

A Report on

Delay-Based Workflow Scheduling for Cost Optimization in Heterogeneous System

*This report is submitted for the partial evaluation of the subject Cloud
Computing Laboratory of B. Tech Degree*

in

Department of Computer Science and Engineering

at

NATIONAL INSTITUTE OF TECHNOLOGY SIKKIM



Submitted By

Muhammad Kasim

Roll No.: B170126CS

B.Tech, 3rd year, 6th Semester

Concerned Faculty

Mr. Balaji Naik Sir

Assistant Professor

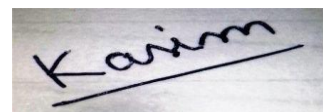
Department of Computer Science and Engineering

National Institute of Technology Sikkim

DECLARATION

I declare that the work presented in this report titled **“Delay-Based Workflow Scheduling for Cost Optimization in Heterogeneous System”**, submitted to the Department of Computer Science and Engineering, National Institute of Technology Sikkim, Ravangla, Sikkim, has been solely carried out by me. All the information used in this report is duly cited and directly not copied from any source.

If direct copying from any source is observed, I will be abiding by all the actions to be taken by the Department, against me.

A photograph of a handwritten signature in black ink on a light-colored surface. The signature appears to be 'Kasim' with a horizontal line underneath it.

Muhammad Kasim (B170126CSE)

B. Tech, 3rd year student

Department of Computer Science and Engineering

National Institute of Technology Sikkim

CONTENT

- Abstract
- Introduction
- Related Work
- Basic Terminologies and Problem Formulation
- Problem Definition
- Proposed Algorithm
- Psuedo Code
- Conclusion

Delay-Based Workflow Scheduling for Cost Optimization in Heterogeneous Cloud System

Abstract

Workflow scheduling has been widely adopted as a deep concern for heterogeneous cloud computing environment and recognized as a well-known NP-complete problem. The scheduling goal of any workflow scheduler is to map the user requests on the available virtual machines (VMs) which are deployed on various cloud servers such that overall processing time and incurred cost is minimized. This paper proposes a delay-based workflow scheduling algorithm for cost optimization in the heterogeneous cloud system. The proposed scheduling heuristic works in two phases. In the first phase, the proposed heuristic introduces a new priority scheme for the tasks. The prioritized tasks are then assigned to the appropriate VMs in the next phase. The algorithm is simulated on various synthetic workflows and two benchmark scientific workflows with different range of tasks. The experimental results of the proposed algorithm supersedes the existing scheduling algorithms in terms of make span, schedule length ratio (SLR) and cost.

INTRODUCTION

Workflow scheduling is recognized as a burning research problem of highly connected and inter-dependent tasks and it belongs to NP-complete class. The Workflow applications liaises to various real life and scientific domains i.e., massively large web graphs and social networks. The contemporary growth of virtualization in the area of computing, communication and storage [1– 3], a standalone or group of physical machines are able to create multiple virtual instances in form of virtual machines (VMs). These VMs can run simultaneously without interrupting to each other. The cloud users are charged by cloud service providers (CSPs) for renting the virtual resources (VMs) as per the pricing policies. However, the workflow applications always demand high computation power, network bandwidth and storage capacity due to the presence of precedence constrains. Therefore, cloud computing is a well suited paradigm which offers multiple set of services to the cloud user with the help of virtualization. By focusing on the cloud user's budget, the cloud service provider (CSP) is to provide an efficient and cost conscious schedule plan for the user request. The pricing policy differs from

VM to VM which means fast processing speed VM demands high charged (cost) while slower one offers lower VM rate [1, 2, 4]. For example Amazon EC2 offers 12 different types of VM instances for the cloud users [1, 5]. Therefore, a cost efficient resource provisioning and workflow scheduling is also accountable for suitable mapping of workflow tasks on the resources, so that the execution can be completed to satisfy criteria demanded by cloud users. Appropriate scheduling can provide a remarkable influence on the performance metrics (makespan, cost and energy) of the system [6–9].

