Novel Approaches for Scheduling Task Graphs in Heterogeneous Distributed Computing Environment

Ehsan Munir[1], Saima Ijaz[1], Sheraz Anjum[1], Ali Khan[3], Waqas Anwar[2], and Wasif Nisar[1]
[1]Department of Computer Science, COMSATS Institute of Information Technology, Wah Cantt, Pakistan
[2]Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan
[3]Department of Electrical Engineering, COMSATS Institute of Information Technology, Lahore, Pakistan

Abstract: *Distributed heterogeneous computing environment comprises of diverse set of interconnected resources that are capable of performing computationally complex tasks efficiently. In order to exploit the high performance of such a system, the task scheduling problem demands for the efficient mapping of the tasks. Because of its fundamental importance, the problem has been studied extensively and several algorithms have been proposed. In this paper, we propose two novel approaches for the task scheduling problem and compare the proposed work on the basis of randomly generated task graphs with the well-known existing algorithms. The simulation results elucidate on the basis of different cost and performance metrics that for most of the scenarios, the proposed approaches outperform the existing ones considerably.*

Keywords: *Heterogeneous distributed computing systems, directed acyclic graphs, task scheduling, task prioritization, makespan.*

Received December 4, 2012; accepted September 2, 2013, published online June 26, 2014

1. Introduction

Availability of distributed set of powerful machines, intercommunicating through high speed links, provides a computing platform for executing applications with multifarious computational demands. Applications running on such Heterogeneous Distributed Computing Systems (HDCS) are decomposed into set of tasks with or without dependencies among themselves. Optimal mapping of the tasks, i.e., their matching and then appropriate scheduling on diverse set of machines in a way to step up the overall efficiency of the system and gain promising potentials of the distributed resources is the main objective of the mapping algorithms for such a distributed HDCS. Mapping of tasks to machines should be done so as to reduce the overall execution time of the application.

Generally, task scheduling problem is modeled by Directed Acyclic Graph (DAG) in which application tasks are shown through nodes of the graph and data dependencies among the tasks are represented through edges with communication costs labeled on the edges and computation costs labeled on the nodes. The task scheduling problem addressed here is a static model as different properties of the application, such as execution times of the tasks on various machines and inter- task communication costs are known in advance.

Plethora of algorithms exists in the literature for solving the task scheduling problem but, being NP-complete [5], finding near optimal solution for the problem requires more efficient scheduling strategies. High s peed up and efficiency can be attained only if mapping of tasks on machines is done appropriately as it can truly exploit system parallelism. The motivation behind this research work is to propose some new strategies for excogitating ways to enhance the system performance. Two novel algorithms have been introduced in the paper that address the task scheduling problem and perform mapping of the tasks to machines in a way to lessen the overall time required for the execution of the application. Proposed work has been compared with the work in [6,13] in terms of different performance and cost metrics such as efficiency, speedup, Schedule Length Ratio (SLR) and makespan. The comparison illustrates that the proposed algorithms yield improved results over the existing work.

The organization of the paper has been made as: Section 2 demonstrates the task scheduling problem. A brief description of the related work has been given in section 3. Proposed algorithms are in section 4 and the results have been presented in section 5. In the end, section 6 concludes the paper.

2. Task Scheduling Problem

The task scheduling system consists of an application program, some environment to run the application on and a strategy for the scheduling. Generally, a DAG is used for this purpose, with a set V of nodes representing n tasks and set E of edges depicting the dependencies among the tasks. There is an entry task for each DAG with no parents and an end task with no children.

