same job should not repeated and the dependencies among job are maintained.

1. Procedure Partially_Matched (S1,S2)
2. N ← number of jobs
3. P1 ← random value (0,n-1)
4. P2 ← randomvalue (0,n-1) where P1 ≠P2
5. If P1 > P2, swap P1 and P2
6. Substring O1 ← substring (S1,P1,P2)
7. Substring O2 ← substring (S2,P1,P2)
8. For all allels (Jobs) in O1 and O2
9. If allels in substring (S1,0,P1)and substring (S1,Pi+1,n-1) does not contain entries from substring (S1,P1,P2) and substring (S2,P1,P2), then replace substring O1 by O2,
10. Else repeat steps 3 to 9.
11. For string O1 and O2
12. Find the left out allels in O1 and O2.
13. Fill them from left to right.
14. Check if the dependencies are satisfied. If not repeat steps 3 to 12

Algorithm 1: Partially matched crossover

The algorithm works for our example graph as follows. The dependencies are 0→1→2→3→4→8

S1 0 1 |2 6 3|7 4 5 8

S2 0 1|2 3 4|5 6 7 8

Let P1 and P2 be 3 and 5. After executing steps 3 to 7 we get

S1 x x|2 3 4|x x x x

S2 x x|2 6 3|x x x x

The allele involved in the substring is: 2, 3, 4 and 6. Executing step 9 will give the following output.

O1 0 1|2 3 4|7 x 5 8

O2 0 1 |2 6 3|5 x 7 8

The step 10 and 11 says to find the left out allele of O1 and O2. In O1, the left out is 6. In O2, the left out is 4. Writing them from left to right, we get

O1 0 1 2 3 4 7 6 5 8

O2 0 1 2 6 3 5 4 7 8

The dependency is not disturbed in O1 and O2. So these can be the new off springs. If not the off springs are re-calculated. Then the corresponding instances and instance types are also updated.

4.3.2 Mutation

The mutation is a genetically operation that maintains alteration in one or more gene values. It is the occasional random alteration of a value in a chromosome with small probability. It is shown in Algorithm 2.

1. Procedure Mutation(S1)
2. N ← number of jobs
3. P1 ← random value(0,n-2)
4. P2 ← random value(1,n-1) where P2>P1
5. Swap positions P1 and P2 and generate new schedule
6. Check if dependency is maintained
7. If not repeat step 3 to 6
8. Check the fitness of new string
9. Repeat step 3 to 8 till a better solution is obtained or the number of iteration done is 'n'

Algorithm 2: Mutation

4.3.3 Initial Population

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

- Topological ordering
- Heterogeneous Earliest Finish Time (HEFT)
- Shortest Job First (SJF)
- Minimal Cost Ordering (MCO)

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

The inclusion of four algorithms in generation of initial population will help in quicker convergence towards the optimal solution. For each schedule generated, the instance-type and job to instance are mapped relatively. The procedure to generate initial population is in Algorithm 3.

1. Procedure initial population
2. N ← number of jobs
3. M ← number of instance types
4. Generate schecule1 by topological ordering based on in degree of any vertex
5. Generate schedule 2 by heterogeneous earliest finish time
6. Generate schedule 3 by Shortest job first algorithm
7. Generate schedule 4 based on the descending order of cost (MCO).
8. For i=1 to n-4
9. Generate a random schedule 'I' so that the dependency is maintained.
10. End procedure

Algorithm 3: Initial population

4.3.4 Complexity Analysis

The complexity of crossover and mutation are $O(\mathbf{n^2})$ and O(n) respectively where 'n' is the number of jobs. The checking of dependency and fitness is O(n). The time complexity of each generation is $O(\mathbf{n^2})$. For a graph of 'n' vertices, a maximum of $\mathbf{n^2}$ edges can exist. If we go for 'g' generations, the overall complexity is $O(\mathbf{gn^2})$. Apart from the evolution, we have used four schedules in initial population based on algorithms. The complexity of these also needs to be included. The HEFT, Topological ordering, SJF and MCO have $O(\mathbf{n^2})$ complexity. The overall complexity is $O(\mathbf{4n^2}) + O(\mathbf{gn^2})$. Which is $O(\mathbf{gn^2})$ in general.

