Journal of Theoretical and Applied Information Technology
10th September 2014. Vol. 67 No.1
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A PROFICIENT LOW COMPLEXITY ALGORITHM FOR PREEMINENT TASK SCHEDULING INTENDED FOR HETEROGENEOUS ENVIRONMENT

[1]D. I. GEORGE AMALARETHINAM, [2]P. MUTHULAKSHMI

[1]Department of Computer Science, Jamal Mohamed College, Trichirappalli, India.
[2]Department of Computer Science, Faculty of Science, SRM University, Chennai, India.
E-mail: [1]di_george@ymail.com, [2] lakshmi.mailspace@gmail.com

ABSTRACT

The major component of any computing system is the scheduling technique that coordinates the entire system. Heterogeneous environments like grid computing environment provide the accessibility to use wide range of resources that are located around the world. In such environment resource management becomes a complex issue due to various factors like high computational demand, diversity among the tasks, heterogeneity of resources, and heterogeneity of vendors who offer services, dynamic nature of resources. An effective scheduling may increase the efficiency of resource management systems. This paper addresses a grid scheduling algorithm. The algorithm is devised to schedule the tasks on available resources. The performance of the algorithm has been evaluated for arbitrary and regular graphs. The algorithm and the compared algorithms are implemented in Java. The algorithm begins by grouping the tasks. Then tasks from various groups are compared and prioritized for scheduling. The results show that the proposed algorithm outperforms the existing algorithms. The test results of the algorithm justify that the algorithm encourages maximum utilization of resources, minimized makespan and balanced load across resources.

Keywords: *Resource Finalizing Factor, Load Balancing, Schedule Length, Quick Finish Time, Prominent Parent, Promising Successor.*

1. INTRODUCTION

Business and scientific applications involve large scale computations. These applications look forward to computing environments having promises like low cost budget, quick finish time and highly accurate services. Such applications have leaded us to the use of distributed computing.

A distributed computing environment is an aggregation of unlimited resources. In such environments, it is not easy to co-ordinate resources and consumers because of the heterogeneity of hardware and software; viz., networks, protocols and other resources. It is very challenging to manage resources and offer best services to those who require it. Grid environment is a kind of distributed environment where the computing power of multiple resources is used in a parallel fashion to solve bigger scientific problems in short span of time.

Mismanagement of resources may lead to low efficiency and immense loss of quality. It is a hot challenge to achieve high performance through proper management of resources. Scheduling optimizes the objective function that is involved with selection of resources. In scheduling, every aspect is completely based on decision(s). Task scheduling problem is known to be NP-complete [1]. Employing a proper scheduling technique ensures time management and guarantees the efficiency of the computing environment. The fairness of a computing environment is appreciated not only for its efficiency in producing proper results but also for the completion of tasks within the deadline. In grid applications, the workflow is structured using task dependencies [2] and is shown using Directed Acyclic Graph (DAG). The communication cost and computation cost are the imperative elements that decide the mapping between tasks and resources in a workflow structure.

Grid Commerce environment fixes the strategies for pricing to use the grid resources. An interface is required to facilitate grid resources trading. The interface is a grid broker that sits in between service providers and users. The Grid Broker System architecture is shown in Figure 1 with its components namely (i) service providers (ii) users and (iii) grid broker. The resources let themselves to use their CPU time. Such resources make themselves known in the grid system as service providers and they establish policies to use them. The policies may include price for usage, advanced reservation, time of availability, and so on. The users submit their applications for execution and may expect their applications to be executed in an efficient and economic manner. The applications may be related to business, astronomy, scientific, research and so on. The users may set priorities and parameters to their applications. Grid resource broker maintains the status of resource availability and users demand and use the status information to allocate resources to users. Grid resource broker makes use of the information given by the service providers and the users to find a best schedule for applications. Grid scheduling is the objective function of the grid resource broker. Generally scheduling aims at low cost and minimum execution time for executing applications, which may lead to improved quality and efficiency. Grid scheduling involves resource discovery, information gathering, mapping of resources and applications for execution, monitors the progress of the submitted applications.

In parallel task scheduling, there may be gaps between scheduled tasks on the resource. This may happen due to (i) the availability of free resource time not suitable for the priority task in the queue, (ii) the dependency of the task that waits for the parent task/tasks to commute from other resources. This may cause an extended completion time of the task graph.