The heterogeneous computing environment, on which the application is to be executed, consists of a set *P* of *m* autonomous processors intercommunicating with each other through high speed networks of varying bandwidth described in B_{mxm} matrix. Estimated time to compute a task on every processor is given in a computation cost matrix *W* of size $n \times m$. The communication cost, $C_{i,j}$ for transferring output, $data_{i,j}$, of a task t_i to t_j , if both are being executed on same processors, is 0. If this is not the case, then following relation gives the communication cost, $C_{i,j}$ between the two dependent tasks.

$$C_{i,j} = data_{i,j} / B_{x,y} \tag{1}$$

For the sake of simplicity, data transfer cost is assumed to be 1.0 in this case. The whole scenario is explained with the help of an example DAG of Figure 1 with 10 nodes (tasks). Computation cost matrix is given in Table 1.

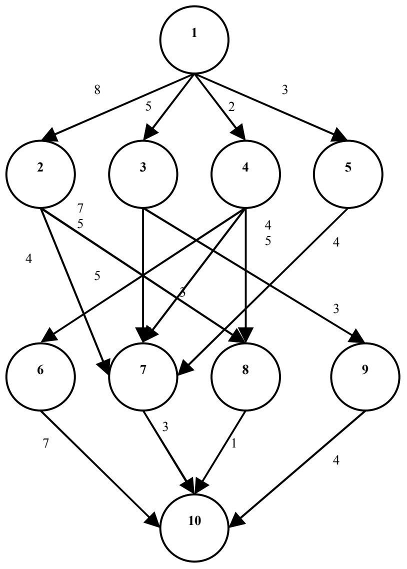


Figure 1. DAG.

Table 1. Computation cost matrix.

Task	P1	P2	P3
1	12	13	9
2	17	18	19
3	27	30	28
4	49	44	43
5	10	11	10
6	18	18	19
7	10	13	14
8	25	23	27
9	35	34	39
10	15	16	15

Let the earliest start time and earliest completion time for the execution of a task t_i on a processor p_j be $EST(t_i, p_j)$ and $ECT(t_i, p_j)$ respectively. *EST* for the entry task, on all the processors, is 0 i.e.,

$$EST(t_0, p_j) = 0 \tag{2}$$

For the rest of the tasks in the application graph, the *EST* and *ECT* values are recursively computed using the Equations 3 and 4 respectively. After the scheduling of a task t_i on a processor p_j, the *EST* and *ECT* become the Actual Start Time (AST) and Actual Completion Time (ACT) for the task t_i. An important consideration here is that a task can be scheduled to execute only if all of its parent tasks have already been executed.

$$EST(t_i, p_j) = \max\{avail[j], \max(ACT(t_p + C_{i,j}))\} \tag{3}$$

$$ECT(t_i, p_j) = W_{i,j} + EST(t_i, p_j) \tag{4}$$

Where, t_p belongs to the set of immediate predecessors of task t_i, $avail[j]$ is the time when processor p_j will be available for the execution of the task t_i and $C_{i,j}$ is the communication cost between the executing task and its parent task. Finally, when all the tasks are mapped to the appropriate processors, the overall completion time of the task graph gives the schedule length/makes pan for that application graph, i.e.

$$makespan = \max\{ACT(t_{end})\} \tag{5}$$

Efficient scheduling of the application requires adopting a scheduling strategy that minimizes the makes pan.

3. Related Work

High performance of the HDCS demands for the efficient scheduling strategies for an application. Because of its fundamental importance, the problem has been studied extensively and bunch of algorithms exist in the literature. The classification of the algorithms has been done into different categories such as list scheduling algorithms [5, 7, 9, 10, 13], guided random algorithms [3], cluster based [8] and task duplication algorithms [1, 2, 4].

List scheduling algorithms have been chosen as a research area for the work proposed in the paper. Here, priority based approach is followed and an ordered list of tasks is maintained on the basis of their priorities [13]. Few of the algorithms in this category that exist in literature are Mapping Heuristics (MH) [10], Modified Critical Path (MCP) [14], Levelized Min Time (LMT) [5], Dynamic Critical Path for Grids (DCP-G) [12], Heterogeneous Earliest Finish Time (HEFT) [13], Critical Path On a Processor (CPOP) [13] and Performance Effective Task Scheduling (PETS) [6]. A brief description of these algorithms is given below.

