Asian Journal of Information Technology 15 (17): 3268-3279, 2016
ISSN: 1682-3915

A Fine Tuned Expedite Evolutionary Approach for Scheduling Deadline Constrained Scientific Application in Cloud

[1]S. Sindhu and [2]Saswati Mukherjee
[1]Department of Computer Science and Engineering, NSS College of Engineering, Palakkad, Kerala, India
[2]Department of Information Science and Technology, Anna University, Chennai, Tamil Nadu, India

Abstract: Recently cloud computing has been widely used by researchers for executing scientific applications. Cloud offers resources in the form of virtual machines which are dynamically provisioned and released and comes with a usage cost. This gives an illusion of abundance of resources available for execution. Hence, efficient scheduling mechanisms which satisfy the various quality of service requirements while minimizing the cost are the need of the hour. Although, scheduling algorithms which map a deadline constrained workflow to a cloud while minimizing cost have been studied, most of the time the makespan of the optimal schedule returned will be less than the deadline. Hence, a fine tuning of the optimal schedule thus returned will be useful and has the advantage of minimizing the under utilization of resources thereby achieving savings in cost. This study proposes an expedite genetic algorithm with fine tuning for scheduling deadline constrained workflow applications onto a cloud.

Key words: Workflow, scheduling, cloud computing, deadline, genetic algorithm, fine tuning

INTRODUCTION

High Performance Computing (HPC) is the use of parallel processing for running advanced application programs efficiently, reliably and quickly. HPC applications are widely being used by scientists for solving a large number of research problems. They exist in almost all disciplines. Examples of such applications can be found in the field of Molecular Dynamics (NAMD), astronomy (Montage), bioinformatics (mpiBLAST) and many more. These applications involve complex sets of computation and data analyses. Each of these computations may consist of thousands of steps and are distributed in the execution environment. The congregation and management of such complex computations pose numerous challenges. Workflows have evolved as the archetype for representing and managing complex distributed scientific computations thereby accelerating the pace of scientific progress (Taylor *et al*., 2014). The execution of such scientific workflows demands a high performance computing environment which is usually achieved by clustering of computing resources. Prior to the era of cloud computing, scientists largely relied on clusters and grid to test and run scientific workflows (Lee *et al*., 2009). However, the onset of cloud computing has brought in a dramatic change in the way scientific workflows are executed. Nowadays, cloud computing infrastructure is increasingly being adopted as a cost effective alternative to supercomputers, clusters and grid for running such large scale business and scientific workflows. Cloud services are gaining popularity because they reduce the cost and complexity of owning huge infrastructure and networks.

Cloud computing is a model of computing which refers to the delivery of computing resources over the internet. It is a model of utility computing characterized by on-demand service, elasticity, resource pooling and broad network access. Other benefits include efficiency, scalability and reliability. Cloud computing infrastructure can be used to test, run and deploy any kind of applications that are normally run on systems ranging from simple desktops to massive supercomputers. The computing resources are offered in the form of Virtual Machines (VMs). Recent studies (Deelman *et al*., 2008; Kondo *et al*., 2009) have shown that Amazon's Elastic Compute cloud (EC2) is widely being used by scientists for running scientific workflows. One of the main reasons behind this change is that cloud provides resources on-demand which is scalable and elastic thus making it conducive for testing and running such workflows.

Corresponding Author: S. Sindhu, Department of Computer Science and Engineering, NSS College of Engineering, Palakkad, Kerala, India

Table 1: Sample of amazon EC2 instance pricing

Instance type	CPU	Memory (GiB)	Cost per hour ($)
Small	1	2	$0.03
Medium	2	4	$0.07
Large	4	8	$0.13
Xlarge	8	15	$0.27

Deadline constrained workflows such as interactive deadline constrained e-learning, online media streaming applications and online banking systems are also pursuing favorable circumstances to utilize computer clouds. Among several others, a standout amongst the most imperative perspective which separates a cloud framework from its counterparts like cluster and grid is the market oriented business model. Amazon EC2 offers various types of instances on a rental basis and a sample of instance types offered by EC2 is shown in Table 1.

Under such circumstances, the execution cost of an application refers to the monetary cost involved in running the application. Therefore, as a user who wishes to deploy his/her application on a cloud this cost is of high significance. Moreover when the application comes with a deadline, there is a tradeoff between the optimal selection of the number of VM instances rented and the completion time. If faster instances are hired, the cost incurred will be more but with the guarantee that the application executes within the deadline. When slower instances are hired, the cost incurred will be less but user has to compromise on the completion time. Therefore an efficient scheduling algorithm which guarantees that an application executes within its deadline while at the same incurring minimum cost is of at most importance. Workflow scheduling is a well known NP-complete problem and many heuristics have been proposed in the literature for scheduling the same onto homogenous (Kwok and Ahmad, 1999) and heterogeneous distributed systems (Bajaj and Agrawal, 2004; Daoud and Kharma, 2008). Given this motivation, this study proposes a Genetic Algorithm (GA) based metaheuristic approach for scheduling a deadline constrained workflow onto a cloud such that the total cost incurred for its execution is minimal. GA is a part of metaheuristic algorithms which can find a good near optimal feasible solution in less time compared to traditional methods. They search in parallel from a population of points. Consequently, it can abstain from being caught in local optima like traditional methods which search from a single point.

Our proposed algorithm works in two phases. In the first phase, a deadline constrained workflow is scheduled onto a cloud using an expedite GA (EGA). Our system model assumes that there exist VM instances of various types (heterogeneous environment) with varying speed and cost. However, in order to arrive at an optimal schedule in considerably less time, we compute an upper bound and lower bound on the cost required for executing the workflow in a cloud. This computation is done by considering a homogenous environment where all VMs instances rented are of the same type. The variation of the cost when scheduled in heterogeneous environment with respect to these bounds determines whether exploration or exploitation of the search space is required. Accordingly either the crossover or mutation is applied. Even though an optimal schedule with minimum cost is returned by the EGA, most of the time it so happens that the makespan of the schedule is less than the deadline. This difference between the makespan and deadline gives room for further improvement. In the second phase, the optimal schedule thus found in the first phase is further refined. A fine tuning method is applied by clubbing two tasks in one VM instance. This is done because VMs are charged on an hourly basis. Therefore, a VM that executes for 20 min and another VM that executes for 40 minutes are charged the same (for 1 h). Hence, by clubbing these two VM instances such that the total time of these two instances are less than or equal to one hour, considerable saving in cost can be achieved. However when doing so, the precedence relation among the tasks is to be preserved.

