Volume 3, Issue 7, July 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Cost-Aware Workflow Scheduling for Utility Grid Environment

Anita Choudhary
Department of Computer Engineering
NIT- Kurukshetra, India

Abstract—Over the last few years, Grid technologies have progressed towards a service-oriented paradigm that enables a new way of service provisioning based on utility computing models. It provides services to users on the basis of pay-to-access as information technology utilities. The main challenge in utility Grid is to satisfy the user Quality of Service requirement, as well as minimizing the execution cost of workflow. The number of existing bi-criteria scheduling algorithms are usually dedicated for following utility Grid environment: 1) fully connected network, and 2) fixed data transfer time and cost (independent from selected services), utility Grid models. These scheduling algorithms are not suitable for arbitrary service networks (type of utility Grid environment), where services are distributed arbitrary. In this work, we proposed the cost based workflow scheduling on arbitrary service network. In arbitrary service network the communication time and cost are depends on available path and bandwidth between services. The proposed workflow scheduling is work in three phases: 1) all-pair bandwidth analysis phase, in which finding the available path and bandwidth between services, 2) deadline distribution phase, in which the sub-deadlines are assign to every tasks, and 3). Planning phase, in which overall deadline is distributed over individual tasks. Two different deadline distribution approaches are implemented. And, the results are trace on random graph with different deadlines.

Keywords—Grid computing; Workflow scheduling; Quality of Service; utility Grid environment; Workflow management system

I. INTRODUCTION

With the rapid development of network technology, Grid Computing has emerged for satisfying the increasing demand of the computing power of scientific computing community. Utility Grid model [1, 2] provide services to users on the basis of pay-to-access as information technology utilities with the aim to satisfy the diverse Quality of Service (QoS) requirements of number of users. Mostly, user applications as in scientific and enterprise domains are constructed in form of workflow, in which precedence relations between tasks are defined. The services are defined as computational resources plus software that are needed to execute the workflow task. The users consume QoS [3] service required for execution of workflow and pay only for consumed services. Because of pay only strategy the Utility Computing system is provide the guarantee of services whereas the traditional/community grid provide the best-effort services. User can make advance reservation of services, which satisfy the user defined QoS requirements on the bases of Service Level Agreement (SLA). SLA is the contract between service provider and user, upon the violation of contract penalty is applied. The SLA is motivating the users for well defining their required services, with the concept of cost like higher cost is pay for more strict QoS. Sometimes it is happen that the user wants cheaper services in this case the cost of workflow is reduces. The workflow is represented by Directed Acyclic Graph (DAG), in which precedence relations between tasks are defined i.e. tasks are dependent on other tasks to start their execution. The workflow is define, manage and executed by the Workflow Management System (WfMS) and they also define workflow execution algorithms. There are number of grid workflow projects have been developed, like Askalon [4, 5], GrADS [6] and Pegasus [7] and WfMS taxonomy is define in[8]. In general, scheduling tasks on distributed resources belongs to a class of problems known as NP-hard problems [9, 10]. Many heuristics [11, 12, 13, 14] and meta-heuristics [15, 16, 17, 18] based algorithms have been developed to schedule task applications in: 1) fully connected service graphs, and 2) fixed data transfer time and cost (independent from selected services). In our work, we the proposed *Workflow Scheduling Algorithm based Upon PCP algorithm* schedule the workflow application on arbitrary service network. In arbitrary service network the communication time and cost are depends on bandwidth between services. The proposed workflow scheduling, is work in three phases: 1) all-pair bandwidth analysis phase, in which finding the available path and bandwidth between services, 2) deadline distribution phase, the overall deadline of the workflow is distributed across individual tasks. First, the algorithm tries to assign sub-deadlines to all tasks of the (overall) critical path of the workflow such that it can complete before the user's deadline and its execution cost is minimized. Then it finds the partial critical path to each assigned task on the critical path and executes the same procedure in a recursive manner, and 3). Planning phase, in which the cheapest services are assign to every task while meeting the sub-deadline of task.

II. SCHEDULING SYSTEM MODEL

The proposed scheduling system model consists of an application model and utility Grid model. An application is modelled by a directed acyclic graph G(T,E) where T is the set of n dependent task $\{t_1, t_2, ..., t_n\}$ and E is the set of arcs. The task graph edge is represented as $e_{i,j} = (t_i, t_j)$, where task t_j is depends on task t_i. *Workflow Scheduling Algorithm based Upon PCP algorithm* requires the single entry and single exit task of entire workflow so we add two dummy task t_{entry} and t_{exit} at the entry and exit point of workflow. The dummy task having both the computation (time/cost) and communication (time/cost, between dummy and workflow tasks) are zero.