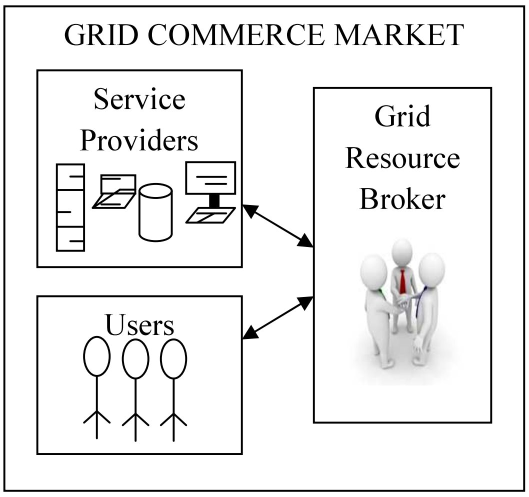

Figure 1. Grid Broker System Architecture

Backfilling can be used to ensure the better use of the available resource time. When backfilling is used the tasks priorities may go out of order. In this paper, an algorithm is proposed that deals with the resource allocation issues. The objective is to (i) reduce the makespan of DAG (ii) manage the effective load distribution among the available resources (iii) to avoid gaps between scheduled tasks without violating the precedence constraints.

The continuation of the paper is compartmentalized as follows: In Section 2, the related works are given which help us to propose some evolutionary ideas. Section 3 gives the view of task scheduling problem and task graph. In section 4, the research problem is described. In section 5, the proposed algorithm is given. Section 6 shows an illustrative example of DAG and its primary attributes required for the algorithm. Section 7 presents the simulated results of the proposed and compared algorithms. Section 8 packages the time complexity of the proposed algorithm. Section 9 contains the conclusion and the plan of future work.

2. RELATED WORKS

In general, the static scheduling problem is represented by Directed Acyclic Graph (DAG). In DAG, the active participants are the set of nodes and set of edges. The nodes represent the tasks and the edges represent the task (inter task) dependencies. The nodes and edges would be given values; computation costs and communication costs (both are usually integers). The computation costs and communication costs are assigned for nodes and edges respectively. The DAG structure is organized on dependency policy.

In the taxonomy, scheduling branches into static and dynamic. In static model/deterministic model, the characteristics of the problem are identified earlier and resources are assumed to be available all the time, task failure is assumed to be not happening. In dynamic scheduling, everything is decided on-the-fly. Scheduling algorithms are always expected to be highly intensive towards the earlier problem completion. The task scheduling algorithm is to allocate resources to execute tasks and the same is done by establishing an order for tasks [3].

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.

Task Graph Scheduling heuristics are based on list scheduling approach. Generally, list scheduling prioritizes the tasks with respect to some policies. Based on the priorities tasks will be allowed to execute on resources. The task having maximum priority will be the first to get allocated to the resource and the allocation will be continued until no more tasks remain unscheduled. List Scheduling involves two phases namely (i) the task selection phase, which is responsible for choosing the task based on priority (ii) the resource selection phase, responsible for choosing the resource that can execute the chosen task in a minimized execution time. Most of the list scheduling algorithms follows the precedence constraints that are framed out of considering the computation costs alone. The insertion policy is encouraged in algorithms Heterogeneous Earliest Finish Time (HEFT) [3], Insertion Scheduling Heuristic (ISH) [4] where the idle time slots are used for executing the ready task that fits the size of the slot. The idle time slots are formed due to the communication delay between tasks. The tasks that cannot be executed in an earlier time on other resources can be executed in the idle time. The insertion policy may improve the efficiency of algorithm as it would result in an earlier completion of the DAG, but the priorities derived for the tasks execution may not be followed.

An extensive study on scheduling policies and scheduling algorithms has been made in [5] [6] [7] [8] [9]. Highest Level First (HLF) [10] is a list scheduling algorithm, which considers the computation time of tasks. HLF assigns priority to the task that has the longest path from itself to an exit task. Earliest Deadline First (EDF) [11] assigns priority to the task that has the earliest deadline.

Min-Min algorithm [12] is based on Minimum Completion Time (MCT) of tasks on available resources. The Min-Min algorithm first finds the minimum execution time of all tasks. Then the task with least execution time is chosen for execution. The task selection repeatedly happens until no more unscheduled task exists.

