SindhUniv. Res. Jour. (Sci. Ser.) Vol. 50 (002) 295-302 (2018)

http://doi.org/10.26692/sujo/2018.06.0050

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Optimized Scheduling for Parallel Computing Environment

A. AHMAD[++], M. M. YOUSAF, S. SARWAR, W-UL-QOUNAIN, L. ASLAM, M. KHALID

Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore

Received 13th December 2017 and Revised 9th February 2018

Abstract: The increasingly parallel, heterogeneous, and powerful computing infrastructure demands efficient scheduling of tasks over computing nodes to fully utilize these computing resources. For an optimal scheduling strategy, it is important to consider all the parameters that have an impact on the overall execution time of a problem having tasks that may run in parallel.This work proposes an optimal scheduling algorithm for a class of problems that can be represented by a directed acyclic graph. The proposed algorithm is static in its nature and considers heterogeneity of processing elements. A comparison with existing algorithms under different scenarios also shows improvement in overall execution time.

Keywords: Directed acyclic graph (DAG; Heterogeneous computing; Scheduling communication

I. INTRODUCTION

The availability of increasingly powerful machines justifies the continuous progress of hardware vendors but at the same time raises significant challenges for software engineers. Now it is highly important to make full utilization of the computing powers of state-of-the-art machines by designing innovative scheduling techniques. Generally, the most expensive resource of a system is its processing units. Scheduling takes care of the maximum utilization of these expensive resources. Scheduling of computing resources has become more complicated because current machines have many such computing resources and these resources are quite diverse in their nature.

It should be highly important to consider the computational needs of parallel tasks, communication patterns and data transfers among different tasks, communication to computation ratio of overall parallel algorithm, and load balancing issues during deployment of parallel jobs on computational resources of a hybrid computing infrastructure. Scheduling strategies vary greatly with respect to the nature of problem. This work explores scheduling of applications that may be represented by a directed acyclic graph (DAG) and propose an optimal algorithm for such problems.

After discussing background of this domain and related work in section II and III respectively, the design of proposed algorithm is presented in section IV. Further, it has been evaluated and compared with existing algorithms under varying scenarios in section V. Finally, the discussion is concluded in section VI.

2. BACKGROUND

We assume that a parallel computing environment is a set of heterogeneous processing elements that are fully connected. Any processor can execute a task and communicate with other processors at the same because of overlapping computation and communication time. Once a processor starts executing a task, it continues without interruption and after completion, it immediately sends the output data to all its dependents. An application or problem to be solved is represented by a weighted, directed, acyclic graph (DAG). Each node of DAG represents a subtask that is executed independently on a processor. Each weighted edge represents the amount of data to be transferred.

Following sub sections discuss some common calculations (Topcuoglu, *et al.*, 2002) that are usually performed by static parallel scheduling algorithms.

A. Communication Cost

Communication cost represents the time required to transfer data from one processor to another over network. Following is a simplified model for communication cost to transfer m units of data from processor i to processor j:

$$c_{i,j} = t_s + mt_{i,j} \quad (1)$$

Where t_s denotes the startup time and $t_{i,j}$ is network speed from processor i to processor j. The startup communication cost for all the processors is usually assumed to be the same.

[++]Corresponding author: {mscsf08m003, murtaza, s.sarwar, swjaffry, laeeq.aslam, madiha.khalid}@pucit.edu.pk

B. Average Earliest Start Time (AEST)

Average earliest start time of a task v_i is computed as:

$$AEST(v_i) = \max_{v_m \in pred(v_i)} \left(AEST(v_m) + \overline{w_m} + \overline{c_{m,\iota}}\right) \quad (2)$$

Where $pred(v_i)$ is the set of immediate predecessors of v_i. $AEST(v_{start})$is 0. $w_{m,j}$is execution time of v_i on jth processor and $\overline{w_m}$ is average execution time of $v_m$which is calculated as:

$$\overline{w_m} = \frac{1}{p}\sum_{j=1}^{p} w_{m,j} \quad (3)$$

Finally, $\overline{c_{m,\iota}}$ is average communication cost.

C. Average Latest Start Time (ALST)

Average latest start time of a task v_i is computed as:

$$ALST(v_i) = \min_{v_m \in succ(v_i)} \left(ALST(v_m) - \overline{c_{m,\iota}}\right) - \overline{w_m} \quad (4)$$

Calculation of this time is started from the bottom of the graph.