We have compared our work with Standard GA (SGA) and PSO algorithms from the literature. The experiments have shown that EGA is able to meet 100% deadline in almost all cases and is able to achieve a significant reduction in cost. The key contributions of this study are:

- A novel way of applying crossover and mutation
- Computation of lower bound and upper bound for the optimization problem
- A novel fitness function based on the bounds
- A fast evolutionary approach called Expedite Genetic Algorithm (EGA)
- Fine tuning the optimal schedule
- Extensive simulation studies considering various cases

Literature review: Workflow scheduling aims at mapping and execution management of tasks which are dependent, to distributed resources. The problem of scheduling a workflow onto distributed systems has been well studied over the years and is known to be an NP-complete problem. Since, it is not possible to generate an optimal solution in polynomial time, heuristic and meta heuristic algorithms are used that focus on achieving an approximate or near-optimal solution. Algorithms for

scheduling workflow applications with various QoS constraints onto clusters and grid have been well explored (Chen and Zhang, 2009; Falzon and Li, 2012ab; Cao *et al*., 2010). For example, Yuan *et al*. (2009) tackles the cost optimization for scheduling a workflow that comes with a deadline constraint in a grid environment. They have proposed a heuristic called Deadline Early Tree (DET) which constructs an early feasible schedule and then a deadline division strategy is applied. For critical activities, an optimal cost solution under given deadline constraint is achieved by using a dynamic programming method.

Workflow scheduling in cloud has also been explored but relatively few works exist when compared to that available in clusters and grids. Abrishami *et al*. (2013), a Partition Balanced Time Scheduling (PBTS) algorithm is proposed which estimates the minimum number of computing hosts required to execute a workflow in a cloud within a user-specified finish time. But here a homogenous set of resources is considered so as to reduce the scheduling overhead. The heterogeneity of the resources offered by a cloud in not explored in this study. Yan *et al*. (2013) studied the problem of scheduling a deadline constrained workflow onto a hybrid cloud infrastructure (grid and cloud). They evaluated the degree of deadline-guarantee for subtasks of workflow in grid system based on a probabilistic deadline guarantee model. Then, proper cloud resources are selected as an accelerator to enhance the deadline-guarantee of subtasks. A similar study was done by Bossche *et al*. (2010) in which a Hybrid Cloud Optimized Cost (HCOC) scheduling algorithm was proposed which decides which resources should be leased from the public cloud and aggregated to the private cloud to provide enough processing power to execute a workflow within a given execution time. A novel compromised-time-cost scheduling algorithm which considers the characteristics of cloud computing to accommodate instance-intensive cost-constrained workflows by compromising execution time and cost with user input is proposed by Liu *et al*. (2010). Here, they concentrate on reducing the cost by compromising on execution time. A Revised Discrete Particle Swarm Optimization (RDPSO) is proposed in (Wu *et al*., 2010) to schedule applications among cloud services that takes both data transmission cost and computation cost into account. Here they try to achieve savings in cost and better performance on makespan. MOHEFT, a pareto based list scheduling heuristic that provides the user with a set of tradeoff optimal solutions from which the one that better suits the user requirements can be manually selected is presented by Durillo *et al*. (2012). The problem is formulated as a multi-objective optimization problem which aims at optimizing two conflicting criteria, namely makespan and economic cost of workflow's execution. The research is closely related to the work presented in (Rodriguez and Buyya, 2014) which uses a PSO based approach to minimize the overall workflow execution cost while meeting deadline constraints in a cloud environment. However, the work differs from their research in that a fine tuning strategy is applied in second phase after getting an optimal schedule. GA based algorithms have been used to address the workflow scheduling problem in grids and clusters (Yu and Buyya, 2006; Nesmachnow *et al*., 2010; Falzon and Li, 2012a, b). However in a cloud infrastructure most of the GA based algorithms try to reduce the makespan and cost. Our work considerably differs from the all the other works mentioned in the literature by computing bounds on execution time and execution cost which are used to guide the GA in arriving at an optimal solution in less time. Furthermore a second phase is applied to further refine the optimal schedule based on the pricing strategy which is typical of IaaS clouds.

Problem formulation: In this study, we briefly describe the system and application characteristics and summarize the various notations used in this research.

System environment: The system model consists of a cloud IaaS provider which offers compute nodes in the form of Virtual Machines to its users. We assume that a service similar to Amazon Elastic Compute Cloud EC2 is offered where different types of VMs which vary in computational speed, memory, cost per hour are available. These services are used to run the workflow. Also, it is assumed that a storage similar to the Amazon Elastic Block Store (EBS)is available to which the compute nodes are attached. This enables the storage of input and output data and also sharing of data among various tasks. A sample of instance types offered by EC2 has been shown (Table 1). Without loss of generality, researchers assume that the computational speed of a VM is expressed as MIPS (Million Instructions Per Second) and this information is available from the cloud service provider. This unit is used in our algorithm to determine the execution time of a task on a VM.