A contention- aware task duplication scheduling algorithm has been proposed by Oliver Sinnen et al [13]. The algorithm is based on scheduling algorithms that are devised for the contention model and the algorithm uses duplication strategy under classic model. The data ready time of any task is known earlier by tentatively scheduling edges on the communication links. Data ready time of a task is defined as, the time that the data is available for the task to start its execution. The insertion technique is used for the tentative scheduling and is to remove redundant tasks and redundant edges.

Efficient Dual Objective Scheduling (EDOS) [14] 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) [15] is a list algorithm with greedy approach. This algorithm is devised by considering both the makespan and cost that would be spent on utilizing the resources.

Mandal et al [16] use in-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. TASK SCHEDULING PROBLEM AND DIRECTED ACYCLIC GRAPH

Generally task scheduling problems are parallel applications. The workflow of the parallel application cab be shown using DAG. DAG is also known as Task Graph. DAG structures the workflow of the application. DAG is a tuple space consists of set of nodes and set of edges, which can be represented as G= (T, E). Where, 'G' is the directed acyclic graph, 'T', the set of nodes (tasks); a task is represented as t_i; $1 \leq t_i \leq nt$ (number of tasks in DAG). 'E' denotes the set of edges (inter task dependencies/communication links); an edge is shown as e_i; $1 \leq e_i \leq ne$ (number of edges in DAG), and it establishes a parent-child relationship between a pair of tasks.

The hierarchy of tasks is shown by placing them in different levels. Tasks are distributed in various levels of the DAG. Each level can have one or more tasks. Level 'i' can have its child/children in any of the levels greater than itself. There cannot be edges between tasks of the same level. Tasks belonging to the same level could be executed concurrently as there is no task dependency between them. Tasks found between the first level and (n-1) level will have child/children, where 'n' is the last level. The least degree level will have tasks of no parent(s) and the most degree level will have tasks of no child/children.

The COMPCost of t_i would be the time taken by task 'i' to execute itself on a resource. In DAG scheduling, tasks scheduled on resources are non preemptive. Each t_i must be executed in the same resource until it gets finished. Each e_i is represented by the COMMCost, which is the time taken to transfer data between resources (this would be spent 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 can start its execution only if all of its parents have completed their execution. This is because the child might need data from its parent(s) to start its execution [17]. The successor starts its execution when all the necessary data from its parents are available. This may lead to idle slots/holes between scheduled tasks which may be used to execute a task that fits the size of the idle slot.

4. RESEARCH PROBLEM AND DESCRIPTION

In this paper, an algorithm is proposed that uses the list scheduling technique and clustering technique. In general, list scheduling establishes an order by allotting priorities for tasks. The order will be followed while executing the tasks on resources. Clustering technique is used to group tasks into clusters. The tasks belonging to the same cluster are executed in the same resource. Finally, clusters will be mapped on the available resources.

The proposed algorithm uses the idea of multi-stage graph of dynamic programming technique [18] to progress from the source towards the destination and the bin packing technique [18] to form groups of tasks for prioritization. During the literature survey, it is found that most scheduling algorithms concentrated on computation time for tasks assignment on resources. Also it is observed that load balancing among the resources is not given importance. The proposed algorithm considers both the computation and communication costs for scheduling the tasks as both are equally important to get a best allocation. It is tried to improve the performance by reducing the movement of data between resources by selecting the fittest resource to execute. It is observed that resources are idle when another resource works for a long time to complete all its assigned tasks. Load balancing is a factor is not considered when there is an excess dedication of resources to a particular set of tasks. An attempt is made to concentrate on load balancing among resources and to complete the task graph earlier and also to make resources available for other task graphs in the grid system to execute.

When a DAG is submitted for execution, the algorithm starts by finding the paths that connect the entry task(s) and the exit task(s). Then create lists and bins with respect to the count of paths. The bin is nothing but a waiting queue. Each bin correspond a list. Each path is populated in a list. Each bin has a capacity of sum of greatest COMPCost of three tasks of in the graph. Bins can be queued with a pack of three tasks from their respective list; which means the bin is populated with a set of tasks containing parent, child, and grand-child. Now, lists of Common Parents (CPL) and Uncommon Parents (UCPL) are generated from the bins. Based on the population of parents in lists and their dependencies, parents are scheduled to resources. The parent having the maximum occurrence and maximum number children is given priority. Prioritized task can be executed in resource that minimizes the finish time (FT).