D. Earliest Start Time (EST)

Earliest start time of a task v_i on processor p_j is computed as:

$$EST(v_i, p_j) = \max\left(avail(p_j), \max(AFT(v_t + c_{t,i}))\right) \quad (5)$$

Where $avail(p_j)$ is the time when processor $p_j$will be available after the execution of parent task(s) of v_i . AFT is the actual finish time for $v_t \in pred(v_i)$.$EST(v_{start})$is 0.

E. Earliest Finish Time (EFT)

Earliest finish time of a task v_i on processor p_j is computed as:

$$EFT(v_i, p_j) = w_{i,j} + EST(v_i, p_j) \quad (6)$$

F. Critical Path (CP)

Critical path (CP) of a DAG is longest path from the entry node to the exit node in the graph. The length of this path $|CP|$ is the sum of the computation costs of the nodes and inter-node communication along the path.

G. Algorithm Pattern for Static Scheduling

Most of the static scheduling algorithms follow a pattern and according to this patter an algorithm is divided into following three phases.

1) Level Sorting Phase

Level sorting is the starting phase of an algorithm, as this will determine the relation between a task and its sub tasks. Tasks on two different levels with a communication like between them can never be executed in parallel. A process can only be executed when all of its predecessor tasks have been executed and data required for this process has been transferred to it.

Entry node is the only node having no parent. All the DAGs have one entry node and one exit node. The exit node is the ending node and having no child. If some DAG has more than one entry and exit nodes then a new node with zero execution time and zero communication cost is created and connected with them.

2) Task Prioritization Phase

This phase is most critical and heart of most of the algorithms as it determines the execution flow and priority to be scheduled on processor. In this phase, using different heuristics, a sub optimal order is determined. So, this phase actually prioritizes tasks based on different calculations and assumptions. The commonly used properties to assign priority are average earliest start time, average earliest finish time, critical path, up link cost, and down link cost.

3) Processor Assignment Phase

In this phase sub tasks are assigned to processors based on the availability of processor. So, this phase gets a list of prioritized sub tasks and schedules them on processors.

3. RELATED WORK

In heterogeneous scheduling environment computing nodes are not necessarily of the same specs. The systems may have different hardware and operating system running on them. In heterogeneous environment the execution time of each sub process on each node will be different and will affect the final execution time of process.

Scheduling can be categorized as decentralized and hierarchical. Decentralized scheduling model does not require central leader responsible for scheduling. Resources owners may apply some restrictions and schedulers will have to follow them. Development of scheduling algorithm becomes difficult if resource owners and scheduler do not agree with respect to resource management. In case of hierarchical scheduling a resource owner can locally apply some policies for external users. So, a scheduler can be designed considering different layers of resource hierarchy (Zhang, 2002)

Adaptive scheduling is suitable for dynamic environments where real time status of the resources is used. It evaluates the needs of tasks on the fly and tries to fulfill it in order to achieve desired goals. It requires an active resource manager that keeps track of the available resources. The available resources can be assigned dynamically (Huedo, 2004). In this case, scheduling strategy and parameters can be replaced at runtime.

Network aware scheduling considers the network realities in a greater depth because data transfer cost plays a vital role in scheduling tasks at distributed locations (Yousaf, *et al.,* 2014). It intelligently uses the information of shared bottlenecks (Yousaf, *et al.,* 2014) for realistic estimation of communication time.

In static scheduling the execution order and processor assignment is determined before the start of execution of process. Flow of sub processes is determined before the actual execution. Usually, the calculations are based on the averages. It can be used when resource requirements of all the tasks is known in advance. Further, the knowledge of underlying infrastructure is also assumed to be known in advance. Obviously, it is not appropriate to situations where dynamic changes are significant and frequent. Some well-known static scheduling strategies are discussed in the following sub sections.

H. Heterogeneous Earliest Finish Time (HEFT)

The level sorting phase of HEFT works in top down fashion (Topcuoglu, *et al.,* 2002). In this phase the given DAG is to sort tasks at each level to group the tasks that are independent of each other. As a result, tasks in the same level can be executed in parallel. For assigning priority to a task, three attributes are used which are average computation cost, data transfer cost, and the rank of predecessor tasks. Priority is assigned to all the tasks at each level based on its rank. Tasks are ranked as upward and downward. At each level the task with highest rank value receives the highest priority. In the processor selection phase, the processor, which gives minimum EFT for a task is selected and the task is assigned to that processor.