3.1. Mapping Heuristic

The computation cost for a task, in case of MH, is the ratio of number of instructions that are to be executed in the task and the processor speed. Priorities are assigned to the tasks on the basis of static upward ranks. The main drawback associated with the MH algorithm is that it does not follow insertion based strategy [13].

3.2. Levelized Min Time

LMT is a two phase algorithm, task prioritization and processor selection. In first phase, level-wise prioritization of the tasks is done such that a lower level task has higher rank than the higher level task. Second phase assigns the tasks to the fastest processor on the basis of computation and communication costs. LMT, however, considers only the computation costs of the tasks to assign the priorities.

3.3. Dynamic Critical Path for Grids

This algorithm considers the lower and upper bounds for the starting time of a task and generates a Critical Path (CP) on this basis. It follows Min-Min algorithm strategy as a task is assigned to a processor that executes it fastest. The main focus here is to minimize the CP length.

3.4. Heterogeneous Earliest Finish Time

HEFT is one of the most well-known algorithms for the scheduling of task graphs. Here, priorities are assigned to the tasks on the basis of their upward ranks which are based on average execution times of tasks and average communication costs among the processors of two successive tasks. A processor which gives minimum finish time for a task is selected for its execution.

3.5. Critical Path on a Processor

CPOP uses downward ranks along with upward ranks for the task prioritization. Along with this, it uses the CP of a graph and a critical processor is used for mapping of tasks that lie on the CP. For remaining tasks, the processor which gives minimum finish time is selected for execution.

3.6. Performance Effective Task Scheduling

PETS algorithms has three phases. It performs level-wise sorting of the tasks before task prioritization and processor selection phases. In the first phase, independent tasks are grouped together so that they can be executed in parallel. Processor selection phase for PETS is same as for HEFT and CPOP.

In our proposed algorithms, the rank computation phase has been extended to include some more attributes that affect the overall makespan of the application. The modifications in the rank computation process result in obtaining efficiency in the scheduling phase of the algorithm. We have considered the average computation costs of the tasks, the communication costs that are required to transfer output of the tasks to their immediate successors and the received communication costs from the immediate predecessors of the tasks. Besides these, we have also considered the number of edges for each task i.e., number of tasks that are dependent on a particular task so that, a task with more number of dependent tasks may have higher rank. Detailed description of the proposed work is given below. HEFT, CPOP and PETS have been chosen for the purpose of comparison with the proposed algorithms.

4. Proposed Algorithms

4.1. HMCT

Heterogeneous Minimum Completion Time (HMCT) is the first proposed algorithm in the paper. HMCT has two main phases: Task prioritization and processor selection. Priorities of the task are computed on the basis of their upward and downward ranks. Two new factors, Output Transfer Cost (OTC) and Edge Percent (EP) have been introduced to compute the downward ranks along with the average computation cost of the tasks. OTC basically is the total cost that is required for transferring the output data from a parent task to all its immediate child tasks, i.e., OTC for a task t_i is the total cost required to transfer the output of t_i to its immediate successors. The second new factor used in the paper, EP, finds out the percentage of the tasks that are directly dependent on the output of a task t_i. By doing this, a task that has more dependent tasks will have higher rank and thus will be scheduled for execution earlier, hence opening way for the execution of dependent tasks.