The out-degree of a node is the count of out-going edges from the node. Task dependency of a task is the out-degree of the task. For the exit task the out-degree is zero mean that no task is depending on it. The number of parents of a node is the in-degree of the node. For the entry task the in-degree is zero. The child having the maximum COMMCost and minimum COMPCost on other resource would be executed in the same resource where its parent was executed. The other tasks would be executed in resources that minimize the finish time. As soon as the parent is scheduled, the Task Allocation Table (TAT) will be updated with the values of task-id, resource-id; also the bins contents would be altered by removing the parents and including the next task from the list to the next of grand-child. Now the child has become the parent, grand-child has turned as child and new task entered would be the grand-child. When there are no tasks available for any component in the bin, pack it with zero. The process of loading the bin and prioritizing would be repeatedly done as long as tasks are available in lists. The task selection is based on the dependencies and the computation, communication cost. The resource selection is based on the availability of the resource at the earliest start time of the task and minimum execution cost that a resource supports for a task. Figure 2 illustrates the methodology of the algorithm through an activity diagram.

4.1. The arithmetic of the proposed algorithm

Expected Computation Cost of a task

The algorithm starts by calculating Expected COMPutation Cost (ECOMPCost) on resource for each task with respect to speed of each resource.

for i= 1 to maxtasks do

for j= 1 to maxresources do

$ECOMPCost(t_i,r_j)=COMPCost(t_i)/speed(r_j)$ (1)

Average Computation Cost of a task on available resources

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

for i= 1 to maxtasks do

$$ACOMPCost(t_i)=\sum_{j=1}^{maxresources} COMPCost(t_i,r_j)/maxresources \quad (2)$$

Here, the average data transfer rates between resources and average communication start up time are considered to be 1.

Quick Start Time and Quick Finish Time

For the tasks at the first level, the QST is Zero.

$QST(t_i,r_j)=0;1\le t_i\le nt;t_i\in level0,1\le r_j\le maxresources$ (3)

For the others tasks, the recursive computation of Quick Start Time (QST) and Quick Finish Time (QFT) are calculated as follows.

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

$RUT(t_p(i))=ACOMPCost(t_p(i))$ (5)

$RTR(t_p(i),r_j)=\max\{(t_p(i)),RUT(t_p(i))+COMMCost(t_p(i),t_c)\}$ $t_p(i)\in parents(i)$ (6)

$QST(t_c,r_j)=\max\{RRT(r_j),RTR(t_p(i),r_j)\}$ (7)

$QFT(t_c,r_j)=ACOMPCost(t_i)+QST(t_c,r_j)$ (8)

where, RRT is Resource Ready Time, FT is Finish Time, RTR is Reach Time on Resource, $t_p(t_c)$ is parent task of child task, RUT refers to Resource Utilization Time,

Resource Finalization Factor (RFF)

The tasks in the first level would be executed on the resources that could complete the execution in Minimum COMPutation Cost (MCOMPCost).

$RFF(t_i)=Resource(\min\{ECOMPCost(t_i)\});$ $t_i\in firstlevel$ (9)