The upward rank of a taskv_i, which is actuallylength of the longest path ofv_i to the exit node, is recursively defined as:

$$Rank_u(v_i) = \overline{w_i} + \max_{v_j \in succ(v_i)} \left(\overline{c_{i,j}} + Rank_u(v_j)\right) \quad (7)$$

HEFT uses EFT to select the processor for each task.

I. Critical Path On a Processor (CPOP)

Level sorting phase of CPOP (Topcuoglu, *et al.,* 2002) works in similar way as it is done in HEFT. In CPOP the priority is calculated using$Rank_u$and $Rank_d$ where $Rank_u$ is computing using equation (7) and $Rank_d$ is calculated as:

$$Rank_d(v_i) = \max_{v_j \in pred(v_i)} \left(\overline{w_j} + \overline{c_{j,i}} + Rank_d(v_j)\right) \quad (8)$$

The priority of a task is calculated as $Rank_u(v_i) + Rank_d(v_i)$. The processor selection phase has two options:

1. If the current node is on the critical path it is assigned to the critical path processor
2. Otherwise it is assigned to the processor that minimize the execution completion time.

J. High Performance Task Scheduling (HPS)

In HPS, all the tasks are sorted according to their level in DAG. The entry task is at the top level and all other tasks in levels follow the following role:

"Level i consist of all tasks v_k such that, for all edges(v_j, v_k), task v_j is in a level less than i and there exists at least one edge(v_j, v_k) such that v_j is in level $i-1$."

The priority of tasks is calculated using down link cost, up link cost, and link cost. For each task, from highest to lowest priority, the processor, which gives minimum EFT is selected.

K. Level and Branch Priority (LBP) (Ilavarasan, et al., 2005)

This algorithm used the same level sorting technique as used in HEFT. The attributes that are used to calculate the priority of tasks are T–Level: the length of the longest path from the entrance node to the task nodev_i and B–Level: the length of the longest path from the task node$v_i$to the exit node. Scheduling of a task v_i is done on the host that optimizes its earliest finish time.

4. DESIGN

This section discusses the design of optimized and flexible scheduler for parallel computing environment that schedules the sub tasks on processing elements to minimize the total execution time of a parallel problem. This algorithm is based on the communication with the children of any node along with uplink cost. The proposed algorithm depicted in **(Fig. 1)**, is divided into three phases and each of the phases is discussed in the following sub sections.

1:	*Read the DAG, associated attributes, and infrastructure details for all tasks v_i at each level*
2:	***begin***
3:	*compute $\overline{w_i}$ using equation (3)*
4:	*compute $Rank_u(v_i)$ using equation (7)*
5:	*compute $DVC(v_i)$ using equation (10)*
6:	*prioritize the task based on $Rank_u(v_i) +$*
7:	*$DVC(v_i)$*
8:	*for each processor p_k in the processor set*
9:	***begin***
10:	*compute $EFT(v_i, p_k)$ using equation insertion*
11:	*(6)*
12:	*$p_k \leftarrow v_i$, which minimize the $EFT(v_i, p_k)$*
13:	***end***
	end

Fig. 1. Proposed algorithm for scheduling.

L. Level Sorting Phase

In first step the DAG is divided into levels and then the rank of each node is calculated using uplink cost along with down vertices cost. Entry node is at level 1 and there should always be one entry point. If there are more entries for a DAG (as it could be) then a new node will be defined with zero execution time and it will be connected to all of the entry points with zero communication cost. Every node *i* connected to node *j* at level *n* will be at level *n+1* if there is a communication link from node *i* to node *j*. The exit node is at the highest level of graph and there should be only one exist node. More than one exit nodes will be connected to a single empty node.

M. Task Prioritization Phase

Priority of each node, in terms of its rank, is defined level wise. The rank of a node $v_i$is sum of its upward rank, as mentioned in equation (7), and down vertices cost (DVC), which is represented as:

$$Rank(v_i) = Rank_u(v_i) + DVC(v_i) \qquad (9)$$

DVC is the amount of collective data to be transferred from a node $v_i$to its child nodes.If v_i has $n$child nodes and $e_{i,j}$ represents the amount of data to be transferred from v_i to its jth child node, then:

$$DVC(v_i) = \sum_{j=1}^{n} e_{i,j} \qquad (10)$$

After calculating the rank of each of node, prioritization is started. The priority of each node is calculated based on its level and rank. Nodes at each level are sorted according to their rank and then priority is assigned. And then processor selection is done based on the priority.

N. Processor Selection Phase

In processor selection phase the prioritized list is traversed in top down order and each node is assigned to an available processor that minimizes the finish time of respective node. The availability of the processor is also maintained. A sub process can start its processing only if all of its predecessors have finished their execution and the required data has been transferred.