A utility Grid (arbitrary service network) model consists of several Grid Service Providers (GSPs), each of which provides some services to the users. The utility Grid environment is modelled by undirected arbitrary service network graph G(S, E), where S is the set of heterogeneous services $\{S^1, S^2, ..., S^m\}$ and E is the set of edges between heterogeneous services without self loop and multiple edges. The undirected edge is represented as $e_{i,j} = (S^i, S^j)$; where $(S^i, S^j) = (S^j, S^i)$. Each workflow task t_i can be processed by mi services $(S_i^1, S_i^2, ..., S_i^{m_i})$ from different service providers with different QoS properties. There are many QoS attributes for services, like execution time, price, reliability, security, and so on.

The estimated execution time $ET(t_i, s)$ is computed through the size of task i.e. million of instructions (mi) and processor speed i.e. million of instruction per second (mips), and it is also computed by the code analysis, analytical benchmarking, code profiling, and statistical prediction [19]. The data transfer (communication) time $TT(e_{i,j})$ and cost $TC(e_{i,j})$ is depends on the bandwidth between selected services (on which task t_i and t_j is executed) and the amount of data to be transfer.

III. WORKFLOW SCHEDULING

Scheduling is the process of mapping of tasks of a workflow (application) on suitable services. And order the task on each service to satisfy the performance criteria (sub-deadline, satisfaction, importance).

A. Main Idea

Our key idea in this work is to propose the scheduling algorithm which works for the arbitrary service network. The arbitrary networks are not fully connected network. In this network the communication time and cost are depends on the bandwidth between services. If the task will schedule on another next slower service then the communication time with its parent and child tasks will changed (because bandwidth is vary between pair of services on which dependent tasks are scheduled) and it atomically affects the makespan and cost of workflow.

The proposed workflow scheduling is work in three phases:

- The all-pair bandwidth analysis phase finds the path between all-pair (source to destination) of services (on which direct dependent tasks are executed). After accessing the path, select the minimum bandwidth on path. Because the maximum data transfer on network in streaming form is depends on the minimum bandwidth, so we select the minimum bandwidth form maximum bandwidth (distance) path.
- The deadline distribution algorithm is based on a Critical Path (CP) and Partial critical path (PCP)[11] heuristic. The critical path of a workflow is the longest execution path between the entry and the exit tasks of the workflow.
- Finally, the planning algorithm [11] schedules the tasks according to their sub-deadlines. In the following sections, we elaborate on the details of the deadline distribution algorithm.

B. Basic Definitions

Some approximate notations for scheduling like the start time of task is define as Earliest Start Time (EST) (used in deadline distribution phase) and Actual Start Time (AST) (used in planning phase). The Execution Time $ET(t_i, s)$ of each task is varies from service to service, so computing the exact execution time of task is not possible. Same as calculating the exact communication time $TT(e_{i,j})$ (also known as data Transfer Time) is not possible because it is also depends on the selected services on which dependent tasks (t_i and t_j) are scheduled. The execution cost is depends on randomly define cost matrix (high speed services having higher cost) and on $ET(t_i, s)$ of task. So the Execution Time and Cost are defined as:

$$ET(t_i, s) = mi(t_i) / mips(s); \quad s \in S_i \tag{1}$$

$$EC(t_i, s) = ET(t_i, s) * CM(t_i, s); \tag{2}$$

The Transmission Cost $TC(e_{i,j})$ is depends on amount of data is transfer between dependent task and bandwidth between selected services. The $TT(e_{i,j})$ and $TC(e_{i,j})$ are defined as:

$$TT(e_{i,j}) = \begin{cases} dataTransfer(e_{i,j}) * \left\{ \dfrac{1}{minBandwidth_{r,s}} \mid r \in S_i, s \in S_j \right\}; & If \quad r \neq s \\ 0; \qquad otherwise \end{cases} \tag{3}$$

$$TC(e_{i,j}) = links * MTT(e_{i,j}) * cost\left(minBandwidth_{r,s} \mid r \in S_i, s \in S_j\right) \quad (4)$$

Minimum Execution Time MET (t_i) of task is define as:

$$MET(t_i) = \min_{s \in S_i} ET(t_i, s) \quad (5)$$

With the help of above definitions we define the EST as:

$$EST(t_{entry}) = 0$$

$$EST(t_i) = \max_{t_p \in t_i\text{'s parents}} EST(t_p) + MET(t_p) + TT(e_{p,i}) \quad (6)$$

Now we define the LFT based on the user defined deadline as:

$$LFT(t_{exit}) = D$$

$$LFT(t_i) = \min_{t_c \in t_i\text{'s children}} LFT(t_c) - MET(t_c) - TT(e_{i,c}) \quad (7)$$

C. Scheduling workflow Algorithm

Algorithm 1 shows the pseudo-code of proposed workflow scheduling. Two dummy tasks entry and exit tasks are added in line 3, to the workflow for easy implementation purpose. Line-4 call the *AllPair(G(S,E)) bandwidth analysis algorithm* for accessing the path and bandwidth between all the services. Lines 5-8 are compute the required data and in line-9 the deadline is assign to entry and exit tasks. Line-10 describe that initially workflow having the null schedule means no one task is schedule on any service. Because of entry and exit tasks having the deadline so we mention them as assigned. Line 12-14 describes that *AssignParents* algorithm is called for each parent of exit task. Line-15 calls the *planning algorithm* for Service Selection (SS) and service reservation. From line 16-22 compute the Total Execution Cost of tasks (TEC_V), Total data Transmission Cost (TTC_E) and finally, Total Execution Cost (TEC_W) of workflow.