Let S_i denote the speed of a VM of instance type i. Assuming there are 'm' VM instance types in the system, the relationship between the processing capabilities of VMs can be specified as given in Eq. 1:

$$S_{i-1} \neq S_i \neq S_{i+1} i \in \{1,m\} \quad (1)$$

The cost incurred to run VM instances per unit time (C_i) is given by a table called VM Cost Matrix (VMCM) (Table 1). The relationship between the costs of VMs can be described by Eq. 2:

$$C_{i-1} \neq C_i \neq C_{i+1} \quad i \in \{1,m\} \tag{2}$$

We assume that there are several instances available for each VM type and these are kept in a VM pool. Each instance of a VM is assigned an id (VM_{xy}) for example VM_{13}. The first digit represents the type of VM (for example, 1 for xlarge, 2 for large and so on) and the second digit represents the id number of instance in that type. Hence, VM_{13} represents an instance numbered 3 of type 1. VMs usually take some time to boot. But since available and ready VMs are stored in the VM pool, this time is assumed to be zero in our work. Tasks are non preemptive and it is assumed that each VM instance can execute only one task at any given time. The VM instances are charged on an hourly basis (called unit time). Hence, if a task executes for 1 h and 10 min on a VM instance, it is charged for 2 h. In all cloud providers, VM instances executing tasks of a single workflow are assigned to a single data centre. Therefore, the bandwidth between the computing nodes is the same and hence, they do not charge for data transfers between two instances. Therefore in our work, the cost of data transmission is not considered while estimating the execution cost of the entire workflow.

Application characteristics: The workflow application is modeled as a Directed Acyclic Graph G = (V, E) where V is a set of 'n' tasks. The set:

$$V = \{T_1, T_2, ..., T_n\} \; ; i \in (1,n) \tag{3}$$

is represented as vertices in the DAG. The set of tasks T $\in$ (1, n) are dependent of one another and hence:

$$T_1 \cap n \; T_2 \cap n T_3, ..., n \; T_n \neq \emptyset \tag{4}$$

The interdependencies between the tasks are represented using the set E which is the set of directed edges. A directed edge $<T_i, T_j>E$ represents the dependency and the precedence between T_i and T_j. That is task T_i has to be executed before task T_j. T_i is called the parent of T_j and T_j the child of T_i. Based on this, researchers can say that a child task T_j cannot be executed until all its parent tasks have completed their execution. An example of a workflow is shown in Fig. 1.

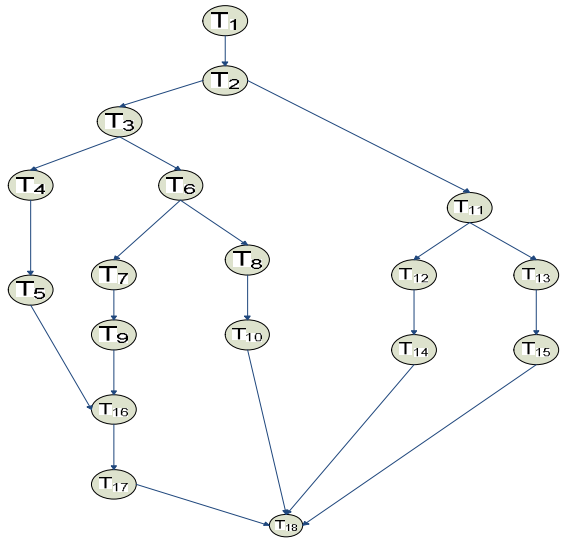

Fig. 1: A sample workflow represented as a DAG

Table 2: Summary of notations

Notations	Description
S_i	Speed of a VM instance of type I expressed in MIPS
C_i	Cost/unit time of VM instance of type i
V	Set of Vertices
E	Set of Edges
T_i	Represents ith task
l_i	Length of task i-expressed as Million Instructions (MI)
W_d	Deadline of workflow W
M_l	Lower bound of makespan
M_u	Upper bound of makespan
C_l	Lower bound of cost
C_u	Upper bound of cost
M_m	Mean of bounds for makespan
C_m	Mean of bounds for cost
I_{im}	Value of makespan for individual i
I_{ic}	Value of cost for individual i
F	Fitness of individual i

Each workflow comes with a user defined deadline W_d. A vertex in a DAG without any parent task is called an entry task and a vertex without any child task is called an exit task. In the sample workflow given in Fig. 1. T_1 is the entry task and T_{18} is the exit task. For DAGs which have multiple entry and exit tasks, our proposed method adds two dummy tasks T_{entry} and T_{exit} with zero execution time, since our algorithm works for workflows having a single entry and exit task. Table 2 summarizes the various notations used in this research.

MATERIALS AND METHODS

Block diagram of proposed model: Figure 2 shows the overall working of our proposed model. The user submits a workflow with deadline to the Workflow Management System (WFMS). The WFMS determines a schedule for executing the tasks of the workflow and gives it to the VM acquisition module. VM monitor monitors the availability

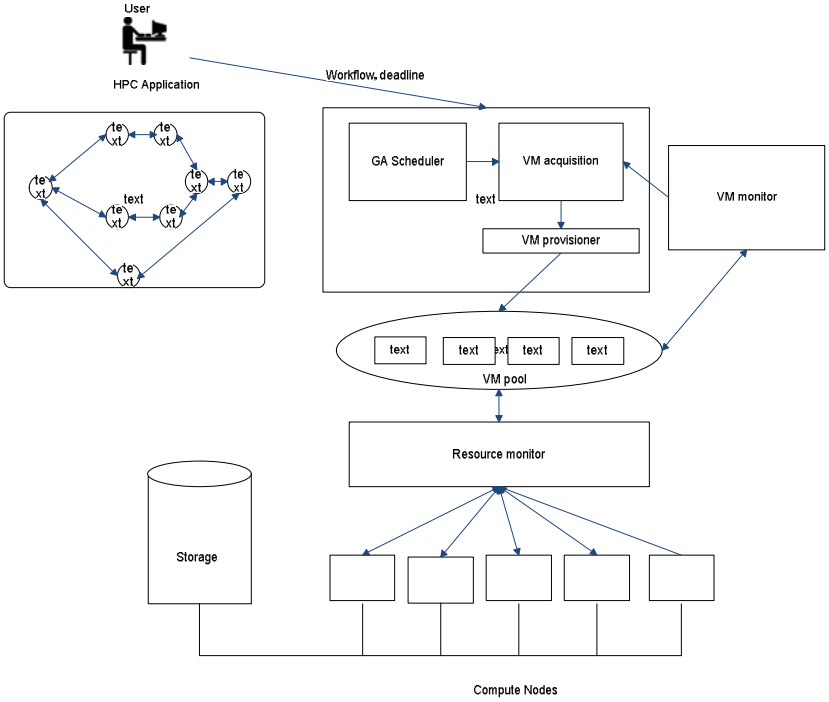

Fig. 2: Block diagram of proposed model

of VMs in the VM pool and gives it as input to the VM acquisition module. VM acquisition maps the tasks to the corresponding VMs. The VM provisioner module is responsible for the provisioning of VMs in the compute nodes. Resource Monitor monitors the execution of VMs on the compute nodes.

