VOL. 14, NO. 12, JUNE 2019 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

PD-SECT: A NOVEL DAG ALGORITHM FOR SCHEDULING PARALLEL APPLICATIONS IN DISTRIBUTED COMPUTING ENVIRONMENT

P. Muthulakshmi and E. Aarthi
Department of Computer Science, SRM Institute of Science and Technology, Chennai, Tamil Nadu, India
E-Mail: sam.m.lakshmi@gmail.com

ABSTRACT

Applications to be solved in parallel fashion mostly use distributed environments. Generally, these kinds of applications are large computational projects of high complexity. The distributed environment is a massive pool of heterogeneous resources which could be utilized by the applications. Grid computing environment is one such environment that aggregates networks, computers, servers, applications, programs and the users. The co-ordination of resources is the mandatory aspect of distributed computing environment and this is achieved by efficient scheduling. An effective scheduling is very significant as it encounters high performances. A high quality scheduling is meant for its low cost, earlier completion, accurate results; and obviously that might be the key expectation of the clients too. These can be met only when resources are properly utilized by the applications. Parallel applications are illustrated mathematically as Directed Acyclic Graphs (DAGs), which help us to understand the dependencies and data mobility in the real applications. In this article, we present a scheduling algorithm that motivates quick and quality schedules, which in turn encourages reduction of makespan time, increases speed up and best rate of result recurrences. The algorithm called PD-SECT (Priority on Dependency and Start, Execution, Communication Time) is based on list heuristics. The algorithm accomplishes the stages of (i) task selection, (ii) resource selection, (iii) mapping the chosen task and resource. The priority in selecting the tasks and resources is based on the following criteria, (i) task selection is based on population of inter dependent tasks; (ii) resource selection is based on resource availability with respect to the start and execution time. The proposed algorithm does not encourage insertion policy as the tasks are encouraged to pack consecutively without idle slots. On comparisons with algorithms of its kind, this algorithm arrived at best results when scaled from smaller to bigger task graphs. The algorithms are implemented in GridSim simulator, which gives a feel of real time environment.

Keywords: list heuristics, scheduling, priority, makespan, speedup, scalable.

1. INTRODUCTION

Computational applications with greater complexities are always expected to be solved in a distributed environment of unlimited resources. The computing power of multiple resources is aggregated to solve bigger scientific problems in earlier time. Generally, commercial applications will always choose a computing environment that quotes low cost, fast finish, accurate services. Grid Computing is such a kind of virtual super computing environment that offers the services to its clients. Due to the heterogeneity both in terms of hardware and software, it is hard to co-ordinate resources and the consumers. The success of the computing environment lies in the effective utilization of resources to complete the problems at the earliest possible time. Scheduling is the key component which is responsible for mapping the applications and resources. Also, scheduling is focused intensively as its yields performance through proper management of resources. In order to solve the problem in a quick fashion, it is divided into smaller indivisible units called tasks which are executed on resources. Scheduling is nothing but the decisions that maps tasks and resources. Workflows are structured using task dependencies [1]; in which the communication cost and computation cost are the imperative elements that decide the mapping between tasks and resources. The optimization path takes a smooth run only when the communication cost and the computation cost are chosen to be the minimum. Packages of scheduling algorithms have been presented in [2] [3].

Generally, scheduling is divided into static and dynamic. In static /deterministic model, the information related to the problem is familiar before the scheduling is undertaken. Scheduling algorithms should aim at finishing the problem at the earliest; and also delivering highly accurate services. Generally, the scheduling algorithms endorse the performance of a computing system. The task scheduling algorithm is to allocate resources to the tasks and to establish an order for the tasks to be executed by the resources [4].

In general, Directed Acyclic Graph (DAG) is used to illustrate a workflow. In DAG, the preliminary constituents are the set of nodes and the set of edges, where the nodes represent the tasks and the edges represent the communication links. This task system contains a set of tasks with a precedence relation determined only by data dependencies [5]. Tasks are the contestants that compete to execute on resources. A task will be bound to the resources until it finishes its execution. The precedence constraint defined as the process of execution of a dependent/child task would start its execution only if all of its predecessors/parents have completed their execution. This is because the child might need data from its parent(s) to start its execution [6]. In this paper, we propose an algorithm that reduces the makespan and encourages the load balance across resources. The proposed algorithm uses list scheduling technique. The proposed algorithm adapts the dynamic programming technique [7]. The computation costs and communication costs are taken for consideration between

nodes/tasks related to three levels (i.e., parent, child/children, grand-child/grand-children) for comparison to arrive at selecting a task for execution. Also, the dependency of task is exploited for the betterment of schedules.