Algorithm 1: Workflow Scheduling Algorithm based Upon PCP

1. ***procedure*** SCHEDULEWORKFLOW*(G(T,E),D)*
2. request available services for each task in G from GMD
3. add t_{entry}, t_{exit} and their corresponding edges to G
4. **call** *AllPair(G(S,E))*
5. compute $MET(t_i)$ for each task according to Eq. 5
6. compute $TT(e_{i,j})$ for each edge according to Eq. 3
7. compute $EST(t_i)$ for each task in G according to Eq. 6
8. compute $EST(t_i)$ for each task in G according to Eq. 7
9. $\text{sub-deadline}(t_{entry}) \leftarrow 0,\ \text{sub-deadline}(t_{exit}) \leftarrow D$
10. schedule←{null, null}
11. mark t_{entry} and t_{exit} as assigned
12. **while**(t_{exit} having an unassigned parents) do
13. call *AssignParents(t_{exit})*
14. **Endwhile**
15. **call** *Planning(G(T,E),D)*
16. $TEC_V = 0;\ TTC_E = 0;$
17. compute $EC(t_i, s \mid \text{schedule})$ for each task according to Eq. 2
18. compute $TC(e_{i,c}, r, s \mid r, s \in \text{schedule})$ for each edge according to Eq. 4
19. forall(task t_i :=1,…n) do
20. $TEC_V = TEC_V + EC(t_i, s \mid \text{schedule})$; // Total Execution Cost of task
21. Forall (task edges $e_{i,c}$:edge from task i to its child's c)
22. $TTC_E = TTC_E + TC(e_{i,c}, r, s \mid r, s \in \text{schedule})$; // Total data Transmission Cost
23. endfor
24. $TEC_W = TEC_V + TTC_E$; // Total Execution Cost of Workflow
25. end procedure

Time Complexity: The workflow graph G is the directed acyclic graph having n node and e edges. The maximum number of edges in the G is $(n-1)\times(n-2)/2$, we also define it as $e \in O(n^2)$. We assume that maximum number of available services for each task is m and l is the length of the longest path between entry and exit task. Now we compute the time complexity of above algorithm as:

Line 5 MET: $O(n.m)$, Line 6 MTT: $O(e.m)$, Line 7 EST: $O(n+e) = O(n^2)$, Line 8 LFT: $O(n+e) = O(n^2)$, Line 15 Planning: $O((n+e).m) = O(n^2.m)$, Line 20 TEC_V: $O(n)$, Line 22 TTC_E: $O(e)$.

D. All−Pair bandwidth analysis algorithm

Algorithm 2 shows the pseudo-code of *All-Pair bandwidth analysis Algorithm*, it is based on DIJKASTRA algorithm [20]. This algorithm is work for accessing the path between source to destination services (on which direct dependent tasks are executed). As a result, we find out the path, now select the minimum bandwidth on the path by comparing the path bandwidth. Because the maximum data transfer on network in streaming form is depends on the bandwidth between source to destination nodes, so we select the minimum bandwidth form maximum bandwidth (distance) path.

Algorithm 2: All-Pair bandwidth analysis Algorithm

1. ***Procedure*** *AllPair(G(S,E))*
2. **for all** i=1,2,…,m
3. Step 0:
4. Temporarily assign band(S^i) = 0; // band(S^i) means the total bandwidth on path to reach at node S^i
5. **for all** j={(1,2,…,m) ∩ i}
6. band(S^j) = −1; // band(S^j) means the current total bandwidth to getting node S^i
7. links(S^j) = 0; //number of links travel
8. Parent(S^j) = null
9. pathbandwidth($S^i \rightarrow S^j$) = −1; //minimum bandwidth on path
10. Step 1:
11. Find the node S^j with the highest temporary value of band(S^j)
12. **if** there are no temporary nodes or if band(S^j) = −1, **then** go to line 2
13. Node S^j is now labeled as permanent.
14. set links(S^j) = links(S^i) + 1
15. set Parent(S^j) = S^i
16. **if** pathbandwidth($S^i \rightarrow S^j$) > bandwidth($e_{i,j}$), **then** set pathbandwidth($S^i \rightarrow S^j$)←bandwidth($e_{i,j}$)
17. Node S^j is now labeled as the current node.
18. band(S^j) and parent of S^j will not change again
19. Step 2:
20. **for each** temporary node labeled vertex S^k adjacent to S^j , make the following comparision:
21. **if** band(S^j) + bandwidth($e_{j,k}$) > band(S^k), **then** band(S^k) ← band(S^j) + bandwidth($e_{j,k}$)
22. Assign S^k to have parent S^j
23. Step 3:
24. Return to step 1
26. **End for**
27. ***end procedure***

Time Complexity: The time complexity of this algorithm is $O(V^3)$.