Calculation of lower bound and upper bound: Since, the problem of scheduling a DAG in a heterogeneous environment is NP-complete, it is difficult to find an optimal makespan satisfying the deadline constraint in polynomial time. In order to estimate the performance of our algorithm, a lower bound and upper bound for makespan and cost are computed for comparison purpose. To facilitate the calculation of these bounds, we relaxed some constraints in our system model. Researchers assumed that all VMs have a uniform configuration i.e., the environment is homogenous.

Researchers apply the Highest Level First with Estimated Times (HLFET) (Adam *et al*., 1974) scheduling policy to compute the lower bound and upper bound for makespan and cost. HLFET is a scheduling mechanism that uses static b-level as node priority and ignores communication cost on the edges. It includes the following steps.

- Calculate the static b-level of each node.
- Make a ready list in descending order of static blevel
- The ready list contains only the entry nodes initially
- Ties are broken randomly

Repeat:

- Schedule the first node in the ready list to a processor that allows the earliest execution, using the non-insertion approach
- Update the ready list by inserting the nodes that are now ready

Until all nodes are scheduled.

Lower bound for makespan (M_l) and upper bound for cost (C_u): In this test case, all VMs are supposed to be in the same configuration with highest speed possible. That is all are xLarge instances and hence cost incurred is comparatively high. The HLFET heuristic is used to compute the minimum expected makespan. For the generated schedule, the cost is calculated using the cost matrix. Since all VMs are xLarge instances, the makespan of the schedule gives the lower bound for makespan (M_l). Alternatively, since all are xLarge instances, the value of cost computed will be the highest possible total cost and this forms the upper bound for cost (C_u).

Upper bound for makespan (M_u) and upper bound for cost (C_l): Inorder to compute values for M and C same scenario as above is applied, except that all VMs are small instances and hence cost is comparatively less. Once the values of M_l, M_u, C_u and C_l have been computed, the mean of these bounds M_m and C_m for makespan and cost are calculated as shown in Eq. 5.

$$M_m = (M_l+M_u)/2;\ C_m = (C_l+C_u)/2 \quad (5)$$

MATERIALS AND METHODS

Proposed method: The algorithm works in two phases which are explained in this section.

Phase 1; Expedite Genetic Algorithm (EGA): The research focuses on finding a schedule to execute the tasks of a workflow which minimizes the execution cost while meeting the workflow's deadline (W_d). Initially, if the deadline $W_d > M_u$, then the workflow is not accepted. A feasible solution for a workflow scheduling problem is one which satisfies the following constraints

- The precedence relations among the tasks are satisfied
- A task appears once and only once in the schedule

A simple one dimensional string as shown in Fig. 3 (corresponding to the sample workflow in Fig. 1) is not suitable for representing a schedule of a workflow. Therefore, when a GA based concept is used to solve the workflow scheduling problem, we need to ensure that each solution satisfies the user constraints and task dependencies to prevent generating infeasible solutions. Generating solutions over multiple iterations that are feasible will take considerable amount of time and for a workflow with thousands of tasks this may result in poor performance. Hence a fast method to generate individuals which are feasible is of utmost importance. For this we divide the workflow into several levels based on the depth of interdependency. We associate a parameter called height along with each vertex in DAG. Researchers use a Breadth First Search (BFS) to traverse the DAG and compute height of individual vertices. The modified BFS (MBFS) to compute height of tasks in a DAG is given below. It also creates an inverted adjacency list for tasks in a DAG which stores the list of parents of each task.

Algorithm to compute height of tasks in a DAG:

```
MBFS(G)
    level=0
    T_entry.height=0    BFS(v)                    //first call to
BFS(v) is v=Tentry
    push v onto queue.
    While queue not empty
    do
            v=delete(queue)        visited[v]=true
            for each task T_j adjacent to v
    do
                    level=level+1
                    if visited[T_j]=false then
                                    push T_j onto queue
                                    T_j.height=level
                                    Append v to the predecessor list
of T_j
                    else
//when T_j has more than one parent. The parent which has the maximum
height is to be considered
                                    if T_j.height<level then T_j.height
= level
end
        end
```

All tasks with same height are in the same level. Tasks in the same level can be executed in parallel. Each task is represented as a tuple called the Task Attribute Vector (TAV) with the following fields.

Task Attribute Vector (TAV):

- Task id
- Height
- Vmid
- EST
- LFT
- Pointer to predecessor list

The EST and LFT represent the Earliest Start Time and Latest Finish Time of each task and are calculated as:

$$EST(T_i) = \begin{cases} 0 & \text{if } T_i = T_{entry} \\ \max\{LFT(T_k)\}; T_k \in pred(T_i) & \text{otherwise} \end{cases} \quad (6)$$

$$LFT(T_i) = EST(T_i)+E_i \quad (7)$$

Where, E_i is the execution time of task T_i.

Representation of a chromosome: Here, researchers use a 2 dimensional string to represent a chromosome and then map it to a 1 dimensional string as shown in Table 3 and Fig. 4.