The average computation cost of the task t_i, (Avg_CC (t_i)) on all the processors is computed as:

$$Avg\_CC_i(t_i) = \sum_{j=1}^{m} w_{i,j} / m \tag{6}$$

OTC for the end task is always 0; for the rest of the tasks, it is computed by the following equation:

$$OTC(t_i) = \sum_{j=1}^{x} C(i,j) \tag{7}$$

Where, t_i is the task whose OTC is to be computed and x is the number of its successor task. If there are total E edges in an application DAG then $EP(t_i)$ of a task t_i that has total x dependent tasks, is the percentage of edges t_i has from total E edges. Like OTC, EP for the end task is 0 and for remaining tasks it is calculated using the following relation:

$$EP(t_i) = \frac{x}{E} \times 100 \tag{8}$$

Downward rank ($rank\_d$) of a task t_i is finally, computed by combining all the three factors as shown in the following relation.

$$rank\_d(t_i) = Avg\_CC(t_i) + OTC(t_i) + EP(t_i) \tag{9}$$

For the upward ranks, first of all Received Cost (RC) is computed. RC is calculated using the following relation.

$$RC(t_i) = max(Avg\_CC(p_i) + C(p_{i,j})) \tag{10}$$

Where, p_i is the set of parents of t_i. Finally, the upward ranks ($rank\_u$) are computed as:

$$rank\_u(t_i) = RC(t_i) + max(rank\_u(succ(t_i))) \tag{11}$$

Where, $succ(t_i)$ is the set of immediate successors of t_i. Total rank ($rank$) for a task t_i is the sum of its upward and downward ranks, i.e.

$$rank(t_i) = rank\_d(t_i) + rank\_u(t_i) \tag{12}$$

In the second phase, appropriate processor for the execution of a task is to be selected. EST and ECT values for all the tasks are computed on all the available set of processors using the Equations 3 and 4. The selected processor is the one which gives the least value for ECT of the task. Like HEFT, the proposed Algorithms work on the insertion based policy, according to which, a task can be scheduled between two already scheduled tasks if there is an idle time slot available, provided the priority constraints are not violated.

4.2. Level-wise Prioritization and Scheduling

Level-wise Prioritization and Scheduling (LPS) approach, second technique proposed in the paper, has an additional phase, level wise sorting of the tasks, besides the task prioritization and processor selection phases. In the level wise sorting phase, DAG is traversed in a way such that independent tasks at each level are grouped together so that they can run in parallel. Task prioritization and processor selection phases are same as for HMCT. The pseudo codes for both the algorithms HMCT and LPS have been summarized in Algorithms 1 and 2 respectively. Both the proposed algorithms are explained with the example DAG of Figure 1. Table 2 shows computed *Avg_CC,* OTC, RC, EP and *rank* value for all the tasks. According to the *rank*, the priority list for HMCT is t_1, t_2, t_4, t_3, t_5, t_6, t_9, t_8, t_7 and t_{10}. LPS priority list, on the other hand is, t_1, t_2, t_4, t_3, t_6, t_5, t_9, t_8, t_7 and t_{10}. ECT and EST for the tasks on all the processors are given in Table 3 for HMCT Algorithm. These attributes can be computed for LPS Algorithms in the same way. For this particular example, the schedule length obtained through both the proposed algorithms is same as shown in Figure 2, but this certainly is not the case for each generated DAG. HMCT and LPS generate schedule length of 154 while the results of HEFT, CPOP and PETS are 208, 167 and181 respectively.

Algorithm 1: HMCT algorithm

```
Input: DAG, number of processors.
Output: Scheduled tasks, make span.

do (beginning from entry node of the DAG)
Compute the downward ranks (rank_d) for all the tasks by traversing the task graph downward.
do (beginning from end node of the DAG)
Compute the upward ranks (rank_u) for all the tasks by traversing the task graph upward.
Compute the overall ranks of the tasks by summing up the downward and upward ranks.
Set the priorities of the tasks according to the non-increasing order of their overall ranks.
while (there are tasks that have not yet been scheduled)
{
    a. Select the highest priority task.
    b. Compute ECT of the task on each processor by following
       the insertion based policy.
    c. Assign the task to the processor that gives least ECT.
}
```