E. Parents Assigning Algorithm

The Pseudo-code of *AssignParent* algorithm is define in algorithm 3. The first call of while loop (2-14 lines) find the critical path of workflow, assign the sub-deadline through *AssignPath(PCP)* procedure and the other call of while loop work for finding partial critical paths. In the line 8 the sub-deadline assigning procedure i.e. *AssignPath(PCP)* is called. The while loop 2-14 is called again and again and its calls is depends on the number of parents of exit task. The line 9-12 update the EST and LFT of unassigned successors and predecessors of currently assigned path and recursively call the *AssignParent* procedure i.e. work for partial critical path.

Algorithm 3: Assigning Deadline to the Parents Algorithm

1. ***procedure*** ASSIGNPARENTS(t)
2. **while** (t has an unassigned parent) **do**
3. $PCP \leftarrow null,\ t_i \leftarrow t$
4. **while** (there exists an unassigned parent of t_i) **do**
5. add $CriticalParent(t_i)$ to the beginning of PCP
6. $t_i \leftarrow CriticalParent(t_i)$
7. **end while**
8. **call** *AssignPath(PCP)*
9. **for all** $(t_i \in PCP)$ **do**
10. update EST for all unassigned successors of t_i
11. update LFT for all unassigned predecessors of t_i
12. **call** $AssignParents(t_i)$
13. **end for**
14. **end while**
15. ***end procedure***

- The *Critical Parent* of a node t_i is the unassigned parent of t_i that has the latest data arrival time at t_i, that is, it is the parent t_p of t_i for which $EST(t_p)+MET(t_p)+TT(e_{p,i})$ is maximal.[11]
- The *Partial Critical Path* of node t_i is
  - empty if t_i does not have any unassigned parents.
  - consists of the Critical Parent t_p of t_i and the Partial Critical Path of t_p if has any unassigned parents.

Time Complexity: Firstly compute the time complexity of partial critical path which is $O(l)$. And the time complexity of *AssignPath* is depends on the selected policy.

The time complexity of the above algorithm, on the basis of selected policy is $O(n^2 . l^2 . m)$ and $O(n^2 . l . m)$.

F. Path Assigning Algorithm

This algorithm assigns the sub-deadline to the input path nodes. Two assigningpath policies are describe 1. Decrease cost policy [11]. and 2. Fair policy[11].

1) Decrease Cost Policy

This policy is based on greedy approach and finds the good solutions in polynomial time. Initially all the tasks are assign to the fastest services (this is the more expensive schedule). And then we assign them to the next slower service with the help of Cost Decrease Ratio (CDR) The CDR value is commuted by following formula:

$$CDR(t_i)=\frac{TEC(t_i,cs)-TEC(t_i,ns)}{TET(t_i,ns)-TET(t_i,cs)} \tag{8}$$

Where cs indicates the current service for task t_i, ns indicates the next slower service for task t_i. The Total Execution Cost $TEC(t_i,s)$ is depends on the task execution cost, task communication cost with its parent and child task on path/assigned task. And the Total Execution Time $TET(t_i,s)$ is depends on the task execution time, task communication time with its parent and child task on path/assigned task.

The current service is replaceable with next slower service or the next slower service is admissible on task, if the following condition (eq. (9)) is satisfied:

Firstly find out the t_i's direct dependent, parent (t_p) and child (t_c) tasks, for which the $TT(e_{i,j})$ is maximum through the following formula:

$$\max\left(\max_{t_p\in t_i\text{'s parents},r\in S_{p_2},s_{ns}\in S_i} TT(e_{p,i},r,s_{ns})+\max_{t_c\in t_i\text{'s childs},s_{ns}\in S_i,r\in S_c} TT(e_{i,c},s_{ns},r)\right)$$

the above parent (tp) and child (tc) tasks are used in replaceable condition. And the condition is:

$$\left(TT(e_{p,i},r,s_{ns}\mid r\in S_p,s_{ns}\in S_i)+EST(t_i)\ +ET(t_i,s_{ns})\ +TT(e_{i,c},s_{ns},r\mid s_{ns}\in S_i,r\in S_c)\right)\le \left(TT(e_{p,i},r,s_{cs}\mid r\in S_p,s_{cs}\in S_i)+LFT(t_i)+TT(e_{i,c},s_{cs},r\mid s_{cs}\in S_i,r\in S_c)\right); \tag{9}$$

After calculating the CDR of each task on path, t* hold the single task from path nodes, which is having the maximum CDR (must be $0<CDR(t_i)$) and also satisfied the eq.(10). The Pseudo-code of this policy define in algorithm 4.