In this paper, we address a cost conscious workflow scheduling algorithm by proposing a new priority scheme called delay based scheduling (ADAP) which is based on as-late-as-possible (ALAP) [10]. The new priority scheme ADAP measures the possible delay of task start time without increasing the schedule length. In general the proposed algorithm keeps running in two stages. In first stage the static task characteristic ADAP are calculated by visiting the directed acyclic graph (DAG) upwards from the exit task node. After that all the tasks are arranged in the decreasing order of ADAP at each level (li) of DAG. In the second stage the prioritized tasks in the priority queue (Qg) are scheduled according to earliest start time and earliest finish time. Our proposed algorithm uses heterogeneous nature of the tasks at each level that differs it from the original ALAP which provides the task prioritization for the entire workflow rather than the heterogeneity at each level. The main advantage of exploring the leveled behavior of the tasks is to balance the differences in completion time of the tasks at a level. This leads to minimize the difference in the starting time of the tasks at the next level. The performance of the proposed scheme is evaluated by comparing it with other four popular existing dependent task scheduling heuristics namely ALAP [10], HEFT [11], PETS [12] and FCFS in terms of performance parameters.

Residual part of the paper is orderly arranged as follows. Section II consists the related work as per the recognition of the research community. Problem statement and basic terminologies are described in Section III followed by Section IV in which we present the proposed algorithm. Section V has been systematized by the simulation results and performance assessments after that Section VI concludes the paper.

RELATED WORK

In recent years, several heuristics as well as metaheuristics approaches have been proposed to deal with the problem of workflow scheduling for cost optimization. Tabu search and

Simulated annealing were explained to solve the scheduling problems in Grid as well as cloud computing environment. In [13], multiobjective optimization based scheduling is explained by considering makespan and energy which uses two-phase genetic algorithm that involves multi-parent crossover parameter. In [14], Improved Genetic Algorithm (IGA) is explained which perform two times better than simple GA and it selects the VMs on the basis of dividend policy. In [15], Particle Swarm optimization based workflow scheduling is discussed with cost as the single objective. The cost model consider execution cost as well as communication cost, however it does not considers data transfer cost among the tasks. Also, the results were not compared with standard algorithm, and the algorithm were not simulated over scientific workflows.

Many research works have been proposed on workflow scheduling and resource provisioning algorithms of cloud computing by addressing various issues such as processing time [7, 8, 12, 16], cost [1, 2] and energy efficiency [3, 4]. In the list based workflow scheduling techniques various priority schemes have been proposed till now. Some of the schemes are b-level, t-level, HLF, CPOP, LPT, PETS and so on [10–12]. Based on above-mentioned priority schemes a number of workflow scheduling algorithms have been designed in cloud computing environment. However, task clustering based approach is also applicable for the cost efficient workflow scheduling but most of them are applicable in the homogeneous environment and they are very compatible for obtaining makespan but none of them have considered cost as the main objective [1]. In the clustered task category, there are more possibilities to reduce data transfer time but sometime cluster formation is not an easy job. Moreover, the intra-cluster data transfer time will be negligible but the data transfer time between inter cluster may be high due the multi-level dependencies among the task nodes in the workflow application.

BASIC TERMINOLOGIES AND PROBLEM FORMULATION

A Workflow application $W_f = (T, E)$ can be modeled by a directed acyclic graph (DAG), where $T = \{t_1, t_2, \dots, t_n\}$ is the task crew and E is the data transfer or precedence edges. Any two dependent task nodes $t_i, t_k \in T$ are scheduled in such manner that task t_k can not be scheduled until execution of t_i has been completed.

Before addressing the problem formulation, this section requires the detailed explanation of few important terminologies and formulas which are as follow.

Definition 1. The set of participated M cloud servers is represented as the CS = {CS₁,CS₂,CS₃,...,CS_M} which consists of m number of heterogeneous virtual machines (VMs). These VMs are the processing elements used for task execution and other operations on cloud servers. It is noteworthy that total number of created VMs may vary according to the deployment capacity of cloud servers. If any two VMs, VM_i and VM_j are deployed or created on same cloud server then the data transfer time between VM_i and VM_j will be considered zero otherwise the data transfer time between VM_i and VM_j will be considered.