Table 3: The 2 dimensional strings

Level	Tasks
0	T_1
1	T_2
2	T_3, T_{11}
3	T_4, T_6, T_{12}, T_{13}
4	T_5, T_7, T_8, T_{14}, T_{15}
5	T_9, T_{10}
6	T_{16}
7	T_{17}
8	T_{18}

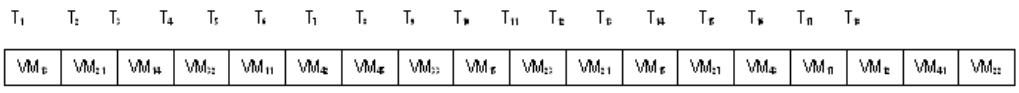

Fig. 3: One dimensional representation of workflow

0	1	2		3				4					5		6	7	8
T_1	T_2	T_3	T_{11}	T_4	T_6	T_{12}	T_{13}	T_5	T_7	T_8	T_{14}	T_{15}	T_9	T_{10}	T_{16}	T_{17}	T_{18}
VM_{13}	VM_{21}	VM_{14}	VM_{32}	VM_{11}	VM_{42}	VM_{45}	VM_{33}	VM_{15}	VM_{23}	VM_{21}	VM_{15}	VM_{27}	VM_{43}	VM_{17}	VM_{12}	VM_{41}	VM_{22}

Fig. 4: The structure of four realistic scientific workflows: a) Montage; b) LIGO; c) Cybershake; d) Epigenomics

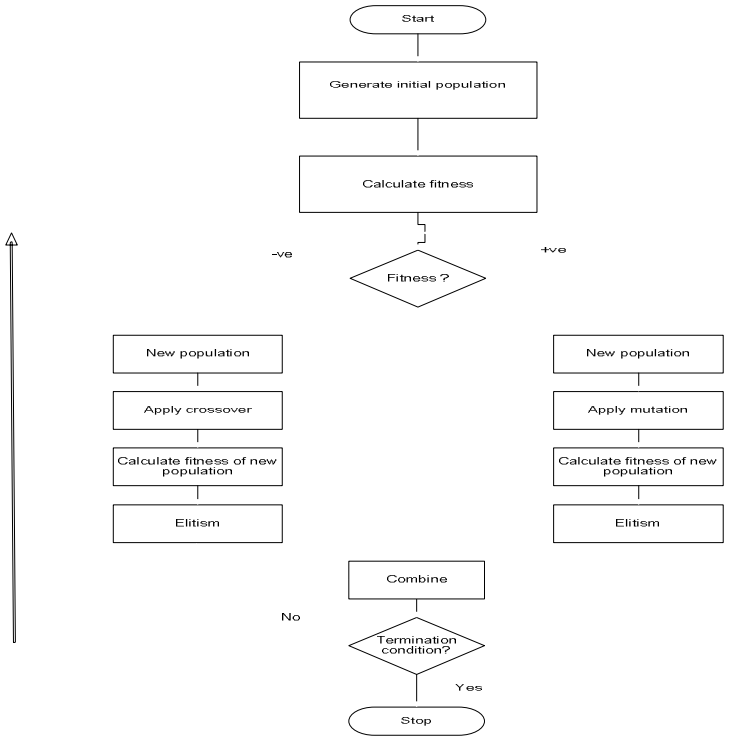

Fig. 5: Flowchart of EGA

A chromosome is a one dimensional vector divided into 'l' parts where l is the number of levels/rows of the 2d string. The values in the vector define the VM instance to which the task is assigned.

Initial population: The individuals in the initial population are generated through a random heuristic by assigning them to the computing nodes chosen at random. It is assumed that the tasks assigned to the same computing node are queued and are executed on a FCFS basis. In each individual the time slot at which the task is to be scheduled at each computing node (EST) is computed and included in the TAV and is determined by the following rules. Choose a ready task T_i that has all of its parents scheduled:

$$EST(T_i) = Max\ LFT(T_j)$$

where, T_j is a parent of T_i.

Fitness function: A fitness function in a GA is a measure of the quality of the individuals according to the given optimization criteria. Here our objective is to minimize the cost while at the same time meeting the deadline. In this work a novel fitness function is defined which measures the deviation of an individual's cost with respect to its mean value. The fitness is computed as shown below:

$$F_i = (I_{ic} - C_m) \quad (8)$$

Where, I_{ic} is the cost of individual I subject to $I_{im} \leq W_d$ A negavite value for F_i implies that I_{ic} lies between lower bound and mean whereas a positive value indiactes that it lies between mean and upper bound. A flowchart of EGA is shown in Fig. 5.

Genetic operators

Selection: Here, in the proposed research a proportionate based selection called roulette wheel selection is used.

After selection process, the entire population is divided into two groups. One group has a collection of all individuals with a negative value of fitness function while the other group consists of all individuals with a positive value of fitness function. Negative value of fitness F_i for an individual i means that $C_l<I_{ic}<C_m$ is satisfied. A positive value on the hand signifies that $C_l<I_{ic}<C_m$ is true. Hence, a crossover (exploitation of the search space) is applied to the first case and mutation (exploration of the search space) is applied to the second case.

Crossover: Crossovers are used to create new individuals in the current population. The basic idea behind crossover is that it may create better individuals by combining two fitter individuals. The crossover is implemented as follows:

- Two parents are selected at random
- Two random points are selected from the schedule of first parent
- The tasks between these two random points form the crossover window
- The computing nodes of all tasks in the crossover window are exchanged with their corresponding entries in parent 2

Mutation: In this process, an individual is picked at random. A child task which has multiple parents has to wait for all parents to be completed. Such a child task is chosen and LFT of parents are compared. Those parents which have a lesser LFT are scheduled on slower VM instances and the EST and LFT of the tasks are recomputed based on the new assignment.

Stopping criteria: The algorithm stops in one of two cases. Either an optimal solution is achieved when the value for F_i is equal to C_l or the best fitness value between several iterations are recorded and the algorithm stops when there is no significant difference among them over generations.

Phase 2; Refinement of Optimal Schedule (ROS): When an optimal schedule is obtained in phase 1 two cases are possible. Let M_0 denote makespan of the optimal schedule.