The continuation of the paper is compartmentalized as follows; Section 2 discusses the related works. In section 3, we describe the research problem. Section 4 is presented with the proposed algorithm. In section 5, the results and discussion is presented. Section 6 discusses the time complexity of the algorithm. Section 8 gives the conclusion of the paper and discusses the ideas that may enhance the present work.

2. RELATED WORKS

A survey of scheduling strategies and algorithms had been presented in [8] as an induction of this research progress. Static task-scheduling algorithms are classified with one of its major branches headed by heuristic based algorithms which are further classified into (i) List scheduling algorithms, (ii) Clustering algorithms and (iii) Task duplication algorithms. List scheduling is based on (i) task priority (ii) resource selection.

Scheduling of any of the above kind is called as DAG scheduling which aims (i) to reduce the schedule length (makespan), (ii) to improve the scheduling efficiency by minimizing communication delay, (iii) to effectively balance the load among the resources. Task scheduling problem is an application that could be shown through a DAG. DAG can be expressed as G= (T, E),
where, 'G' is the directed acyclic graph,

'T' represents the set of tasks and a task is identified as t_i, $1 \leq t_i \leq nt$ (maximum number of tasks)

'E' denotes the set of edges (inter task dependencies) between tasks; an edge can be shown as e_i, $1 \leq e_i \leq ne$ (maximum number of edges)

An edge is the communication link between the tasks exist two different levels. Hierarchy of tasks in the work flow is shown in terms of levels. Tasks in level 'i' can have its child/children in any of the levels greater than its level. No edge can connect tasks belonging to same level. The tasks found between the first level and last-1 level will have child/children. The first level will have tasks of no parents and the last level will have tasks of no child/children.

The COMPCost of t_i, $1 \leq t_i \leq nt$ would be the cost that would be taking to execute itself on a resource. Each t_i must be executed in the same resource until it gets executed each e_i, $1 \leq e_i \leq ne$ is represented by the COMMCost, the cost taken to transfer data between resources (if parent and child are executed in different resources). The COMPCost would become zero when the parent and child tasks are executed in the same resource. The child task would be waiting to start its execution until all the necessary data from its parent list are available.

Heterogeneous Earliest Finish Time (HEFT) [9] is a dependency mode algorithm which aims at reducing the makespan of tasks in DAG. HEFT has two major phases; (i) task prioritizing phase for computing priorities of all tasks and (ii) resource selection phase for selecting the tasks in the order of these priorities based on ranking. Each selected task is scheduled on a best resource that could minimize the finish time. HEFT algorithm uses an insertion-based policy, i.e., a task can be inserted in an earliest idle time-slot between two scheduled tasks in a resource.

Critical Path on Processor (CPOP) [9] algorithm selects the task on the critical path of the DAG and chooses the resource that minimizes the finish time of the task.

Efficient Dual Objective Scheduling (EDOS) [10] algorithm aims at the planning of advanced reservation of resources for entire workflow. The reservation of resources is done at early binding and mapping of resources to a particular task is done at late binding.

Time and Cost Improvement Algorithm (TCI) [11] is a list algorithm with greedy approach. This algorithm is devised by considering both the makespan and cost spent on utilizing the resources.

Mandal *et al* [12] used advance static scheduling to ensure that the important computational steps are executed on the proper resources and there of minimized a large set of data transportation.

3. RESEARCH PROBLEM AND DESCRIPTION

In this section, we define and describe the scheduling problem. The proposed scheduling algorithm uses the bind factor for finalizing the task a resource to be mapped. The bind factor constitutes the computation costs and communication costs of tasks in three different (not essentially contiguous) levels that have the relationship of parent, child, and grand-child on the execution path.

The task having more dependencies in the child level and the grand-child level will be given the priority for executing in the resources that minimizes the finish time (FT). The other parallel tasks will be assigned to other available resources based on the earliest finish time. The child having the maximum computation cost would be assigned to the same resource where its parent was executed. The results from other parent tasks which were executed in other resources would be made available to execute the grand-child.