Definition 2. (Estimated Computation Time): The estimated computation time ECT of a task t_i is the estimated execution time on a VM_j which is evaluated as dividing the task size (MI) by the processing speed of VM (MIPS).

$$ECT = \begin{matrix} & \overbrace{VM_1 \dots VM_{|CS_1|}}^{CS_1} & \dots & \overbrace{VM_{\alpha+1} \dots VM_{\alpha+|CS_M|}}^{CS_M} \\ \begin{matrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_n \end{matrix} & \begin{bmatrix} ECT_{1,1} & \dots & ECT_{1,|CS_1|} & \dots & ECT_{1,\alpha+1} & \dots & ECT_{1,\alpha+|CS_M|} \\ ECT_{2,1} & \dots & ECT_{2,|CS_1|} & \dots & ECT_{2,\alpha+1} & \dots & ECT_{2,\alpha+|CS_M|} \\ ECT_{3,1} & \dots & ECT_{3,|CS_1|} & \dots & ECT_{3,\alpha+1} & \dots & ECT_{3,\alpha+|CS_M|} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ ECT_{n,1} & \dots & ECT_{n,|CS_1|} & \dots & ECT_{n,\alpha+1} & \dots & ECT_{n,\alpha+|CS_M|} \end{bmatrix} \end{matrix} \quad (1)$$

In order to define problem definition, some other useful terminologies are explained as below.

The average estimated computation time (Avg_ECT): It is the average estimated computation time of any task t_i against all the m VMs which is described as follows.

$$Avg_ECT(t_i) = \sum_{j=1}^m \frac{ECT_{i,j}}{m}$$

EST(t_i,VM_j): It is the earliest start time at which the task t_i is ready for execution on available VM pool. The EST of entry task will be zero because there are no predecessor task node of entry task t_i. EST(t_i,VM_j) of any task t_i on any VM_j is defined as.

$$EST(t_i, VM_j) = \max \left\{ \max_{m'} \{Ayl[j]\}, \max_{pt_i \in predr(t_i)} \{AFT_{pt_i} + DTT_{pt_i}\} \right\}$$

where m' is virtual machine type on any cloud server

EFT(t_i, VM_j): Earliest finish time of task t_i on VM_j can be expressed by Eq. (4).

$$EFT(t_i, VM_j) = ECT_{i,j} + EST(t_i, VM_j)$$

where, $predr(t_i)$ is the set of just previous predecessor task nodes of task t_i and $Avl[j]$ is the minimum time at which VM_j is ready for task execution and $ECT_{i,j}$ is the estimated computation time of task t_i on VM_j .

Problem Definition

The mapping of the n task nodes of a given workflow on m active VMs in order to generate an optimal schedule plan with the following objectives.

Objective 1: The overall processing time of workflow application W_f i.e., makespan and schedule length ratio (SLR) are to be minimized.

Objective 2: The total cost (T cost) incurred by the cloud user for the given workflow W_f also to be minimized.

The afore-mentioned objectives are vividly described in the upcoming statements.

Makespan: It is the overall workflow processing time by the participated cloud servers on m active VMs. Let $t_i \rightarrow VM_j$ denote mapping of task t_i on VM_j at cloud server C_k . It is calculated as follows:

$$MKspan = \min\{EFT(t_{exit})\}$$

Schedule Length Ratio (SLR): It is the proportion between parallel execution time and the sum of computation time of those tasks lying on critical path running on fastest VM. The minimizing nature of SLR is attractive for any scheduling algorithm.