Case 1; ($M_0 = W_d$): This denotes the case when makespan returned by the optimal schedule is exactly equal to the deadline. Under such circumstances, no refinement is needed and the algorithm returns the optimal schedule. However, studies have shown that the makespan of the optimal schedule will always be less than the deadline. This necessitates a fine tuning of the optimal schedule which leads to the solution falling under case 2.

Case 2; ($M_0<W_d$): The optimal schedule returned by the GA is examined in a bottom up fashion. In the optimal schedule, each task of the workflow is assigned to exactly one VM instance. Therefore, the total number of instances rented for execution of the workflow is equal to the number of tasks in the workflow. In the second phase, the schedule is examined further to reduce this total number of VM instances rented. For this, the tasks of the workflow are examined in a bottom up manner starting with the exit task.

For a task which has only one parent (this information is encoded in the TAV (Fig. 4), the following computation is done (this means that the child task and the parent task are in sequential execution). Let T_c denote the child task and T_p denote the parent task. Assume they have been mapped to VM instances of type x and y, respectively.

Let U_x and U_y denote the unit cost of VM instances of type x and y (these are stored in a table called VMCM-Table 1). Let E_{cx} and E_{py} denote the execution times of T_c and T_p, respectively. (The x and y in the subscript denote the VM instance type). If any of the following condition holds:

$$E_{cx} \bmod U_x \neq 0 \text{ and } E_{py} \bmod U_y \neq 0$$
$$E_{cx} \bmod U_x = 0 \text{ and } E_{py} \bmod U_y \neq 0$$
$$E_{cx} \bmod U_x \neq 0 \text{ and } E_{py} \bmod U_y = 0$$

then the following computation is done to examine whether these two tasks can be clubbed in one VM instance. The cheaper of the two instances is chosen. Let us assume that VM instance of type x is the cheaper among the two:

$$D = (E_{cx}+E_{px}) \bmod U_x$$

If D = 0 then both are clubbed on one instance thereby reducing the number of instances. Based on this assignment, the EST and LFT of tasks are recomputed. If not, the workflow is further examined to identify any such tasks and the process repeats itself.

Experimental setup: This study elaborates the experiments conducted in order to evaluate the performance of our algorithm. We use a simulated cloud environment provided by CloudSim (Calheiros *et al*., 2011) to validate the approach.

In order to evaluate a workflow scheduling algorithm, its performance has to be measured against some sample workflows. Two commonly used approaches are:

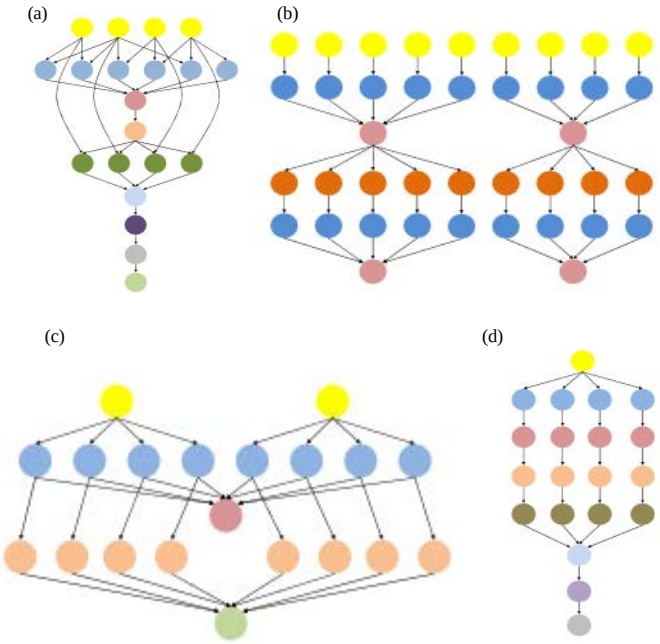

Fig. 6: The structure of four realistic scientific workflows: a)Montage; b) LIGO; c) Cybershake; d) Epigenomics

- Either uses a random graph generator to generate several workflows with varying characteristics like, number of tasks, dependency, level, etc
- Or use realistic workflows conventionally used in the scientific community

Here, researchers use the latter. Four different scientific benchmark applications from the Pegasus Workflow Generator, namely Montage, Laser Interferometer Gravitational Wave Observatory (LIGO), CyberShake and Epigenomics (GENOME) were used. Each of these workflows varies in their structure and computational requirements (Fig. 6).

The Montage workflow is an astronomy application used to generate custom mosaics of the sky based on a set of input images. The LIGO is an application in gravitational physics which aims to detect gravitational waves produced by various events in the universe as per Einstein's theory of general relativity. The CyberShake workflow is used to characterize earthquake hazards using the Probabilistic Seismic Hazard Analysis (PSHA) technique. The Epigenomics workflow is essentially a data processing pipeline that uses the Pegasus workflow management system to automate the execution of the various genome sequencing operations. In addition to these workflows, we simulate four types of VMs (refer table 1). We use three different workflow sizes for each application, namely, small (about 30 tasks), medium(about 50 tasks) and large (about 1000 tasks). Another important factor to be considered is the timing interval of a VM instance. We assume a timing interval of 1 h and VM boot time is ignored, since we assume that ready instances are available in the VM pool. Finally, to evaluate EGA we have to assign a deadline to each workflow. For this, the lower bound and upper bound of makespan is considered. These bounds represent the minimum (lower bound) and maximum (upper bound) makespan of a workflow in a homogenous environment. The difference between these bounds is divided by 5 to get a deadline interval size â. This interval size is added to the lower bound of makespan in the order $M_l+\beta$, $M_l+2\beta$, $M_l+3\beta$, $M_l+4\beta$, $M_l+5\beta$ to define 5 different deadlines for the application.. We use standard GA[ ], HEFT[ ] and PSO[ ] algorithms as baseline to evaluate our algorithm. In standard GA implemented in order to evaluate the algorithm, crossover is carried out in each iteration, whereas the mutation occurs with a probability of 0.5.