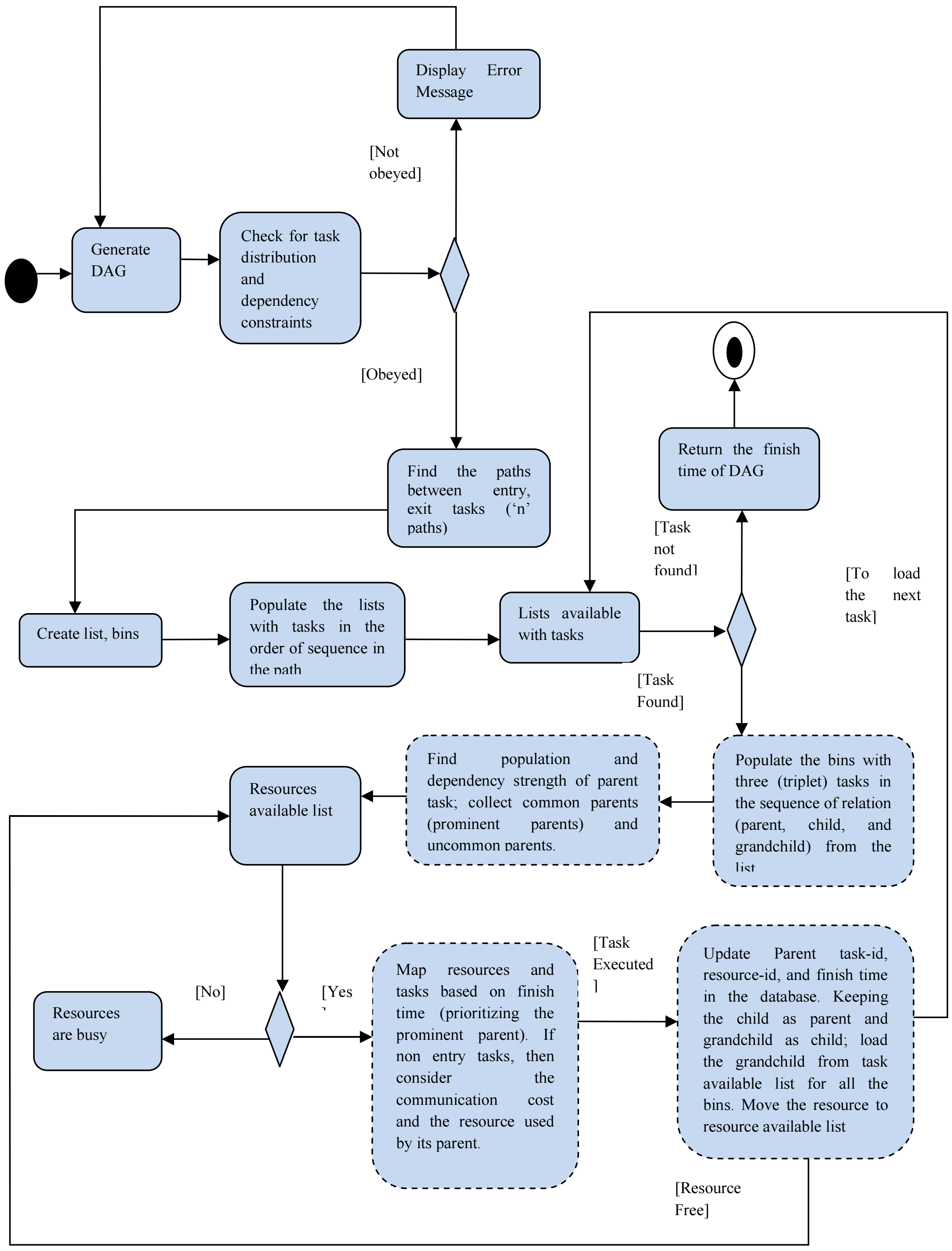
Display Error Message
[Not obeyed]
Generate DAG
Check for task distribution and dependency constraints
[Obeyed]
Find the paths between entry, exit tasks ('n' paths)
Return the finish time of DAG
[Task not found]
[To load the next task]
Create list, bins
Populate the lists with tasks in the order of sequence in the path
Lists available with tasks
[Task Found]
Find population and dependency strength of parent task; collect common parents (prominent parents) and uncommon parents.
Populate the bins with three (triplet) tasks in the sequence of relation (parent, child, and grandchild) from the list
Resources available list
[Task Executed]
Resources are busy
[No]
[Yes]
Map resources and tasks based on finish time (prioritizing the prominent parent). If non entry tasks, then consider the communication cost and the resource used by its parent.
Update Parent task-id, resource-id, and finish time in the database. Keeping the child as parent and grandchild as child; load the grandchild from task available list for all the bins. Move the resource to resource available list
[Resource Free]

Figure 2. Methodology of the algorithm

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:

$$RFF(t_c(i))=\max\{COMMCost(t_p,t_c(i))\}+\min\{ECOMPCost(t_c(i))\};t_c(i)\in children(t_p) \quad (10)$$

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; also 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 the other parents. Then execute the t_{gc} in the same resource by moving the data from other parents. The other promising successors (grandchildren) are executed with respect to their MCOMPCost and RFF in other available resources that minimize the finish time.