Algorithm 4: Decrease Cost Path Assigning Algorithm

1. ***procedure*** *AssignPath(PCP)*
2. cur $\leftarrow$ assign the fastest service to each task of the path
3. compute $CDR(t_i)$ for each task of the path according to Eq. (8)
4. **repeat**
5. $t^* \leftarrow null$
6. **for all** $(t_i \in path)$ **do**
7. **If** ($CDR(t_i) > CDR(t^*)$ and t_i is admissible according to Eq. (9)) **then**
8. $t^* \leftarrow t_i$
9. **end if**
10. **end for**
11. **If** (t^* is not null) **then**
12. update cur by assigning t^* to the next slower service
13. update LFT of all Parent tasks $\in$ path and EST of child tasks $\in$ path w.r.t. t^*
14. update $CDR(t^*)$
15. **end if**
16. **until** (t^* is null)

17. **If** (there is an inadmissible assignment in cur) **then**
18. set sub-deadline(t)=EST(t)+MET(t) for all tasks t on the path
19. **else**
20. set sub-deadline according to cur for all tasks ∈ path
21. **end if**
22. mark all nodes of the path as assigned
23. ***end procedure***

Time Complexity: The time complexity of above algorithm is $O(l^2 . m)$.

2) Fair policy

This policy tries to distribute the sub-deadline on all the nodes of path in fair manner, by recursively checking the next slower service on the task without exceeding the LFT of task. This process is repeated again and again until no more replacements are processed. The pseudo-code of this policy is define in algorithm 5.

Algorithm 5: Fair Path Assigning Algorithm

1. ***procedure*** *AssignPath(PCP)*
2. cur ← assign the fastest service to each task of the path
3. **repeat**
4. **for all** $(t_i \in path)$ **do**
5. **If** (assigning t_i to the next service is admissible according to Eq. (9)) **then**
6. update ***cur*** by assigning t_i to the next slower service
7. update LFT of all Parent tasks on path and EST of child tasks on path w.r.t. t_i
8. **end if**
9. **end for**
10. **until** (no change is done)
11. **If** (there is an inadmissible assignment in cur) **then**
12. set sub-deadline(t)=EST(t)+MET(t) for all tasks t on the path
13. **else**
14. set sub-deadline according to cur for all tasks ∈ path
15. **end if**
16. mark all nodes of the path as assigned
17. ***end procedure***

Time Complexity: The time complexity of above algorithm is $O(l . m)$.

G. The Planning Algorithm

This algorithm tries to select the best services for each task and gives the optimized schedule having the minimum cost. It is make the local decisions on each step when selecting the services. At each ready state it selects the cheapest services that execute the task before its sub-deadline. So the selected service for a ready task t_i, $SS(t_i)$, is the service $s \in S_i$ for which

$$EC(t_i, s) + \sum_{t_p \in t_i\text{'s parents}} TT(e_{p,i}, SS(t_p), s)$$

is minimized subject to the condition that

$$AST(t_i, s) + ET(t_i, s) \leq \text{sub-deadline}(t_i) \quad (10)$$

where the Actual Start Time of t_i on s, $AST(t_i, s)$, is the maximum between the latest data arrival time of the parents of ti to the services, and the start time of the suitable free time slot on the service s.

It is possible that no service can execute t_i before its sub-deadline, because the sub-deadlines are calculated by using idealized EST and estimated schedules (in the path assigning phase) which do not consider the actual free time slots on the services. Therefore, it is possible that all suitable services are busy at that moment. In that case, we just select the service with the minimum finish time, i.e., $SS(t_i)$ is the service $s \in S_i$ for which

$$AST(t_i, s) + ET(t_i, s) \quad (11)$$

is minimized. Of course, this delay must be compensated in the following selections, as soon as possible. The pseudo-code of Planning algorithm is shown in Algorithm 6.

Algorithm 6: Planning Algorithm

1. ***procedure*** PLANNING(G(T, E))
2. $Queue \leftarrow t_{entry}$
3. **while** (Queue is not empty) **do**
4. t ← delete first task from Queue
5. query available time slots for each service from related GSPs

6. compute SS(t) according to Eq. (10) and (11)
7. $AST(t) \leftarrow$ the actual start time of t on SS(t)
8. make advance reservation of t on SS(t)
9. schedule $\leftarrow \{t, SS(t)\}$
10. **for all** ($t_c \in$ children of t) **do**
11. **If** (all parents of t_c are scheduled) **then**
12. add t_c to Queue
13. **end if**
14. **end for**
15. **end while**
16. ***end procedure***

IV. PERFORMANCE EVALUATION

A. *Experimental setup*

The utility grid environment is simulated by multi-cluster grid, in which a single cluster having the 20-100 nodes with same processor. The clusters are differ by speed and cost. All the cluster having the random number of nodes and all the clusters are connected by arbitrary network. We assume that all the clusters having the required software, so task is executed on any cluster. The random cost per second is assigned to each cluster on the basis of fastest cluster cost is more then the slower cluster cost. In the intra-cluster environment all the nodes are connected with each other. The average inter-cluster bandwidth is between 1to 5Mbps, and its cost is proportional to the bandwidth. The average intra-cluster bandwidth is 1Gbps and its access is free. We assume that all are services and bandwidth between services is available free, so they are available at any desired time and each task is executed on any of the service.