Cost: Now, to evaluate the cost we need the time for which the VMs are used for executing the workflow applications. Our cost model involves rate of all VMs to evaluate the total cost. The total cost incurred for scheduling applications can be calculated using Eq. 6. In cost matrix unit can be in minutes, hour and days etc. It is depended on the requirement of users.

$$T_{cost} = \sum_{i=1}^m Mkspn(VM_i) * rate(VM_j)$$

PROPOSED ALGORITHM

In this paper, we propose a cost conscious workflow scheduling algorithm which is based on the ADAP (As Delay As Possible) characteristic of the task's start time. The ADAP start time of a task node can be measured as the delayed start time which does not affect on the overall execution time (makespan) for the given workflow W_f . The proposed algorithm runs in two phases. In first phase, it assigns the priority of each task t_i of the given workflow W_f by using ADAP priority scheme. After-that, at each level (l_i) of the W_f the tasks are arranged as per their decreasing order of their respective priority. Then the second phase allocates the task on the suitable VM among the available VM pool by mapping the optimal possibilities of t_i . The mapping between task-VM is accommodated by calculating earliest start time (EST) and earliest finish time (EFT) for each task t_i . The above-mentioned steps are repeated until all the task nodes are scheduled of the workflow W_f .

ALGORITHM

Algorithm 1 :

Proposed Scheduling Algorithm

Input – $W_f(T,E)$ workflow consisting of n tasks nodes

Output - Optimal Schedule Plan Generation S of $W_f(T,E)$ by all m active VMs which are deployed on M cloud servers

```

Read the workflow  $W_f$  and find the requirements of tasks against
the all available VMs on different cloud server
Set the average estimated computation time  $Avg\_ECT(t_i)$  of every
tasks in ECT matrix
Compute ADAP ( $t_i$ ) for all tasks by traversing DAG in bottom up
fashion from the exit task node
for each level  $l_i$  of  $W_f(T,E)$  do
    Sort each tasks node in the Priority Queue ( $Q_g$ ) as per the
    descending order of ADAP( $t_i$ ) value.
    while the  $Q_g$  consists of unscheduled task do
        Choose the first task,  $t_i$  from the  $Q_g$ 
        for each VM from the deployed set of  $m$  at the cloud
        server set  $M$  and  $M_m \geq |M_k|$  do

```

```

        Calculate EST ( $t_i$ ,  $CS_k$ ) and EFT( $t_i$ ,  $CS_k$ ) by Eq.
        (3) and Eq. (4) by using insertion scheme
        Assign task  $t_i$  to the virtual machine  $VM_j$  at  $CS_k$ 
        that minimizes the EFT( $t_i$ ,  $CS_k$ ).
    end for
end while
end for

```

Algorithm 2 :

COMPUTE ADAP(t_i)

```

Build a tasks list in reverse topological order (Rev. Top List)
for each task node  $t_i$  in Rev.Top List do
    Min_Length = CPL
    for each child  $t_y$  of  $t_i$  do
        if ADAP( $t_i$ )-DTT( $t_i$ ,  $t_y$ ) < Min_Length
            then
                Min_Length = ADAP( $t_i$ ) - DTT( $t_i$ ,  $t_y$ )
            end if
        end for
    ADAP( $t_i$ ) = Min_Length - Avg_ECT( $t_i$ )
end for

```

Psuedo Code :-

```

class Graph:
    constructor for initialization
        Initialize the variables src, V, E, parent,
        createGraph

    function createAdjacencyList(self, edges):
        for v in self.V:
            self.E[v] ← list(map(lambda x:
            x[1], list(filter(lambda x: (x[0] = v), edges))))
            for u in self.E[v]:
                self.parent[u].append(v)

    function calculate_CPL_utils(self, u, d, visited, path,
    CPL, DTT, avg_ECT, pre←0, w←0):
        visited[u-1] ← True
        path.append(u)
        w += avg_ECT[u-1]
        if pre>0: w += DTT[pre-1][u-1]