RESULTS AND DISCUSSION

Deadline evaluation: In order to analyze the performance of algorithms in meeting the user deadline, we plot the percentage of deadlines met for the 5 different deadlines. Since a large set of workflows with varying sizes and characteristics are used, we compute the average of makespan and then calculate the percentage of deadlines met over several runs. The results are displayed in Fig. 7.

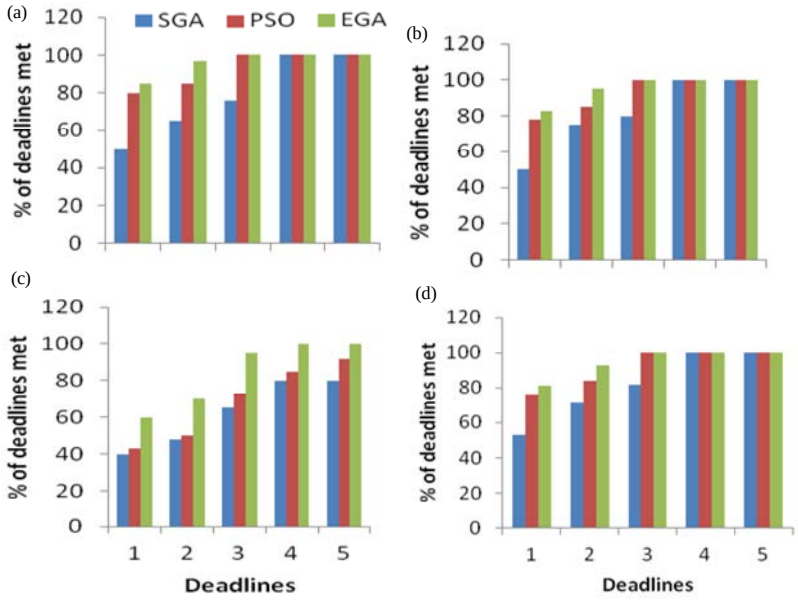

Fig. 7: Percentage of deadlines met for 4 different workflows: a)Montage; b) LIGO; c) Cybershake; d) Epigenomics

For the Montage workflow, SGA fails to meet the deadlines when they are strict, ie, the deadlines are close to the lower bound of makespan. When the deadlines defined were strict SGA achieves <50% in the first interval, <80% in the second and third intervals. But when moving to more relaxed deadlines, it achieves 100%. The same can be attributed for PSO and EGA but they exhibit a better performance than SGA in the initial cases also. This can be attributed to the fact that PSO and EGA converges faster and are able to meet the deadlines earlier than the SGA. Similar performance is achieved for the LIGO and Epigenomics workflow. As for the CyberShake workflow, both SGA and PSO fails to meet the deadlines when compared with EGA. This is because of the complex structure and characteristics of the workflow which has a large number of parallel tasks and dependencies. Under such a scenario, when the size if the individual is large, EGA performs better than PSO, since PSO assumes an initial boot time for VM instances which contributes a significant part in the completion time of the workflow. On the whole it has been found that both PSO and EGA are able to meet the deadlines in all the scenarios when compared with SGA. This is because, SGA takes more time to explore the search space as the constraints becomes stricter and is unable to return an optimal schedule.

Cost evaluation: Since, a large number of workflows with different characteristics are used, in order to effectively analyze the impact of our proposed approach on the cost of running the application, we define a Regulated Cost Factor (RCF) of a workflow as follows (Fig. 8):

$$RCF = \text{Total cost}/C_1$$

where C_1 gives the cheapest cost possible for executing a workflow. It is seen that the cost incurred by EGA is comparatively less than the PSO and SGA in all the cases. This is because even when the PSO meets 100% deadline, the makespan of the resulting schedule is always less than that of the deadline. Hence, as for the deadline satisfaction criteria, it performs 100% in almost all the cases and the workflows are able to execute before their assigned deadline. But when compared with EGA, EGA is also able to achieve 100% deadline satisfaction. But when it finds that the makespan of the resulting schedule is less than that of the deadline, a fine tuning method is adopted. This accounts for a lesser value in cost as compared to the PSO. Moreover, it can be seen that the cost is considerably high when the deadlines are strict and this gets reduced as deadlines are relaxed.

When deadlines are strict, there is little room for fine tuning and even if fine tuning is applied the instances rented are of high speed to meet the deadline. For the CyberShake workflow, both PSO and EGA incur almost the same cost. This can be attributed to the complex structure of CyberShake application where PSO fails to meet the deadlines and if it meets, the cost incurred is almost the same as that of EGA. The experiments were conducted on small and medium workflows and the results obtained were similar to the ones for large workflows.

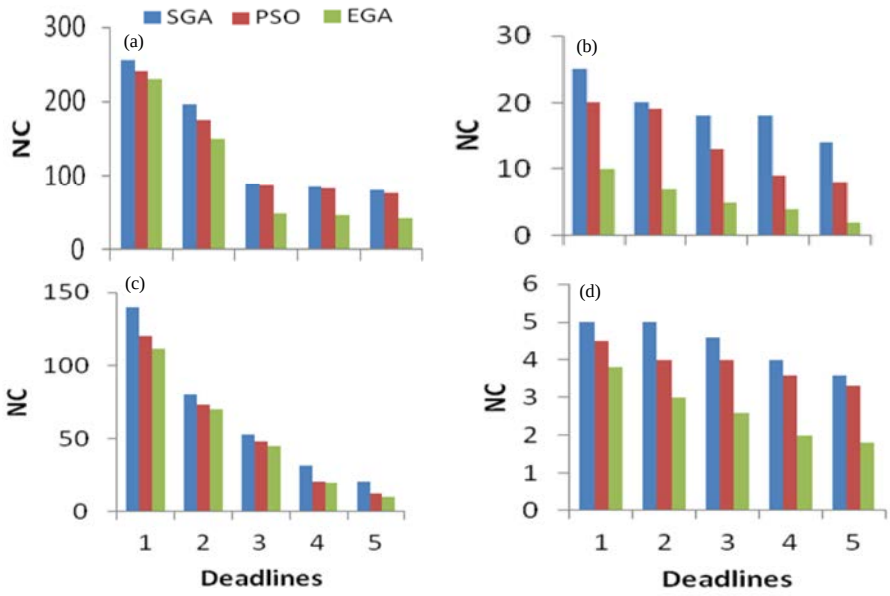

Fig. 8: Evaluation of Cost for 4 different workflows: a) Montage; b) LIGO; c) CyberShake; d) Epigenomics