B. *Experimental Results*

The *Workflow Scheduling Algorithm based Upon PCP algorithm* is work better for small size of workflow (10 to 100 tasks). As the size is increase, the number of task is waiting for allocation on service, so the performance of algorithm is affected.

To show how the *Workflow Scheduling Algorithm based Upon PCP algorithm* work, we use the random generated task graph G(T, E), shown in Figure (1), with 9 nodes and two dummy entry and exit nodes. The execution of algorithm is show with both of the *AssignPath(PCP)* policies.

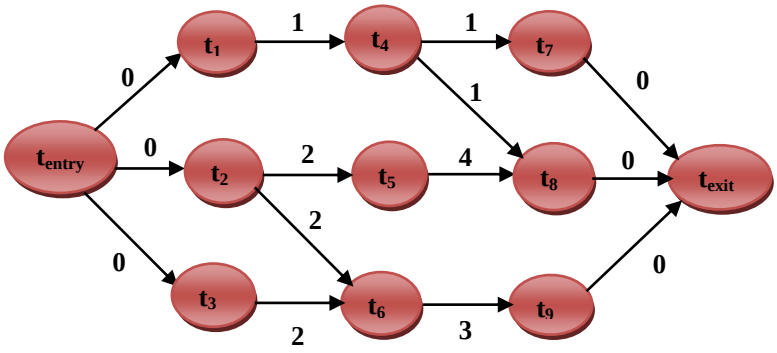

Fig. 1 Workflow task graph

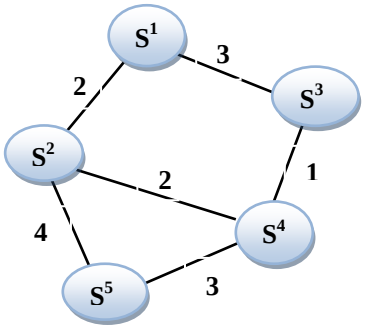

Fig. 2 Service graph

The Fig. 2 shows the arbitrary network service graph G(S,E),with 5 service clusters and undirected edges between clusters.

- The tasks mi (million of instruction) is shown in Table 1:

TABLE I TASKS MI

Task id →	t_{entry}	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{exit}
Task *mi*→	0	110	170	85	200	150	225	160	190	65	0

- The processor speed mips (million of instruction per second), shown in Table 2:

TABLE II PROCESSORS MIPS

Processor id →	S^1	S^2	S^3	S^4	S^5
Processor *mi*→	9.5	8.0	6.5	5.0	4.0

- The random cost matrix ($CM(t_i, s)$) per unit of time, shown in Table 3:

TABLE III COST MATRIX PER UNIT OF TIME

	S^1	S^2	S^3	S^4	S^5
t_1	30.0	23.0	17.0	11.0	7.0

t_2	32.5	25.0	19.0	13.0	7.0
t_3	35.0	28.0	21.0	14.5	9.0
t_4	33.5	27.0	20.5	14.0	9.0
t_5	31.5	24.5	18.0	13.0	8.0
t_6	34.0	27.0	20.0	13.0	9.0
t_7	32.0	24.0	17.5	13.5	9.0
t_8	31.0	25.0	18.0	12.0	6.5
t_9	32.0	24.0	18.0	13.0	7.0