The proposed algorithm starts by calculating Expected COMPutation Cost (ECOMPCost) on resource for each task with respect to speed of each resource in the list of available resources.

$$For\ i = 1\ to\ maxtasks$$
$$For\ j = 1\ to\ maxresources$$
$$ECOMPCost(t_i, p_j) = COMPCost(t_i)/speed(p_j) \quad (1)$$

Then the Average COMPutation Cost (ACOMPCost) is found for each task with respect to number of resources.

$For\ i = 1\ to\ maxtasks$
$ACOMPCost(t_i) = \sum_{j=0}^{Maxresources} COMPCOST(t_i, p_j)/ maxresources$ (2)

Here, the average data transfer rates between resources and average communication start up time are treated as 1.

The communication cost matrix of *n x n* (n represents number of tasks) size is given the values of communication cost with nodes having direct edges between source and destination. The Direct Communication Cost Matrix of (DCM) size n x n (n represents number of tasks / nodes) is stored with the communication cost of (t_i, t_j), where t_i and t_j are connected by an edge. Therefore, the step length is always one. The Intermediate Communication Cost Matrix (ICCM) of size n x n is stored with the communication cost(s) of (t_i, t_j) where t_i and t_j are connected by intermediate tasks. Here, the step length is number of intermediate nodes/tasks plus one.

For the tasks at the first level, the Quick Start Time is Zero.

$QST(t_i, p_j) = 0;\ 1 \le t_i \le nt\ and\ t_i \in level\ 0, 1 \le p_j \le maxresources$ (3)

For the rest, Quick Start Time (QST) and Quick Finish Time (QFT) are recusively calculated as follows:

$RRT(p_j) = FT(t_p(t_c), p_j))$ (4)

$For\ i = 1\ to\ parents(t_c)$
$RTR(t_p(i), p_j) =$
$max\{ReUT(t_p(i)) + COMMCost(t_p(i), t_c)\}$ (5)

$QST(t_c, p_j) = max(PRT(p_j), RTR(t_p(i), p_j)$ (6)

$QFT(t_C, p_j) = ACOMPCost(t_i) + QST(t_c, p_j)$ (7)

Where,
RRT represents Resource Ready Time
FT means Finish Time
RTR symbolizes Reach Time on Resource
$t_p(t_c)$ signifies Parent task of child task
ReUT refers to Resource Utilization Time

The first task in the first level is executed on the resource that could complete the execution in Minimum COMPutation Cost (MCOMPCost). The following equation finalizes the resource (Resource Finalization Factor (RFF)) for the chosen task,

$RFF(t_{i=0}) = Processor(min\{ECOMPCost(t_i)\})$ (8)

For more tasks in the first level, the execution of each task begins with the mapping of tasks of order t_1 to t_n (tasks in first level) to the ascending order of COMPCost(t_i) on available resources, i.e., t_1 is executed on resource p_j with MCOMPCost, t_2 is executed on resource with next MCOMPCost (to that of resource p_j), and so on.

For tasks found in second level onwards, the Resources Finalization Factor (RFF) is evaluated as:

$RFF(t_c(i) =$
$\{COMMCost\left(t_p, t_c(i)\right) +$
$min\{ECOMPCost(t_c(i))\}\};\ \ t_c(i) \in children(t_p)$ (9)

The child having maximum RFF will be executed in the same resource where the parent task had been executed and the child having the minimum RFF will be executed in the resource(other than the one which had been given to the child having max(RFF)), where the COMPCost is comparatively less. Recursively, the next children of either ends (max end and min end) would be choosing their resources to execute. In Parallel to the above execution, these tasks will be identifying the child (ren) in the next level (i.e., the grand-children (t_{gc}) of t_p (parents of t_c) with respect to t_c). Search the parent's t_{gc} and identify the prominent parent of all the t_{gc} whose RFF is comparatively greater than other with respect to other parents. Then execute the t_{gc} in the same resource by moving the data from other parents. The other conservative successors (grandchildren) are executed with respect to their MCOMP Cost with respect to the RFF.

The schedule length (make span /overall completion time of the DAG) is defined to be the resource's time that completes its work at the last of all the available resources and it must have executed the task in the last level (it must have executed at least one of the tasks in the last level of DAG in case, when there are multiple node in the last level).