5. Evaluation and Comparison

We have evaluated our algorithm on three different scenarios. Further, its performance has been compared with the performances of HEFT and CPOP on these scenarios.

O. Scenario 1

First scenario consists of eleven sub tasks and three processors are available to execute this problem. The execution time of each sub task on each of processor is different and is provided in **(Table I).** The execution time of each sub tasks is tentative but not actual.

The communication between two subtasks is the amount of data transferred from one node to other. The communication cost between any two processors is assumed to be the same so we will ignore the data transfer rate and will be only considering the amount of data to be transferred. The complete DAG is represented in **(Fig. 2).**

Table: 1. Execution Time of Tasks for Scenario 1

Tasks	Execution Time		
	P_1	P_2	P_3
1	4	4	4
2	5	5	5
3	4	6	4
4	3	3	3
5	3	5	3
6	3	7	2
7	5	8	5
8	2	4	5
9	5	6	7
10	3	7	5
11	5	6	7

For task prioritization, first average execution time for each of the sub task is calculated by summing up the execution time on the processors and then divided by number of processors. By adding upward rank and down vertices cost, rank of each node is computed. Then the final prioritized list is computed level wise. For each level, all the tasks are sorted with respect to their ranking. This process is conducted level wise. All the computed results are presented in **(Table 2).**

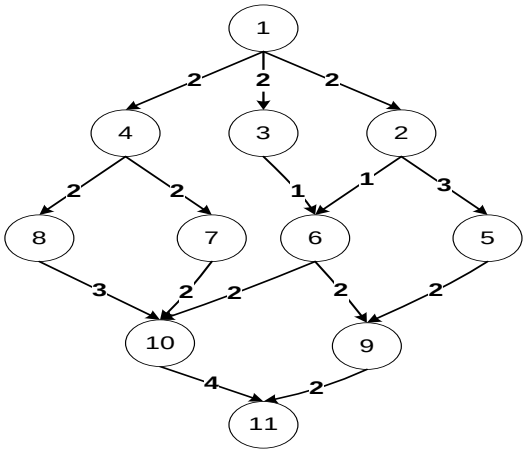

Fig.2. Directed acyclic graph of scenario 1.

For process allocation, earliest finish time of all the tasks is computed using equation (6). After that, prioritized list is traversed and tasks are allocated to suitable processors.

Execution flow of proposed algorithm is computed and compared with the execution flow of HEFT and CPOP. It has been observed that final execution time of proposed algorithm for scenario 1 is better than HEFT and CPOP by 10% and 25% respectively. Execution flows for all three algorithms are depicted in **(Fig. 3)**.

Table: 2. Prioritization Comutation for Scenario 1

Tasks	$\overline{w_i}$	$Rank_u(v_i$	$DVC(v_i)$	$Rank(v_i)$	Level	Priority
1	4	34	6	40	1	1
2	5	28	4	32	2	2
3	4.67	27	1	28	2	4
4	3	28	4	32	2	3
5	3.67	20	2	22	3	5
6	4	21	4	25	3	6
7	6	23	2	25	3	8
8	3.76	22	3	25	3	7
9	6	14	2	16	4	10
10	5	15	4	19	4	9
11	6	6	0	6	5	11

P. Scenario 2

This scenario consists of 10 tasks that have relatively large amount of data to be transferred and its tasks are computational intensive in comparison to scenario 1. All the tasks are spread over four levels. Tasks at each level require data from the tasks of previous level. In this way, tasks at a level are dependent on the complete execution of tasks at their previous level.

This scenario has been also discussed in the literature (Topcuoglu, *et al.*, 2002) (Huedo, 2004) and it also assumes three processing elements. The scenario is depicted in **(Fig. 3)** and tentative execution times of all the tasks on three processing units are presented in **(Table 3)**.

Table 3. Execution Time of Tasks for Scenario 2

Tasks	Execution Time		
	P_1	P_2	P_3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	7
9	18	12	20
10	21	7	16

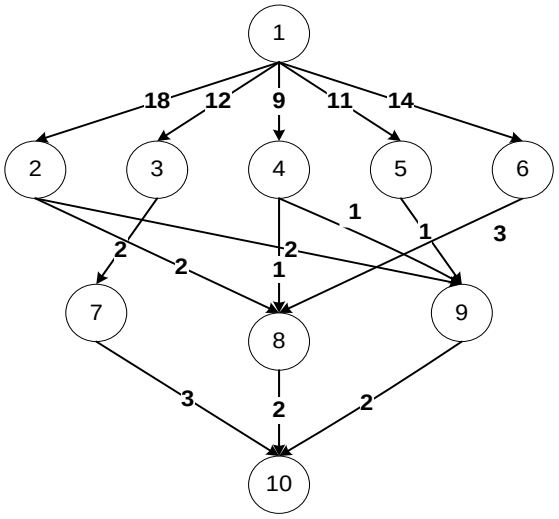

Fig.3.Directed acyclic graph of scenario 2.