CONCLUSION

This study presents an expedite genetic algorithm for scheduling workflow applications with deadline onto a cloud computing infrastructure. It has been observed that while scheduling workflows with deadline, most of the time while the optimal schedule satisfies the deadline criteria, the makespan of the resultant schedule is less than the deadline. This difference is of significance in a cloud environment where the resources are rented and considerable savings in cost can be achieved by renting les costlier resources. This process calls for a fine tuning method of the optimal schedule. The experiments conducted have shown that our proposed EGA with fine tuning is able to achieve considerable savings in cost when compared with already existing algorithms in the literature. As a future work, we would like to include data transmission cost among data centers so that an application may be deployed in different regions.

REFERENCES

Abrishami, S., M. Naghibzadeh and D.H.J. Epema, 2013. Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds. Future Gener. Comput. Syst., 29: 158-169.

Adam, T.L., K.M. Chandy and J.R. Dickson, 1974. A comparison of list schedules for parallel processing systems. Commun. ACM, 17: 685-690.

Bajaj, R. and D.P. Agrawal, 2004. Improving scheduling of tasks in a heterogeneous environment. IEEE Trans. Parallel Dist. Syst., 15: 107-118.

Bossche, R.V.d., K. Vanmechelen and J. Broeckhove, 2010. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. Proceedings of the 2010 IEEE 3rd International Conference Cloud Computing (CLOUD), July 5-10, 2010, IEEE, Miami, Florida, ISBN: 978-1-4244-8207-8, pp: 228-235.

Calheiros, R.N., R. Ranjan, A. Beloglazov, C.A.F. de Rose and R. Buyya, 2011. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Pract. Experience, 41: 23-50.

Cao, H., H. Jin, X. Wu, S. Wu and X. Shi, 2010. DAGMap: Efficient and dependable scheduling of DAG workflow job in Grid. J. Supercomputing, 51: 201-223.

Chen, W.N. and J. Zhang, 2009. An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. IEEE Trans. Syst. Man Cybernet. Part C: Appl. Rev., 39: 29-43.

Daoud, M.I. and N. Kharma, 2008. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J. Parallel Distrib. Comput., 68: 399-409.

Deelman, E., G. Singh, M. Livny, B. Berriman and J. Good, 2008. The cost of doing science on the cloud: the montage example. Proceedings of the 2008 ACM/IEEE conference on Supercomputing, November 15-21, 2008, IEEE Press Piscataway, NJ., USA., ISBN: 978-1-4244-2835-9, pp: 1-50.

Durillo, J.J., H.M. Fard and R. Prodan, 2012. Moheft: A multi-objective list-based method for workflow scheduling. Proceedings of the 2012 IEEE 4th International Conference Cloud Computing Technology and Science (CloudCom), December 3-6, 2012, IEEE, Taipei, Taiwan, ISBN: 978-1-4673-4511-8, pp: 185-192.

Falzon, G. and M. Li, 2012a. Enhancing list scheduling heuristics for dependent job scheduling in grid computing environments. J. Supercomputing, 59: 104-130.

Falzon, G. and M. Li, 2012b. Enhancing genetic algorithms for dependent job scheduling in grid computing environments. J. Supercomputing, 62: 290-314.

Kondo, D., B. Javadi, P. Malecot, F. Cappello and D.P. Anderson, 2009. Cost-benefit analysis of cloud computing versus desktop grids. Proceedings of the IEEE International Symposium on Parallel & Distributed Processing, May 23-29, 2009, IEEE, Rome, Italy, ISBN: 978-1-4244-3751-1, pp: 1-12.

Kwok, Y.K. and I. Ahmad, 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv., 31: 406-471.

Lee, K., N.W. Paton, R. Sakellariou, E. Deelman and A.A.Fernandes *et al.*, 2009. Adaptive workflow processing and execution in pegasus. Concurrency Comput. Pract. Experience, 21: 1965-1981.

Liu, K., H. Jin, J. Chen, X. Liu and D. Yuan *et al.*, 2010. A compromised-time-cost scheduling algorithm in SwinDeW-C for instance intensive cost-constrained workflows on cloud computing platform. Intl. J. High Perform. Comput. Appl., Vol. 2010, 10.1177/1094342010369114

Nesmachnow, S., H. Cancela and E. Alba, 2010. Heterogeneous computing scheduling with evolutionary algorithms. Soft Comput., 15: 685-701.

Rodriguez, M.A. and R. Buyya, 2014. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. Cloud Comput. IEEE. Trans., 2: 222-235.

Taylor, I.J., E. Deelman, D.B. Gannon and M. Shields, 2014. Workflows for e-Science: Scientific Workflows for Grids. Springer Publishing Company, New York, USA., ISBN: 9781849966191, Pages: 522.

Wu, Z., Z. Ni, L. Gu and X. Liu, 2010. A revised discrete particle swarm optimization for cloud workflow scheduling. Proceedings of the 2010 International Conference Computational Intelligence and Security (CIS), December 11-14, 2010, IEEE, Nanning, China, ISBN: 978-1-4244-9114-8, pp: 184-188.

Yan, C., H. Luo, Z. Hu, X. Li and Y. Zhang, 2013. Deadline guarantee enhanced scheduling of scientific workflow applications in grid. J. Comput., 8: 842-850.

Yu, J. and R. Buyya, 2006. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci. Program., 14: 217-230.

Yuan, Y., X. Li, Q. Wang and X. Zhu, 2009. Deadline division-based heuristic for cost optimization in workflow scheduling. Inf. Sci., 179: 2562-2575.