4. THE PROPOSED ALGORITHM: Priority on Dependency and Start, Execution, Communication Time (PD-SECT)

1. Algorithm PD-SECT(DAG (tasks, weights, edges, edgecost), maxresources)
2. {
3. Compute ECOMPCost(t_i, p_j) of all the tasks
4. Compute ACOMPCost(t_i) for all tasks
5. Display CostMatrix
6. Compute QST
7. Compute QFT
8. Execute tasks of first level in resources of min(QFT)
9. Compute RFF
10. SortAscendingOrder (RFF)

11. While(unassigned tasks exist)
12. {
13. if(max(RFF(children))) then
14. Assign in same resource where parent had completed its execution
15. if(min(RFF(children))then
16. Assign in an another resource where the COMPCost(t_c(i)) is lesser

17. Compute RFF of grandchildren
18. if(prominent parent)// the task of (max(RFF)) then
19. Assign the task grandchild(ren) where prominent parent was executed and continue execution by obtaining data from other parents
20. else
21. Assign the task grandchild(ren) to resource that could execute in resource
    having MCOMPCost with respect to RFF
22. }//while ends here
23. Schedule-Extent=max(finish time of all the resources)
24. }//Algorithm ends here

5. RESULTS AND DISCUSSIONS

As a part of our research work, we developed a tool to generate random directed acyclic graphs [13]. The outcome of the tool could be the pictorial view of the graph with COMMCost and COMPCost, the text based information of the generated graph, the databases (tables) to store the data and samples are shown in Figures 1, 2, 3.

The inputs to be given to the tool are (i) the total number of tasks (thereof the height factor would be approximated with the maximum number of tasks in each level). The tool has been devised as it could alter the DAG's height factor based on user data of maximum tasks per level. And not all the levels would be equally distributed, but randomly distributed taking the seed as maximum tasks per level. (ii) Number of resources and their respective speed, further it would be asking the choice of generating a DAG with either one in the starting and ending level(level '1' and level 'n') or irrespective of the either. (iii) Maximum COMMCost and Maximum COMPCost which would be taken as the seed value to generate the random COMMCost and COMPCost of individual edge and task respectively.

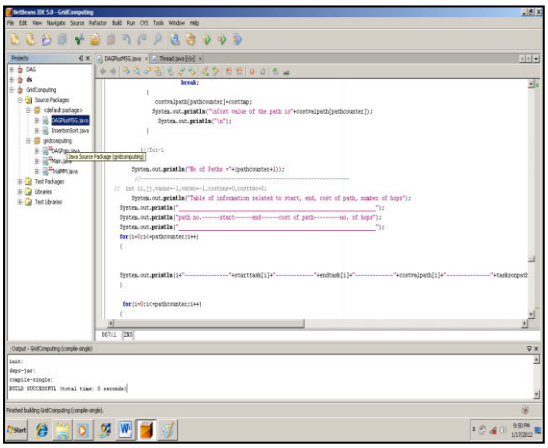

Figure-1. Code to generate DAG.