5. Testing

a. Parameters

IaaS Model

The testing is done based on the instance specification by Amazon EC2

Workflows

Various workflows given by Pegasus workflow management systems are needed for testing

Montage

This workflow can be highly parallelizable as shown in Figure 4a.

5.1 Epigenomics

This is a type of workflow in which the given application is split into multiple paths and then combined. It is shown in Figure 4b.

5.2 Cybershake

The cyber shake workflow system has architecture as shown in Figure 4c. This is used by Southern California Earthquake Centre (SCEC)

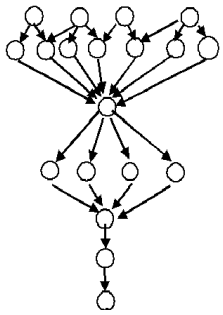

Figure 4a. Montage.

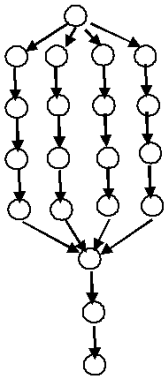

Figure 4b. Epigenomics.

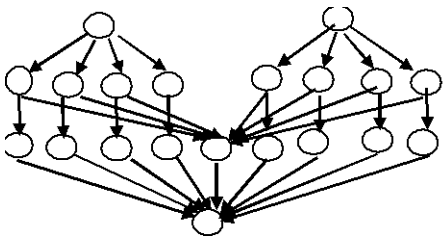

Figure 4c. Cybershake.

5.3 Sip Hit

This workflow will automate search and is shown in Figure 4d.

5.4 Inspiral

This workflow can be used to generate sequence from data[10,11]. It is shown in Figure 4e.

The IaaS parameters used for different virtual machines are mentioned in the Table 1 (source www.aws.amazon.com).

5.5 Algorithms compared

We have compared three algorithms namely IC-PCP[11,12], Genetic algorithm and memetic algorithm. For a graph with 'n' jobs having 'm' instances types, there are mxn instances prepared.

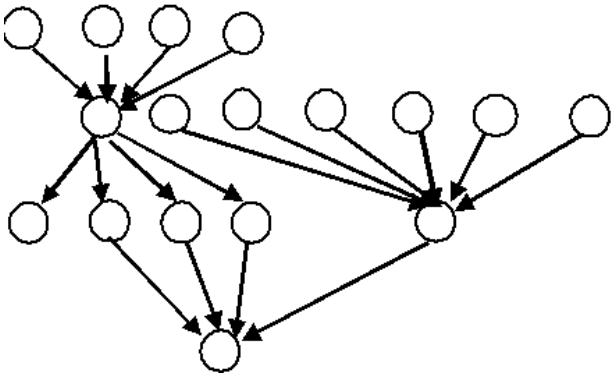

Figure 4d. Sip hit.

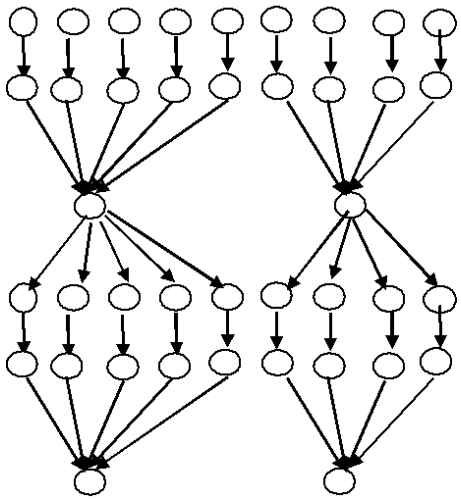

Figure4e. Inspiral.

Table 1. IaaS parameters used.