```

```

        if u = d: CPL[-1] ← max(CPL[-1], w)
    else:
        for i in self.E[u]:
            if visited[i-1]=False:
                self.calculate_CPL_utils(i,          d,
                visited, path, CPL, DTT, avg_ECT, u,
                w)
    path.pop()
    visited[u-1] ← False

function calculate_CPL(self, DTT, avg_ECT):
    visited ← [False]*len(self.V)
    path ← []
    CPL ← [0]
    self.calculate_CPL_utils(self.src,  T[-1],  visited,
    path, CPL, DTT, avg_ECT)
    return CPL[-1]

function calculate_CPmin_utils(self, u, d, visited, path,
CPmin, DTT, ECT_t, pre←0, w←0, sl←0):
    visited[u-1] ← True
    path.append(u)
    w += min(ECT_t[u-1])
    sl += min(ECT_t[u-1])
    if pre>0: w += DTT[pre-1][u-1]
    if u = d and CPmin[0] < w:CPmin[0], CPmin[1] ← w, sl
    else:
        for i in self.E[u]:
            if visited[i-1]=False:
                self.calculate_CPmin_utils(i,          d,
                visited, path, CPmin, DTT, ECT_t, u,
                w, sl)
    path.pop()
    visited[u-1] ← False

function calculate_CPmin(self, DTT, ECT_t):
    visited ← [False]*len(self.V)
    path ← []
    CPmin ← [0, 0]
    self.calculate_CPmin_utils(self.src,          self.V[-1],
    visited, path, CPmin, DTT, ECT_t)
    return CPmin

function topological_sort_util(self, u, visited, stack):

```

```

        visited[u-1] ← True
        for v in self.E[u]:
            if not visited[v-1]:
                self.topological_sort_util(v, visited,
                    stack)
        stack.append(u)

function topological_sort(self):
    visited ← [False]*len(self.V)
    stack ← []
    for u in self.V:
        if not visited[u-1]:
            self.topological_sort_util(u, visited,
                stack)
    return stack

function printG(self):
    print the adjacency representation of the graph

# Distance Travel Time
function get_DTT(T, E):
    DTT ← [[0]*len(T) for t in T]
    for e in E:
        DTT[e[0]-1][e[1]-1] ← e[2]
    return DTT

# Computing ADAP
function compute_ADAP(g, DTT, avg_ECT):
    ADAP ← [0]*len(g.V)
    tplgl_order ← g.topological_sort()
    CPL ← g.calculate_CPL(DTT, avg_ECT)
    for u in tplgl_order:
        min_length ← CPL
        for v in g.E[u]:
            if ADAP[v-1] - DTT[u-1][v-1] < min_length:
                min_length ← ADAP[v-1]-DTT[u-1][v-1]
        ADAP[u-1] ← min_length-avg_ECT[u-1]
    return ADAP

# Generating the schedule for the tasks and Creating VM_Mapping
table
def scheduler(g, ADAP, DTT):

```

```

src ← [g.src]
next_src ← []
Avl ← [0]*VM
exec_order ← []

Mkspan ← [0]*VM
AFT ← [math.inf]*len(T)
VMID ← [0]*len(T)
CSID ← [0]*len(T)
VM_mapping ← [{"EST":[0]*len(T), "EFT":[0]*len(T)} for vm
in range(VM)]

while src:
    tasks ← []
    for u in src:
        if (-1*ADAP[u-1], u) not in tasks:
            heappush(tasks, (-1*ADAP[u-1], u))
        if u not in next_src: next_src += g.E[u]
    while tasks:
        t ← heappop(tasks)[1]
        exec_order.append(t)
        parent ← g.parent[t]
        for vm in range(VM):
            for p in parent:
                if VM_CS[vm+1]=CSID[p-1]:
                    VM_mapping[vm]["EST"][t-1]
                    ←max(VM_mapping[vm]["EST"][t-1],AFT[p-1])
                else:
                    VM_mapping[vm]["EST"][t-1]
                    ←max(VM_mapping[vm]["EST"][t-1],AFT[p-1]+DTT[p-1][t-1])
            VM_mapping[vm]["EST"][t-1] ← max(Avl[vm],
            VM_mapping[vm]["EST"][t-1])
            VM_mapping[vm]["EFT"][t-1] =
            VM_mapping[vm]["EST"][t-1]+ECT[vm][t-1]
            if AFT[t-1]>VM_mapping[vm]["EFT"][t-1]:
                AFT[t-1] ← VM_mapping[vm]["EFT"][t-1]
                VMID[t-1] ← vm+1
                CSID[t-1] ← VM_CS[vm+1]
            Avl[VMID[t-1]-1] ← VM_mapping[VMID[t-1]-1]
            ["EFT"][t-1]