1) Implementation result based on Decrease Cost Policy

TABLE IV EXECTUION VALUES LIKE EST, LFT AND DL FOR FIG. 3 WITH USER DEFINE DEADLINE 75

Task	**EST**	**LFT**	**Step1**			**Step1.1**			**Step2**			**Step3**		
id			**EST**	**LFT**	**DL**	**EST**	**LFT**	**DL**	**EST**	**LFT**	**DL**	**EST**	**LFT**	**DL**
t_1	0	33.59	0	**24.38***		0.0	**24.38***	**22.0***	0.0	24.38	22.0	0.0	24.38	22.0
t_2	0	39.21	0.0	**25.02***	**21.25***	0.0	25.02	21.25	0.0	25.02	21.25	0.0	25.02	21.25
t_3	0	44.47	0	44.47		0	44.47		0	44.47		0.0	**30.38***	**21.25***
t_4	11.58	55.0	11.58	**45.44***		**22.5***	**45.44***	**43.55***	22.5	45.44	43.55	22.5	45.44	43.55
t_5	17.89	55.0	**22.25***	**43.77***	**41.0***	55.0	22.25	43.77	55.0	22.25	43.77	55.0	22.25	43.77
t_6	17.89	68.16	**22.25***	68.16		17.89	68.16		17.89	68.16		**22.25***	**65.0***	**56.87***
t_7	32.63	75.0	32.63	75.0		**43.55***	75.0		**43.55***	**75.0***	**68.17***	43.55	75.0	68.17
t_8	33.68	75.0	**43.0***	**75.0***	**72.23***	43.0	75.0	72.23	43.0	75.0	72.23	43.0	75.0	72.23
t_9	41.58	75.0	**45.93***	75.0		45.93	75.0		45.93	75.0		**57.87***	**75.0***	**67.86***

The *Workflow Scheduling Algorithm based Upon PCP algorithm* working steps are trace in the Table 4, in which the Decrease cost policy algorithm is called. Let us assume the user-define deadline is 75. Firstly assign the deadline to the entry and exit task of workflow which is 0 and 75. And assign the initial values to the EST and LFT of each task. Now we call the assignparent algorithm on exit task.

Step1: When the assignparent algorithm is called on exit task it gives the critical path, which is tasks t_2-t_5 -t_8. Now the assignpath algorithm is call which updates the values of EST, LFT and sub-deadlines (DL). And schedule these tasks on following services:

Output of Step1: t_2-S^2, t_5-S^2, t_8-S^3

After assigning the DL values it mark the task as assigned task. Update the EST values of successor tasks (t_6 and t_9 task) of path. And LFT values of predecessor tasks (t_1 and t_4) of path. In the future no values are modified of assigned task.

Step1.1:Now it is calling the *AssignParent* algorithm on path task one by one, in which when it is call for t_2 tasks then it find the partial critical path which is t_1. For partial critical path t_1 it again calls the *AssignPath* algorithm in step 1.1 and updates the LFT and DL values of itself. And also update the EST and LFT values of its successor (t_7) and predecessor tasks (no predecessor).

Output of Step1.1: t_1-S^4, t_4-S^1

Step2: the above same procedure (Step1) is applied. The partial critical path is t_7**.**

Output of Step2: t_7-S^3

Step2.1: when again calling the *AssignParent* on path, no task is access.

Step3: the same (step 1) procedure is applied.

Output of Step3: t_3-S^5 t_6-S^3 t_9-S^3

2) Implementation result based on Fair Policy

Execution schedule with user define deadline 75 is:

Output of Step1: t_2-S^3, t_5-S^3, t_8-S^2

Output of Step1.1: t_1-S^3, t_4-S^3

Output of Step2: t_7-S^3

Output of Step2.1: --------

Output of Step3: t_3-S^5 t_6-S^3 t_9-S^3

3) Comparison matrix based on AssignPath policies

Table 5 shows the cost and makespan value with respect to different-deferent deadline inputted on random generated workflow graph, shown in Fig. 1 and corresponding service graph Fig. 2.

TABLE V COMPARISON MATRIX FOR CDR AND FP POLICES

Sr.	Deadline	CDR		FP	
No.		Cost	Makespan	Cost	Makespan
1	**50**	4567	53.68	4567	53.68
2	**55**	4473	53.68	4473	53.68
3	**60**	4300	58.72	4271	60
4	**65**	4124	66.75	4136	66.75
5	**70**	4083	69.65	4083	69.65
6	**75**	3981	72.23	3935	75.98
7	**80**	3784	78.75	3800	81.46
8	**85**	3697	86.32	3701	86.32
9	**90**	3693	87.23	3669	87.36
10	**95**	3596	94.5	3560	95.80
11	**100**	3407	100.5	3459	100.33
12	**105**	3384	103	3346	105
13	**110**	3202	108.25	3216	108.5
14	**115**	3197	114.08	3130	113.75
15	**120**	2966	116	2872	120
16	**125**	2919	122.33	2872	120
17	**130**	2720	128.5	2720	129.5
18	**135**	2720	128.5	2720	129.5

For the Fig. 1 workflow, the Fair Path (FP) policy performs better with varying deadline values.

V. CONCLUSION AND FUTURE DIRECTIONS

The utility Grid provides the guarantee of services with user interaction, based on the service level agreement. It defines the QoS service features in utility Grid environment, in which user access the services with deadline constraint and optimizes the workflow cost. For tight deadline the workflow cost is high and for loose deadline the cost is low. The user pay for services on the bases of required time of execution and on the type of service they are access that means fastest services having higher cost.