Schedule Length

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).

$$SL(DAG)=\max\{CompletionTime(r_i)\}; 1\leq r_j\leq maxresources \quad (11)$$

Resource Deployment Time

Resource Deployment Time (RDT) is the ratio between task completion time of a resource and schedule length of the DAG. The Average Resource Deployment Time (ARDT) can be given as the ratio between the summation of resource deployment time of all the available resources and the number of resources. This can be given by,

$$RDT(r_i)=CompletionTime(r_i)/ScheduleLengthofDAG; 1\leq r_j\leq maxresources \quad (12)$$

$$ARDT(resources)=\sum_{j=1}^{maxresources} RDT(r_i)/maxresources \quad (13)$$

5. THE ALGORITHM: TRIPLET BIN TASK GROUPING and PRIORITIZNG (TBTGP)

```
1.  find all the paths between the entry task(s) and the exit task(s)
2.  for each path , create a list//path(i) corresponds
                                        to list(i), 1≤i≤maxpaths
3.  populate the nodes of the path in the list
4.  for each list, create a bin,//list(i) corresponds
                                        to bin(i), 1≤i≤maxlists
5.  k=1;
6.  while(unassigned task in the lists) do
7.  {
8.  for(i=1 to maxlist)
9.  {
10. for(j=k to k+2)
11.  {
12.     if(task(j)==null)
13.     task(j)=0;
14.     pack the bin with the triplets (parent-k,child(k),grandchild(k));
15.  }
16. }
17. Consider all the bins and their data for finding the common-parent-list and uncommon-parent-list
18. while(unassigned tasks in CPL && UCPL)
19. {
20.    if(entry level tasks)
21.        update the TAT(task-id, proc-id that minimizes the FT)
22. else
23. {
```

Journal of Theoretical and Applied Information Technology
10th September 2014. Vol. 67 No.1
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
24.         if(max(RFF(children))) then
25.         {
26.             assign in same the resource where parent had completed its execution
27.             update the TAT(task-id, proc-id)
28.         }
29.         if(min(RFF(children)))
30.             {
31.               compute RFF(grandchildren)
32.
33.               if(prominent parent) then // that has more dependencies
34.                 assign the task (grandchild(ren)) where prominent parent was executed and
                    continue the execution by obtaining data from other parents