```

```

        Mkspan[VMID[t-1]-1] += VM_mapping[VMID[t-1]-
        1]["EFT"][t-1] - VM_mapping[VMID[t-1]-
        1]["EST"][t-1]
    src ← next_src
    next_src ← []
print("\nOrder ", end="")
for vm in range(VM): print("EST EFT ", end="")
print("VMID AFT CSID")
for t in exec_order:
    print("%5d"%(t),end="")
    for vm in range(VM):
        print("%5d%5d"%(VM_mapping[vm]["EST"][t-1],
        VM_mapping[vm]["EFT"][t-1]), end="")
    print("%4d%6d%4d"%(VMID[t-1], AFT[t-1], CSID[t-1]))
return Mkspan, AFT, VMID, CSID, VM_mapping

def main():
    g ← Graph(T, E)
    DTT ← get_DTT(T, E)
    print("\nData Transfer Time between nodes")
    for dtt in DTT:
        print(dtt)
    #avg_ECT
    avg_ECT ← list(map(lambda x:sum(x)/len(x), zip(*ECT)))
    print("\nAverage Estimated Computation Time : ",avg_ECT)
    #ADAP
    ADAP ← compute_ADAP(g, DTT, avg_ECT)
    print("\nADAP : ",ADAP)

    Mkspan, AFT, VMID, CSID, VM_mapping = scheduler(g, ADAP,
    DTT)
    # SL
    SL ← g.calculate_CPmin(DTT, list(zip(*ECT)))[1]
    # Make Span
    MKS ← AFT[-1]
    # Cost Utilization
    Cost_VM ← [0]*VM
    for vm in range(VM):
        Cost_VM[vm] ← Mkspan[vm]*VMrate[vm]
    total_cost ← sum(Cost_VM)
    # Execution Time -> Energy Consumption
    Ebusy, Eidle ← .7, .2
    energy_utilization ← sum(map(lambda x:x*Ebusy + (SL-
    x)*Eidle, Mkspan))

```

```

# Load Balancing
AT ← [0]*VM
for vm in range(VM):
    AT[vm] ← sum(list(map(lambda x, y: x if y=vm+1 else
        0, ECT[vm], VMID)))
avg_AT ← sum(AT)/len(AT)
SD ← (sum(list(map(lambda x:(x-avg_AT)**2, AT)))/VM)**.5
#VM Utilization
utilization ← list(map(lambda x: x/MKS, AT))
avg_U ← sum(utilization)/VM
#Speed up
sum_ECT ← []
for vm in range(VM):
    sum_ECT.append(sum(ECT[vm]))
speed_up ← min(sum_ECT)/MKS
# SLR
SLR ← MKS/SL
print all output (SL, Mkspan, total_cost,
energy_utilization, Standard deviation of Active time,
average VM Utilization, Speed up, Schedule Length ratio)
if __name__ == '__main__':
    main()