The prioritization of tasks for scenario 2 is calculated and presented in **(Table 4)** Final execution time of proposed algorithm for this scenario is found to be better than HEFT and CPOP by 3.75% and 12.5% respectively. Execution flows for all three algorithms are depicted in **(Fig. 4)**.

Table 4. Prioritization Comutation For Scenario 2

Tasks	$\overline{w_i}$	$Rank_u(v_i)$	$DVC(v_i)$	$Rank(v_i)$	Level	Priority
1	13	108	64	172	1	1
2	16.7	77	35	112	2	3
3	14.3	80	23	103	2	4
4	12.7	80	50	130	2	2
5	11.7	69	13	82	2	5
6	12.7	63	15	78	2	6
7	11	43	17	60	3	7
8	10	26	11	37	3	9
9	16.7	14	2	16	4	10
10	14.7	15	4	19	4	9

Scenario 3

This scenario consists of 13 tasks with greater set of dependencies among the tasks. It also assumes three processing elements. The scenario is depicted in **(Fig. 4)** and tentative execution times of all the tasks on three processors are presented in **(Table 5).**

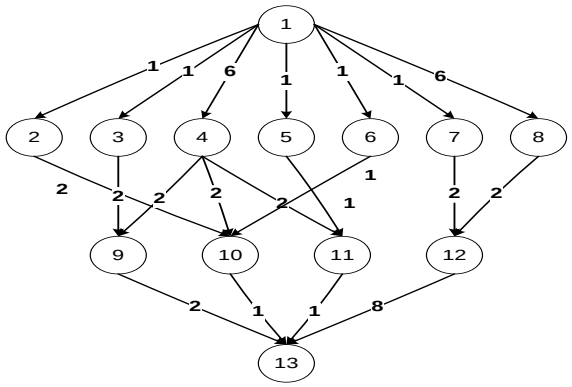

Fig.4. Directed Acyclic Graph Of Scenario 3.

Table 5. Execution Time of Tasks for Scenario 3

Tasks	Execution Time		
	P_1	P_2	P_3
1	22	15	17
2	15	25	5
3	22	5	9
4	19	19	16
5	16	20	24
6	16	14	9
7	22	5	9
8	19	19	16
9	15	12	21
10	9	21	3
11	5	6	25
12	15	12	21
13	10	17	24

The priority of tasks for scenario 3 is computed and presented in **(Table 6).** Final execution time of proposed algorithm for this scenario is found to be better than HEFT by 6.5%. Execution flows in this scenario for the proposed algorithm and HEFT are depicted in **(Fig. 5-7)**.

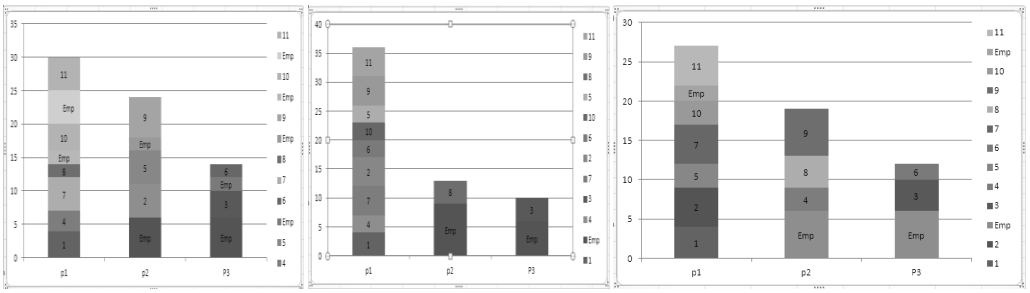

Fig. 5. Execution flow of HEFT, CPOP, and proposed algorithm for Scenario 1.