Instance type	vCPU	Memory (GiB)	PIOPS Optimized	N/W Preference	Price per hour
Db.m1.small	1	1.7	-	Low	$0.044
Db.m1.medium	1	3.75	-	Moderate	$0.087
Db.m1.large	2	7.5	Yes	Moderate	$0.175
Db.m1.xlarge	4	15	Yes	High	$0.350
Db.m2.xlarge	2	17.1	-	Moderate	$0.210
Db.m2.2xlarge	4	34.2	Yes	Moderate	$0.420
Db.m2.4xlarge	8	68.4	Yes	High	$0.840
Db.m2.8xlarge	32	244	-	High	$1.680

The following are the setups used.

- For IC-PCP we consider heterogeneous VM's that could be acquired on demand[13].
- For GA, the schedule is repeated for 12 times in order to check the effectiveness of convergence to solution.
- For Memetic algorithm, the schedule is repeated with the same constraints as GA.

We have assumed that the execution time of tasks are known in advance[14,15].

The testing was done using four different deadline intervals of 1, 2, 3 and 4. The deadline interval is taking by finding the difference between fastest time and the slowest time and then dividing by five to get and interval size. The deadline interval is obtaining by adding the interval size. Say, if the slowest time is 50 ns and the fastest time is 100 ns, the difference is 50 ns. The interval is 10ns. Now the deadline interval 1 is 60ns, deadline interval 2 is 70 ns and so on.

6. Results and Analysis

6.1 Deadline Constraint

We have plotted graphs to show how far the deadlines are met. The graph is plotted based on the percentage of deadline met. Graphs are shown in Figure 5a to 5e. For the Montage workflow, IC-PCP did not meet the deadlines. The GA gave a better result than IC-PCP. GA gave better