35.               else
36.               assign the task grandchild(ren) to resource that could execute in resource
                  having mcompcost with respect to RFF
37.               update the TAT(task-id, proc-id)
38.               }
39.   }
40.   }
41.   k=k+1;
42.   }
```

6. EXPERIMENTS

A sample DAG is shown in Figure.3, which has 7 tasks and 9 communication links established between the tasks. The COMMCost and COMPCost are shown in Table 1, Table 2 respectively. P1, P2 used in Table 2 are the available resources. The algorithm decides and assigns the tasks to be executed on P1 and p2.

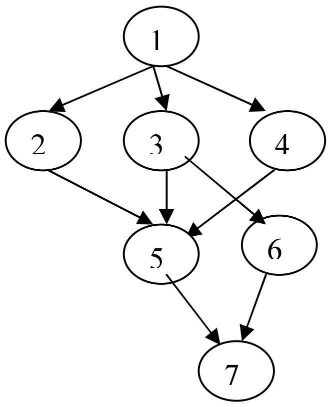

Figure 3. Sample DAG

Table 1.Communication Cost

Parent	Child	Communication Cost
1	2	12
1	3	7
1	4	6
2	5	11
3	5	13
3	6	14
4	5	17
5	7	11
6	7	10

Table 2. Computation Cost

Task-Id	P1	P2
T1	9	12
T2	3	7
T3	6	4
T4	11	5
T5	13	6
T6	5	13
T7	16	11

Tasks in the paths have been populated in the lists and are shown in Table 3 and Table 4.

Table 3. Path Details

Paths	Tasks in the paths
1	1,2,5,7
2	1,3,5,7
3	1,3,6,7
4	1,4,5,7

Table 4. List and Population of Tasks

List	Populating the list from paths			
1	1	2	5	7
2	1	3	5	7
3	1	3	6	7
4	1	4	6	7

Table 5 gives the illustration of packing the bins with tasks.

Journal of Theoretical and Applied Information Technology
10th September 2014. Vol. 67 No.1
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 5. Initial Bin Packing

Tasks	Bin 1	Bin 2	Bin 3	Bin 4
Parent	1	1	1	1
Child	2	3	3	4
Grand-child	5	5	6	5

Table 6 shows the removal of the parent in the previous level, the transition of child into parent, grand-child into child, insertion of the next level dependent tasks.

Table 6. Bin Packing after assigning First Level

Tasks	Bin 1	Bin 2	Bin 3	Bin 4
Parent	2	3	3	4
Child	5	5	6	5
Grand-child	7	7	7	7

The schedule lengths of the compared algorithms for the sample DAG shown in Figure.3 are shown in Table 7.

Table 7. Schedule Length of the Algorithms

Algorithms	Schedule Length
TBTGP	54
HEFT	58
Min-Min	60

7. RESULTS AND DISCUSSION

Randomly generated graphs with diverse properties have been used for assessing the proposed algorithm with Min-Min algorithm and HEFT algorithm. A task graph generator has been developed to generate various sizes of DAG [19] with varied potentialities. The DAGs are generated using normal distribution. The generated DAGs have been used to study and evaluate the performance metrics. It is observed that almost equal schedule length is obtained when a very small graph (3 tasks) is used. The comparisons are based on the factors namely (i) schedule length ratio (ii) speedup and (iii) load balance among the resources. A wide range of Communication to Computation Cost Ratio (CCR) values have been used for the scaling of performances.

(i) Schedule length ratio of a graph is given by the ratio between the schedule length and the total of minimum computation cost of each individual task in the graph. Low value on schedule length ratio is obtained on executing various graphs generated randomly.
(ii)Speedup ratio of a graph is the ratio between the minimum computation cost obtained by scheduling all the tasks to a single resource and the makespan. In this metric, it is found that the speedup values of graphs are always found to be more than one.
(iii)Load balance among resources is the fair distribution of execution time on resources at the completion of task graph execution. It is found that the TBTGP algorithm performs well in balancing the load.
(iv)Communication to Computation Cost Ratio of a task graph is defined as the ratio between the average of all communication costs of communication links and the average of all computation costs of tasks in the graph.

DAGs of various sizes and different CCR values are used for comparison. The comparison of algorithms with respect to makespan is shown in Table 8.

Table 8. Algorithms with Makespan

No. of tasks in task graph	Algorithms and makespan (5 CPUs Used)		
	TBTGP (msec.)	HEFT (msec.)	Min-Min (msec.)
50	5	7	11
100	8	13	15
125	11	14	17
150	16	19	22

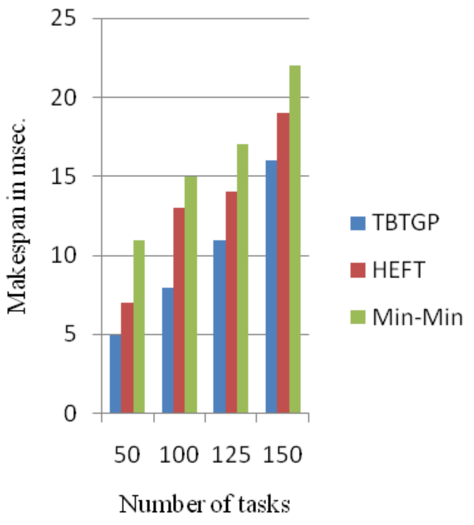

Figure.4. Comparison of makespan

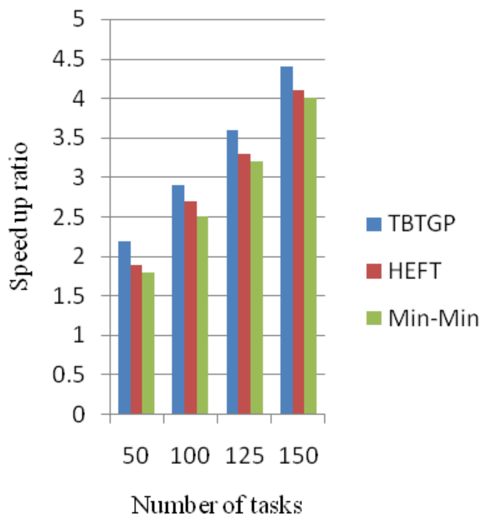

Figure.5. Comparison of speed up ratio

Figure 4 and Figure 5 show that the TBTGP algorithm is consistently giving better results with respect to the factors makespan and speedup ratio.