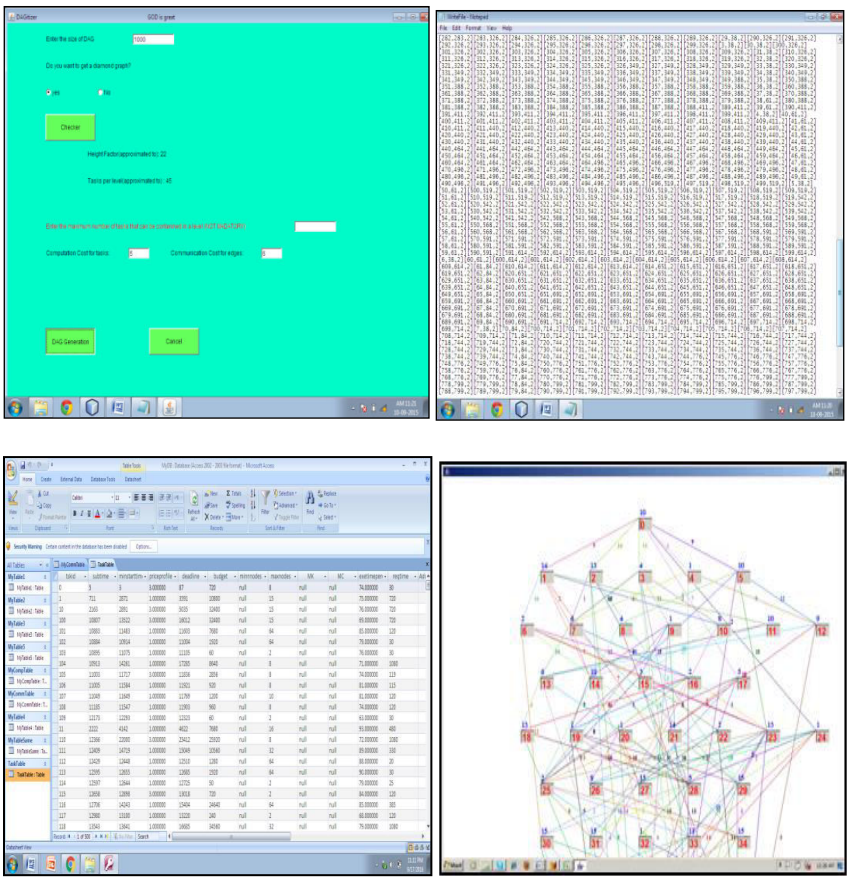

Figure-2. DAG generation (User Interface Screen, Values in Comma Separated Format, Values in Table, Pictorial View).

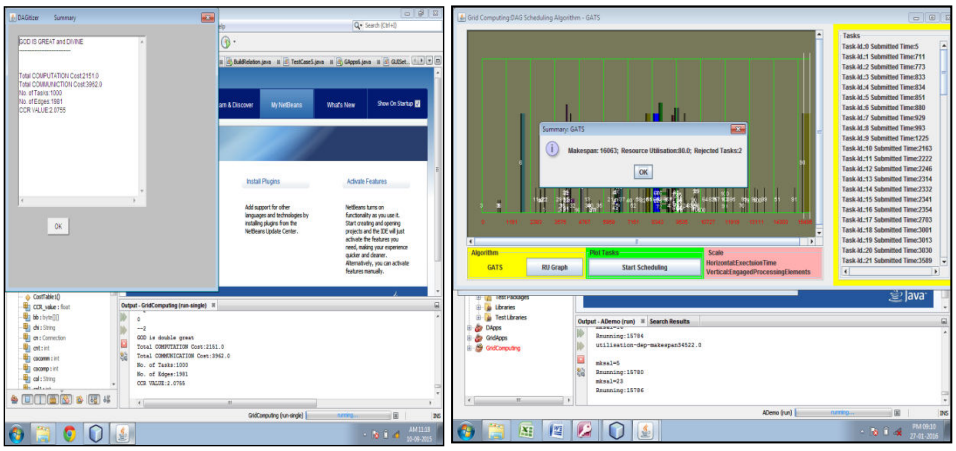

Figure-3. Summary of DAG results, simulation environment screen shot.

Now, the tool has been revised to calculate the QFT of the tasks (obvious that other factors would be calculated to arrive at QFT). Many combinations of task list and resource list (various processing speed) have been given to generate more number of DAGs of various sizes and characteristics to process the algorithm.

The performance of the algorithms has been scaled are compared with respect to the following attributes:

a) Schedule extent
b) Work load between resources
c) Execution time of individual tasks

Schedule length is the principle factor that decides the eminence of an algorithm. It is found that the proposed algorithm comparatively gives a lesser value of schedule length than EDOS algorithm.

Work load distribution between resources is almost equal to both the algorithms, here we consider work load as the total time that a particular resource is busy.

Execution time of individual tasks is the time taken by a particular task to complete itself and get ready for the availability to precede the execution of tasks in successive levels. And we found that some of the tasks in the earlier stages of scheduling are having the same execution time when executed in both the algorithms.

Many sets of data have been tested on both the algorithms and found that our algorithm gives the best result with respect to schedule extent. As the proposed algorithm totally focuses on the dependencies of a particular task, it is found that the idle time of a resource between one execution and another is very smaller.

Simulation results are tabulated in Tables 1, 2, 3 and the comparison outcomes of Tables (1, 2, 3) are correspondingly expressed in terms of charts in Figures 4, 5 and 6. Here, we use the same task graph (DAG) of 50 tasks and the completion time of every 10 tasks executed in 4, 8, 16 resources are shown in Table-1, Table-2, Table 3 respectively. Figure-4 presents the comparison of algorithms with respect to the completion time (schedule extent) of 50 tasks executed using 4 resources; PD-SECT, EDOS result with the schedule extent of 63 and 69 time units (1 msec) respectively. Figure-5 presents the proficiency of PD-SECT over EDOS when executed the same task graph using 8 resources and the schedule length is found to be reduced by 6 time units. Figure-6 presents the competence of PD-SECT over EDOS when using 16 resources for executing the same task graph; and the proposed algorithm proves its efficiency by an earlier completion by 3 time units.