```

Example

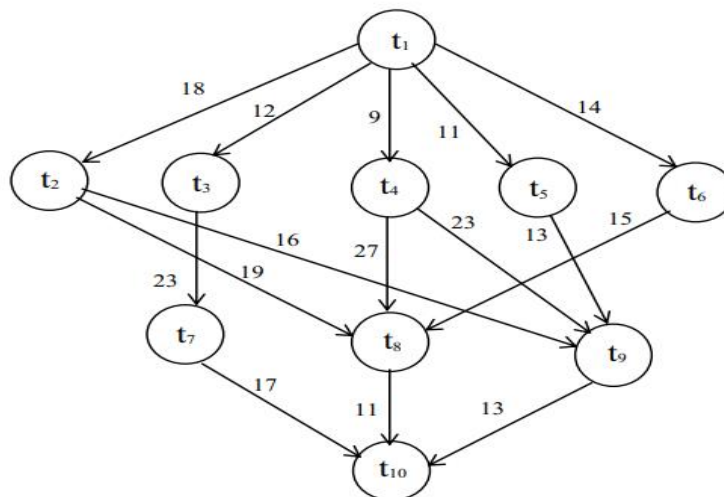


Fig. 1: An example of sample workflow with 10 task nodes

First, we find the average estimated computation time (Avg ECT) for all the tasks as per the Eq. 1 then compute the ADAP value of each task in the given DAG according to Algorithm 2. The Avg ECT for all the 10 tasks will be as [13, 16.66, 14.33, 12.66, 11.66, 12.66, 11, 10, 16.66 and 14.66]. All the 10 tasks [t1, t2, t3, t4, t5, t6, t7, t8, t9 and t10] have the corresponding ADAP values [1, 32, 29, 29, 40, 45.66, 66.33, 73.33, 64.66 and 94.33] respectively. Now at level 1 only t1 is presented and it is entry task so it will be scheduled first and rest of the task will be arranged as per the decreasing order of the ADAP at each level. For example at level 2 there are five tasks t2, t3, t4, t5 and t6. So at this level task t6 have the highest ADAP value so among these five tasks then task t6 will be scheduled by calculating EST and EFT as per the Eq. 3 and Eq. 4. We repeat this process at each level of DAG until whole workflow DAG is scheduled. Therefore, the complete task-VM mapping and scheduling are given in Fig. 2.

		t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
CS1	VM1	14	13	11	13	12	13	7	5	18	21
	VM2	16	19	13	8	13	16	15	11	12	7
CS2	VM3	9	18	19	17	10	9	11	14	20	16

Table III : ECT Matrix for given workflow sample

By applying Eq. 5, $Mkspn = \min\{EFT(textit)\}$ on the table of Fig. 2 the makespan of the given example is 61 time unit as per the proposed algorithm wheres the makespan of existing algorithms ALAP, PETS, HEFT and FCFS are 73, 70, 73 and 88 time unit respectively. As per the final schedule generation the makespan of V M1,VM2 and V M3 are 25, 40 and 28 time unit respectively. Moreover, as per the Eq. 6 the incurred cost is 126.91499999999999 unit.

Output :-

	CS1				CS2		VMID	AFT	CSID
	VM1		VM2		VM3				
	EST	EFT	EST	EFT	EST	EFT			
t1	0	14	0	16	0	9	3	9	2
t6	23	36	23	39	9	18	3	18	2
t5	20	32	20	33	18	28	3	28	2
t2	27	40	27	46	28	46	1	40	1
t3	40	51	21	34	28	47	2	34	1
t4	40	53	34	42	28	45	2	42	1

t8	42	47	42	53	69	83	1	47	1
t7	47	54	42	57	57	68	1	54	1
t9	54	72	42	54	65	85	2	54	1
t10	54	75	54	61	71	87	2	61	1

Overall task-VM mapping and scheduling of the given workflow

Schedule Length : 41

Mkspan : [25, 40, 28]

MKS : 61

Total Cost Utilization : 126.91499999999999

Total Energy Utilization : 71.1

Standard Deviation of Active time : 6.48074069840786

Average VM Utilization : 0.5081967213114754

Speed up : 2.081967213114754

Schedule Length Ratio : 1.4878048780487805

CONCLUSION

In this paper, we have presented a cost conscious workflow scheduling heuristic using the delay characteristic of the task node in a given workflow application. We have also introduced a new priority scheme for task ordering which is being accommodated on each level of DAG. The proposed heuristic kept an effective balance between time and cost incurred for VM renting when compared to existing state of art of algorithms. We have tested the scheduling algorithm on two scientific benchmark workflows (cybershake and epigenomic) and different sizes of random DAGs upto 1000 task nodes using arbitrary distribution of ETC matrix and data transfer time. The paper also evaluates the performance metrics (makespan, SLR and Cost) and results were compared with ALAP, HEFT, PETS and FCFS based cost effective scheduling technique considering same parameters. The simulation results pacify the effectiveness of the proposed approach over existing scheduling algorithms. However, the analysis was carried out considering the static behavior of workflows instead of dynamic condition.