In this paper, we proposed a *Workflow Scheduling Algorithm based Upon PCP algorithm* cost-based workflow scheduling algorithm that minimizes the cost of execution while meeting the deadline. The detail working of *Workflow Scheduling Algorithm based Upon PCP algorithm* on arbitrary service network is explained, in which the communication time and cost is highly dependent on the selected services. When the tasks are scheduled on next slower services its communication time and cost with its successors and predecessors task are changed based on the selected services, as if all are scheduled on same service type, then the time and cost both are zero, otherwise having some values based on the bandwidth and on the number of links between the selected service.

We evaluate the algorithm on random generated graph and show the execution results with two deadline distribution policies 1. Decrease cost policy and 2. Fair path policy. The complexity of Fair policy is low and it performs better then the Decrease cost policy. Future direction: In the future we will work on arbitrary service network for improving the communication cost. The cost is depends on the number of links between the selected service for data transfer on scheduled task.

REFERENCES

[1] J. Broberg, S. Venugopal, and R. Buyya, "Market-oriented Grids and Utility Computing: The State-of-the-Art and Future Directions," *J. Grid Computing*, vol. 6, no. 3, pp. 255-276, 2008.

[2] G. Thickins, "Utility Computing: The Next New IT Model", *DarwinMagazine*, April 2003.

[3] D. Laforenza, "European Strategies Towards Next Generation Grids," *Proc. Fifth Int'l Symp. Parallel and Distributed Computing (ISPDC '06)*, p. 11, 2006.

[4] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of Scientific Workflows in the Askalon Grid Environment," *SIGMOD Record*, vol. 34, pp. 56-62, 2005.

[5] T. Fahringer et al., "ASKALON: a tool set for cluster and Grid computing", *Concurrency and Computation: Practice and Experience*, Wiley InterScience, vol. 17, pp. 143-169, 2005.

[6] K. Cooper et al., "New Grid Scheduling and Rescheduling Methods in the GrADS Project", In *NSF Next Generation Software Workshop, International Parallel and Distributed Processing Symposium*, Santa Fe, IEEE CS Press, Los Alamitos, CA, USA, April 2004.

[7] E. Deelman et al., "Pegasus: A Framework for Mapping Complex Scientific Workflows Onto Distributed Systems," *Science Programming*, vol. 13, pp. 219-237, 2005.

[8] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *J. Grid Computing*, vol. 3, pp. 171-200, 2005.

[9] J. Ullman, *NP-complete Scheduling Problems, Journal of Computer and System Sciences*. vol. 10, pp. 384-393, 1975.

[10] D. Fernandez-Baca, "Allocating modules to processors in a distributed system", In *IEEE Transaction on Software Engineering*, pp. 1427-1436, 1989.

[11] S. Abrishami, M. Naghibzadeh, and Dick H.J. Epema, "Cost-Driven Scheduling of Grid Workflows Using Partial Critical Paths", *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, august 2012.

[12] J. Yu, R. Buyya, and C.K. Tham, "Cost-Based Scheduling of Scientific Workflow Applications on Utility Grids," *Proc. First Int'l Conf. e-Science and Grid Computing*, pp. 140-147, July 2005.

[13] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M.D. Dikaiakos, "Scheduling Workflows with Budget Constraints," *Integrated Research in GRID Computing*, CoreGRID Series, S. Gorlatch, and M. Danelutto, eds., pp. 189-202, Springer, 2007.

[14] Y. Yuan, X. Li, Q. Wang, and X. Zhu, "Deadline Division-Based Heuristic for Cost Optimization in Workflow Scheduling," *Information Sciences*, vol. 179, no. 15, pp. 2562-2575, 2009.

[15] J. Yu and R. Buyya, "Scheduling Scientific Workflow Applications with Deadline and Budget Constraints Using Genetic Algorithms," *Scientific Programming*, vol. 14, nos. 3/4, pp. 217-230, 2006.

[16] J. Yu, M. Kirley, and R. Buyya, "Multi-Objective Planning for Workflow Execution on Grids," *Proc. IEEE/ACM Eighth Int'l Conf. Grid Computing*, pp. 10-17, 2007.

[17] A.K.M.K.A. Talukder, M. Kirley, and R. Buyya, "Multiobjective Differential Evolution for Scheduling Workflow Applications on Global Grids," *Concurrency and Computation: Practice & Experience*, vol. 21, pp. 1742-1756, 2009.

[18] W.N. Chen and J. Zhang, "An Ant Colony Optimization Approach to Grid Workflow Scheduling Problem with Various QoS Requirements," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 39, no. 1, pp. 29-43, Jan. 2009.

[19] S. Verboven, P. Hellinckx, F. Arickx, and J. Broeckhove, "Runtime Prediction Based Grid Scheduling of Parameter Sweep Jobs," Proc. Asia-Pacific Conf. Services Computing, pp. 33-38, 2008.

[20] E. Hazzard, Available: http://vasir.net/blog/game_development/dijkstras_algorithm_shortest_path/, 2010.