PD-SECT shows a beneficial performance. In intense graph, it is observed that the difference between the schedule length of PD-SECT and EDOS is very big; and in a small graph, the difference is found to be small (and is obvious). It is found that the proposed algorithm produces consistent results on various executions experimented with task graphs of diverging task count on same set of resources.

Table-1. Schedule Extent of 50 tasks executed using 4 resources.

NO. OF TASKS	SCHEDULE LENGTH(*) (completion time of every 10 tasks)	
	A1	A2
50	19	11
	27	24
	42	46
	56	59
	*63	*69

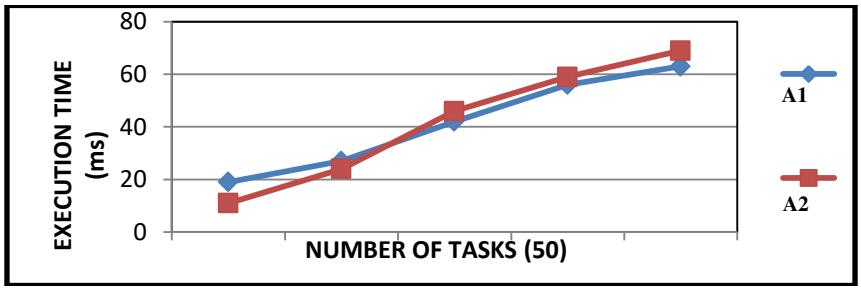

Figure-4. Performance comparison of PD-SECT (A1) and EDOS (A2) for values in Table-1.

Table-2. Schedule Extent of 50 tasks executed using 8 resources

NO. OF TASKS	SCHEDULE LENGTH(*) (completion time of every 10 tasks)	
	A1	A2
50	7	9
	11	11
	16	17
	25	31
	***29**	***35**

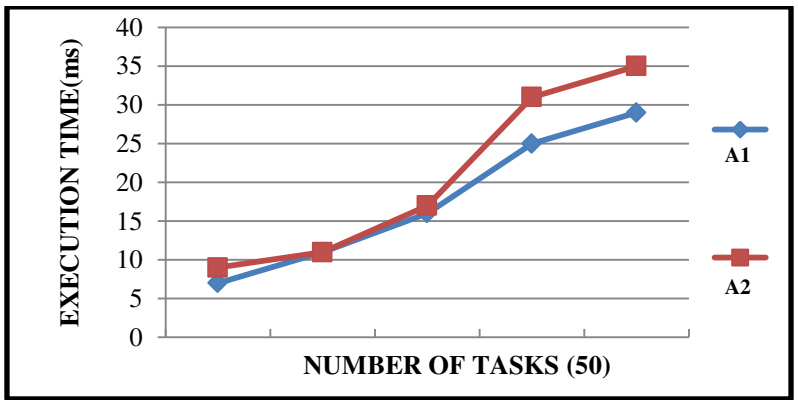

Figure-5. Performance comparison of PD-SECT (A1) and EDOS (A2) for values in Table-2.

Table-3. Schedule Extent of 50 tasks executed using 8 resources

NO. OF TASKS	SCHEDULE LENGTH(*) (completion time of every 10 tasks)	
	A1	A2
50	2	2
	4	4
	7	9
	9	11
	***11**	***14**

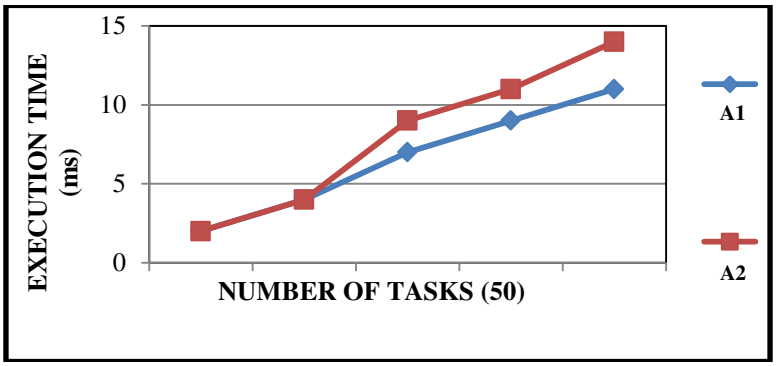

Figure-6. Performance comparison of PD-SECT (A1) and EDOS (A2) for values in Table-3.