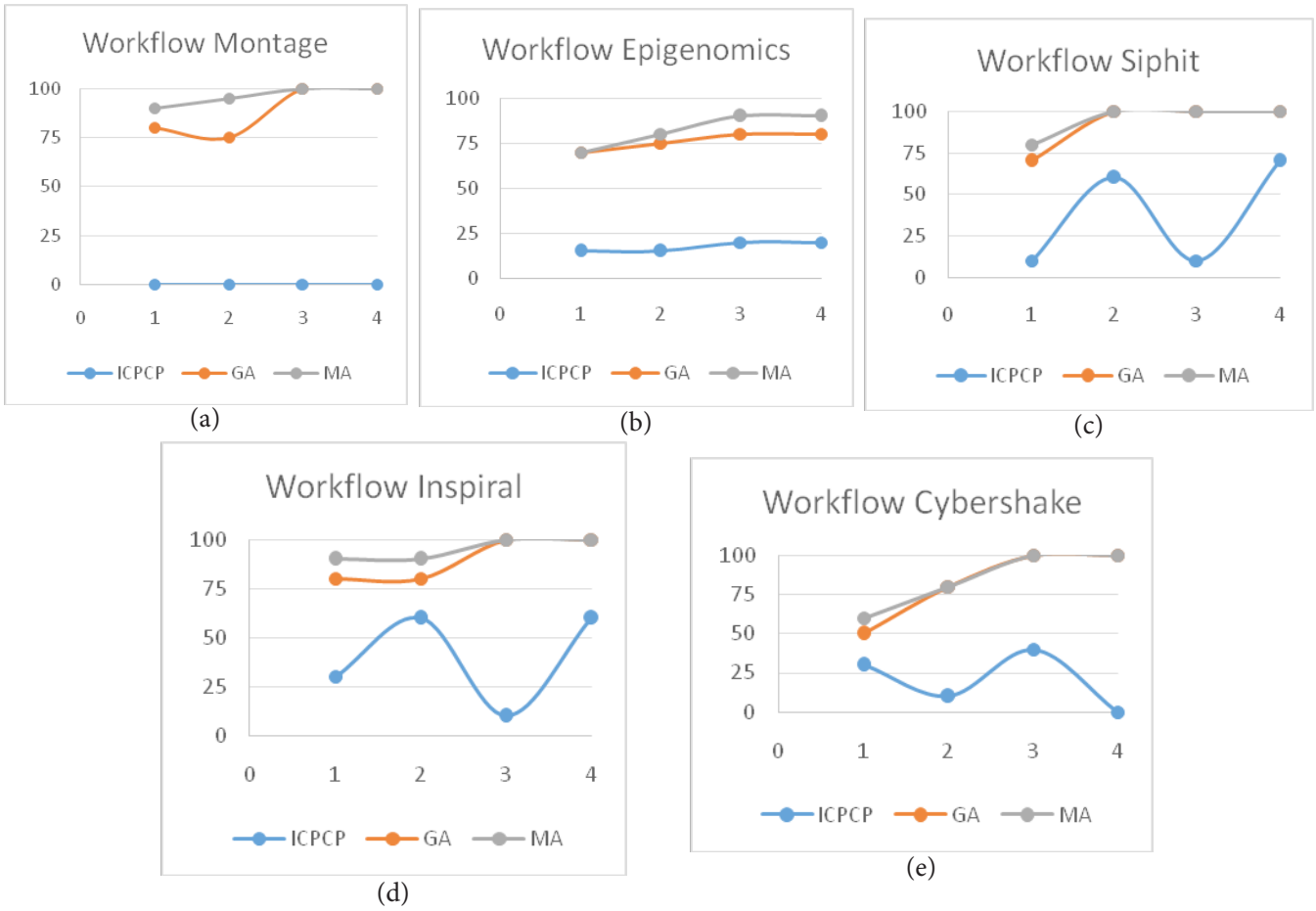

Figure 5. Deadline met **(a)** Montage **(b)** Epigenomics **(c)** Sip hit **(d)** Inspiral **(e)** Cybershake

result for deadline interval 3 and 4. The MA achieved 100% result in deadline interval 3 and 4. It gave 90% and 95% result in deadline interval 1 and 2 respectively. The results for Epigenomics also show that MA gives a better result than the other two.

While comparing the performance of all three algorithms in Cyber shake workflow, we found that IC-PCP gives slightly better performance than Montage and Epignomics. Still fails to meet the deadlines. Both the GA and MA showed a decently better performance in deadline 1 and 2. The performance was 100% for deadline 3 and deadline 4.

As for the Sip hit workflow, IC-PCP again met the least amount of deadline with the highest percentage being 70 for the deadline interval 2 and the other two algorithms gave better performance. For the In spiral workflow, the IC-PCP once again showed a poor performance of 10 % for the deadline interval 3. MA showed the best of 90 % for interval 1 & 2 and 100% for interval 3 &4. GA came closer to MA having 80% accuracy in interval 1 & 2 and 100% for interval 3 &4.

Observing the Figure 5a to 5e, we can say that the other two algorithms outperform IC-PCP. Out of the two evolutionary algorithms, MA has proven better than GA.

6.2 Makespan

Makespan is the longest time taken to generate the schedule[16]. The algorithm fails if the time taken to generate the schedule is longer than the given deadline. Here too we split into four deadline intervals based on the time of generating the schedule within the deadline. The deadline intervals 1 to 4 are defined based on the number of jobs. We have taken n=1 to 25 as deadline interval 1, n=26 to 50 as deadline interval 2, n=51 to 75 as deadline interval 3 and n=76 to 100 as deadline interval4.

The results show that the IC-PCP is not much efficient in generating the schedule within the given deadline. The other heuristic search algorithms are better than IC-PCP. For the deadline interval 1 and 2 GA generates the schedule quicker and for deadline interval 3 and 4 memetic algorithm generates the schedule quicker than GA.

7. Further Analysis

The performance was analysed for different instances. It was found that the solution was generated with lower makespan most of the time. As future work, the algorithm can be compared with other optimization techniques.

A noticeable point is the impact on the selection of initial population. The first four algorithms selected have a major difference in time for the convergence to the solution. It helps in faster convergence.

The algorithm was checked with the input condition of one Virtual machine of every type in each job. It was found that the algorithm takes more execution time for longer jobs compared to shorter jobs.