The resource deployment time is found almost equal in most of the experiments. And it is proven that the load is balanced among the resources.

8. TIME COMPLEXITY

The time complexity of the algorithm in finding the task to schedule gives O(2e+n) when the inter task dependency among tasks is found to be high. In order to find the resource to execute the task, the time complexity arrives at O(p). On consolidating, O(2e+n+p) is found to be the time complexity of the proposed algorithm.

9. CONCLUSION

An algorithm called Triplet Bin Task Grouping and Prioritizing is proposed in this paper. It works for static task scheduling. Many algorithms that are implemented using Java of similar kind have been studied and encourage us to implement the algorithm using Java. The notable differences in the performances are a proof for the efficiency of TBTGP. Better quality results could be achieved for even bigger task graphs. When looking at the lower CCR values, the TBTGP algorithm and Min-Min are close in their results. When scaling with higher CCR values, the algorithms have considerable differences in their results and their performance ranking could follow the order in which first comes TBTGP followed by HEFT algorithm and Min-Min algorithm. It is observed that TBTGP algorithm guarantees load balancing among the resources for various task graphs. As an extension of this work, we have planned to work on duplication technique and contention awareness on links.

REFERENCES:

[1] M. R. Gary, D. S. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness”, *W.H. Freeman and Co*, 1979.

[2] Radu Prodan, Marek Wieczorek, “Bi-Criteria Scheduling of Scientific Work Flows”, *IEEE Transactions on Automation Science and Engineering*. Vol. 7, No. 2, April 2010, pp. 364-376.

[3] Haluk Topcuoglu, Salim Hariri, Min-You Wu, “ Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 3, March 2002, pp. 260-274.

[4] B. Kruatrchue, T. G. Lewis, “Duplication Scheduling Heuristic, a New Precedence Task Scheduler for Parallel Systems”, *Technical Report 87-60-3*, Oregon state University.

[5] Y. Kwok and I. Ahamed, “Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors”, *IEEE Transactions on Parallel and Distributed System,* Vol.7, No. 5, May 1996, pp. 506-521.

[6] G. C. Sih and E. A. Lee, “A Compile Time Scheduling Heuristic for Interconnection and constrained Heterogeneous Processor Architecture”, *IEEE Transaction on Parallel and Distributed Systems*, Vol. 4, No. 2, Feb 1993, pp. 175-186

[7] H. Chen, M. Maheswaran, “Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems”, *Proceedings of the International Parallel and Distributed Processing Symposium*, 2002, pp. 88-97.

[8] H.D Karatza, “Performance of Gang Scheduling Policies in the presence of critical sporadic jobs in distributed systems”, *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, 2007, pp. 547-554.

[9] X. Qin and H. Jiang, "Dynamic, Reliability-Driven Scheduling of Parallel Real-Time Jobs in Heterogeneous Systems", *Proceedings of International Conference on Parallel Processing*, 2001, pp. 113-122.

[10] T. L. Adam, K. M. Chandy, J.R. Dickson, "A Comparison of List Schedules for Parallel Processing Systems", *Communications of the ACM*, 1974, pp. 685-690.

[11] C. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *Journal of the ACM*, 1973, pp. 46-61.

[12] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen and Richard.F.Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems", *Heterogeneous Computing Workshop*, 1999, pp. 30-44.

[13] O Sinnen, A To, M Kaur , "Contention-Aware Scheduling with Task Duplication", *Journal of Parallel and Distributed Computing*, 2011.

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

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

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

[17] Mohammed I Daoud, Nawwaf Kharma, "A Hybrid Heuristic-Genetic Algorithm For Task Scheduling in Heterogeneous Processor Networks", *Journal of Parallel and Distributed Computing*, Vol. 71, May 2011, pp. 1518-1531.

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

[19] Dr. D. I George Amalarethinam, P. Muthulakshmi, "DAGITIZER – A Tool to Generate Directed Acyclic Graph through Randomizer to Model Scheduling in Grid Computing", *Advances in Intelligence and Soft Computing*, Springer Verlag, Vol. 167, May 2012, pp. 969-978.