6. TIME COMPLEXITY

The proposed algorithm results in a time complexity of O (2e x p), 'e' is the number of edges and 'p' is the count of resources used to execute the tasks. Here, the resource is finalized for the child (thereof for the grandchild) when the parents (parents with respect to child/grandparents with respect to grandchild) are executing themselves and therefore the idle times of the resources are considerably reduced.

7. CONCLUSIONS

It was observed that the proposed algorithm has produced better results on test beds by efficiently reducing the schedule extent and balancing the load between resources as well. It is also found that, the rate of best results is very high among almost all simulations. The proposed algorithm significantly gives an effective speed up than the compared algorithm. The consistency is prevailed on various categories of experiments conducted; and the tested categories are: (i) keeping the same task graph with varied number of resources, (ii) taking different task graphs with same set of resources, (iii) keeping same count of tasks but DAGs are generated with varied relations (a node-id=1(parent in level 1) of a graph will have a set of children(say node-id=2(child 1 in level 2), node-id=3(child in level 2), node-id=7(child in level 3) and node-id=17(child 4 in level 7) and the same node-id=1(parent) of an another graph may not have the same set of children as the graph generated previously) with same set of resources.

The future plan is to improve the algorithm by (i) pertaining task duplication phase, (ii) defining rescheduling point and (iii) to incorporate the algorithm in some real time applications.

REFERENCES

[1] Radu Prodan, Marek Wieczorek. 2010. Bi-Criteria Scheduling of Scientific Work Flows. IEEE Transactions on Automation Science and Engineering. 7(2): 364-376.

[2] Y. Kwok and I. Ahamed. 1996. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. IEEE Transactions on Parallel and Distributed Systems. 7(5): 506-521.

[3] G. C. Sih and E. A. Lee. 1993. A Compile Time Scheduling Heuristic for Interconnection and constrained Heterogeneous Processor Architecture. IEEE Transaction on Parallel and Distributed Systems. 4(2): 175-186.

[4] Haluk Topcuoglu, Salim Hariri, Min-You Wu. 2002. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Transactions on Parallel and Distributed Systems. 13(3): 260-274.

[5] Kai Hwang, Faye A. Briggs. 1985. Computer Architecture and Parallel Processing. McGraw-Hill Book Company, International Edition.

[6] Mohammed I Daoud, Nawwaf Kharma. 2011. A Hybrid Heuristic-Genetic Algorithm for Task Scheduling In Heterogeneous Processor Networks. Journal of Parallel and Distributed Computing. 71: 1518-1531.

[7] Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran. 2006. Fundamentals of Computer Algorithms. Galgotia Publications Pvt. Ltd.

[8] D. I George Amalarethinam, P. Muthulakshmi. 2014. Scheduling Strategies in Grid Computing Environment: A Survey. International Review on Computers and Software. 9(7): 1134-1152.

[9] Fang Peng Dong and Selim G. Akl. 2006. Scheduling Algorithms for Grid Computing: state of Art and Open Problems. Technical Report No. 2006-504, School of Computing, Queen's Unversity, Kingston, Ontario. pp. 1-55.

[10] D.I.George Amalarethinam, F.Kurus Malai Selvi. 2011. An Efficient Dual Objective Grid Workflow Scheduling Algorithm. International Journal of Computer Applications. pp. 7-12.

[11] Hamid Mohammadi Fard, Hossein Deldari. 2008. An Economic Approach for Scheduling Dependent Tasks in Grid Computing. The 11th IEEE International Conference on Computational Science and Engineering.

[12] Mandal A., Kennedy K., Koelbel C., Mrin G., Crummey J., Lie B., johnsson L. 2005. Scheduling Strategies for Mapping Application Workflows on to the Grid. The 14th IEEE International Symposium on High Performance Distributed Computing.

[13] D. I George Amalarethinam, P. Muthulakshmi. 2012. DAGITIZER - A Tool to Generate Directed Acyclic Graph through Randomizer to Model Scheduling in Grid Computing. Book: Advances in Intelligence and Soft Computing, Springer Verlag. 167: 969-978.