Overall, the memetic algorithm performs better than other algorithms. When seeing the computational complexity, the memetic algorithm is better than the other algorithms like PSO as memetic algorithm is heuristic based.

Further analysis can be done with hybrid algorithms of ACO or Bees algorithm with genetic algorithm.

8. Conclusion and Future Work

There are many scheduling algorithms for multiprocessor architectures in cloud environment. However, most of these are difficult to be directly applied in cloud. The algorithm we developed overcomes the issues as we have used the real-world cloud computing models.

In order to give a solution to the multi-objective cloud-scheduling problem, we have given an encoding scheme that represents the scheduling criteria, the different instances of jobs and their types. The memetic operators like evaluation function, crossover and mutation are used. 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 and memetic algorithm using more than one pricing scheme.

9. References

1. Yu J, Kirley M , Buyya R. Multi-Objective Planning for Workflow Execution on Grids, 8th IEEE/ ACM International Conference on Grid Computing, IEEE Computer Society, USA, 2007, p.10–17.
2. Chen WN, Zhang J. An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem with Various QoS Requirements, IEEE Transaction System Man Cybern. 2009; 39(1):29–43.
3. Zhang F, Cao J, Wu HK. Ordinal Optimized Scheduling of Scientific Workflows in Elastic Compute Clouds, 3rd IEEE International Conference on Cloud Computing Technology and Science, China, IEEE, 2011, p. 1–9.
4. Diallo L, Hashim AHA, Olanrewaju RF, Islam S, Zarir. Two Objectives Big Data Task Scheduling using Swarm Intelligence in Cloud Computing, Indian Journal of Science and Tecnology. 2016; 9(28):1–10.
5. Sakellaiou R, Zaho H, Tsiakkouri E, Dikaiakos M. Scheduling Workflows with Budget Constraints, Integrated Research in GRID Computing, Springer, 2007, p. 189–202.
6. Zuo X. Self Adaptive Learning PSO – Based Deadline Constrained Task Scheduling for Hybrid IaaS Cloud, IEEE Transactions on Automation Science and Engineering. 2014 Apr; 11(2):564–73.
7. Garg R, Singh AK. Multi-objective Workflow Grid Scheduling Based on Discrete Particle Swarn Optimization, Swarn, Evolutionary and Memetic Computing. 2011 Springer; 7076:183–90.
8. Durillo J, Pordan R. Multi-objective Workflow Scheduling in Amazon EC2, Cluster Computing. 2014; 17(2):169–89.
9. Zhu Z, Zang G.,Evolutionary Multi-Objective Workflow Scheduling in Cloud, Transactions on Parallel and Distributed Systems. 2015; 27(5):1344–57.
10. Abrishami S, Naghibzadeh M, Epema D. Deadline Constrained Workflow Scheduling Algorithms for IaaS Clouds, Future Generation Computer Systems. 2012; 19(3):680–89.
11. Bharathi S, Chervanak A, Deelman E, Mehta G. Characterization of Scientific Workflows, 3rd Workshop on Workflows in Support of Large Scale IEEE, 2008, p.1–11.
12. Juve G, Chervenak A, Deelman E, Bharathi S, Mehta G, Vahi K. Characterizing and Profiling Scientific Workflows, Future Generation Computer System. 2013; 29(3):682–92.
13. Rodriguez MA, Buyya R. Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Cloud, IEEE Transactions on Cloud Computing. 2014 Apr-Jun; 2(2):222–35.
14. Udomkasemsub O, Xiaorong L, Achalkul T. A Multiple Objective Workflow Scheduling Framework for Cloud Data Analytics, 24th IEEE International Joint Conference in Computing Sciences and Software Engineering, IEEE, 2012, p.1–8.
15. Ge Y, Wei G. GA Based Task Scheduler for the Cloud Computing Systems, International Conference on Web Information Systems and Mining, 2010, 2, p.181–86.
16. Eizadpanah E, Koroupi F. Timing of Resources in Cloud Computing by using Multi-Purpose Particles Congestion Algorithm, Indian Journal of Science and Technology. 2015; 8(8):474–83.