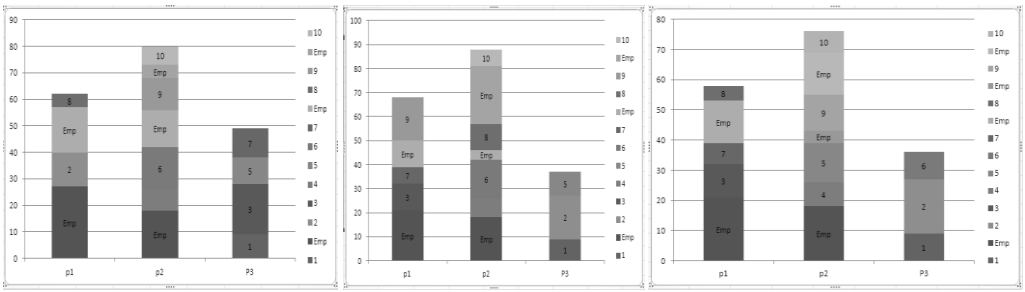

Fig. 6. Execution flow of HEFT, CPOP, and proposed algorithm for Scenario 2.

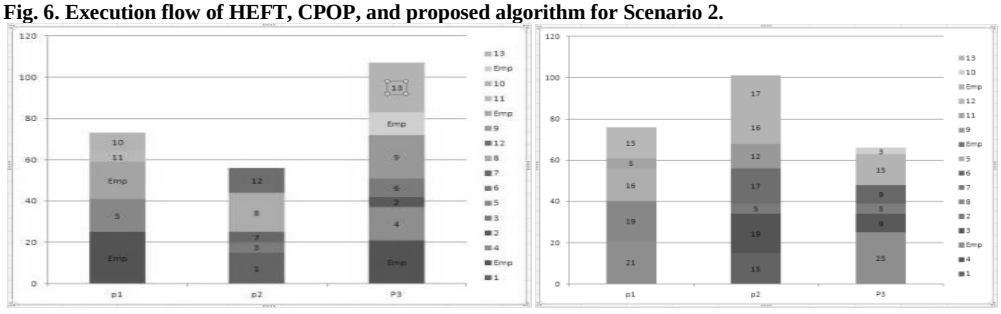

Fig. 7. Execution flow of HEFT, and proposed algorithm for Scenario 3.

Table 6. Prioritization comutation for scenario 3

Tasks	$\overline{w_i}$	$Rank_u(v_i)$	$DVC(v_i)$	$Rank(v_i)$	Level	Priority
1	18	127	74	201	1	1
2	15	75	22	97	2	5
3	12	95	22	117	2	3
4	18	103	72	175	2	2
5	20	73	10	83	2	7
6	13	64	13	77	2	8
7	12	75	22	97	2	4
8	18	73	14	87	2	6
9	16	61	28	89	3	9
10	11	38	10	48	3	12
11	12	43	14	57	3	10
12	16	41	8	49	3	11
13	17	17	0	17	4	13

5. CONCLUSION

This work proposed a scheduling strategy for problems that can be represented by a directed acyclic graph. The algorithm prioritizes all the tasks of a problem based on their ranks and suggests a mapping on computing elements. This mapping reduces the overall execution time of the problem. The algorithm is compared with existing strategies under different scenarios and shows its improvement up to 12.5%.

REFERENCES:

Case-Based Reasoning, (1993) Morgan Kaufmann Publishers,

Dong, F. and S. G. Akl, (2006) "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems", Technical Report No: 2006/504.

Huedo, E. (2004) "Experiences on adaptive grid scheduling of parameter sweep applications", 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing,Coruna, Spain.

Ilavarasan, E., P. Thambidurai, and R. Mahilmannan, (2005) "High Performance Task Scheduling Algorithm for Heterogeneous Computing System", International Conference on Algorithms and Architectures for Parallel Processing, ICA 03-10

Ma, D and W. Zhang, (2003) "A Static Task Scheduling Algorithm in Grid Computing",. International Conference on Grid and Cooperative Computing, GCC.

Topcuoglu, H., S. Hariri, and M. Y. Wu, (2002) "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, 260–274.

Yousaf, M. M. and M. Welzl, (2014) "Network-Aware HEFT Scheduling for Grid", The Scientific World Journal, DOI: 10.1155/2014/317284.

Yousaf, M. M and M. Welzl, (2013) "On the Accurate Identification of Network Paths Having a Common Bottleneck", The Scientific World Journal, DOI: 10.1155/2013/890578.

Zhang, L. (2002) "Scheduling algorithm for real-time applications in grid environment", IEEE International Conference on Systems, Man and Cybernetics, Yasmine Hammamet, Tunisia, Tunisia.