Algorithm 2: LPS algorithm.

```
Input: DAG, number of processors.
Output: Scheduled tasks, make span.

Do (beginning from entry node of the DAG)
Compute the downward ranks (rank_d) for all the tasks by traversing the task graph downward.
Do (beginning from end node of the DAG)
Compute the upward ranks (rank_u) for all the tasks by traversing the task graph upward.
Compute the overall rank of the tasks by summing up the downward and upward ranks.
Beginning from the first level, set the priorities of the tasks according to the non-increasing order of their overall ranks on each level such that a task at higher level has higher rank.
While (there are tasks that have not yet been scheduled)
{
    a. Select the highest priority task.
    b. Compute ECT of the task on each processor by following
       the insertion based policy.
    c. Assign the task to the processor that gives least ECT.
}
```

Table 2. Computed attributes for HMCT and LPS.

Task	Avg_CC	OTC	RC	EP	Rank_D	Rank_U	Rank
1	11.33	169	0	25.00	205.33	279.67	485.00
2	18.00	122	92.33	12.50	152.50	279.67	432.17
3	28.33	51	66.33	12.50	91.83	242.00	333.83
4	45.33	132	13.33	18.75	196.08	204.00	400.08
5	10.33	43	42.33	6.25	59.58	218.00	277.58
6	18.33	76	96.33	6.25	100.58	190.67	291.25
7	12.33	35	81.33	6.25	53.58	175.67	229.25
8	25.00	19	93.00	6.25	50.25	187.33	237.58
9	36.00	43	64.33	6.25	85.25	158.67	243.92
10	15.33	0	94.33	0	15.33	94.33	109.67

Table 3. Computed EST and ECT on each processor and the selected processor for HMCT.

Tasks w.r.t Priorities	Processors						Selected Processor
	P1		P2		P3		
	EST	ECT	EST	ECT	EST	ECT	
1	0	12	0	13	0	9	P3
2	90	107	90	108	9	28	P3
4	11	60	11	55	28	71	P2
3	64	91	64	94	28	56	P3
6	106	124	55	73	106	125	P2
5	40	50	73	84	56	66	P1
9	92	127	92	126	56	95	P3
8	103	128	103	126	100	127	P2
7	91	101	126	139	95	109	P1
10	149	164	138	154	149	164	P2

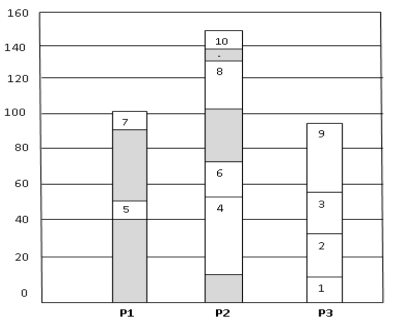


Figure 2. Schedule length for the HMCT and LPS.

5. Results and Discussion

In this section, the proposed techniques have been evaluated through their comparison with HEFT, CPOP and PETS. Random task graph generator function has been implemented through which DAGs with diverse attributes have been generated and then used for the experimental purposes. Moreover, evaluation of the proposed algorithms has also been made on the basis of task graphs from real world application.

5.1. Attributes of the Task Graph

The attributes of the DAGs depend on various input parameters such as, number of nodes in the graph (N), height/shape parameter of the graph (α), out degree of a node, communication to Computation Cost Ratio (CCR) and range percentage of computation cost (β). Different combinations of values (given below) have been selected in the DAG generation for the experimental purpose.

$N= \{20, 40, 60, 80, 100, 150, 200, 250, 300, 350, 400\}$
$\alpha = \{0.5, 1.0, 1.5, 2.0\}$
$CCR = \{3, 5, 7, 10, 15, 20, 25\}$
$Out\_degree = \{1, 2, 3, 4, 5\}$
$\beta = I0.25, 0.5, 0.75, 1.0I$

Height of a DAG is generated randomly from a uniform distribution whose mean is equal to *sqrt(n)/α* and its width from *sqrt(n)xα*. For small values of α, the generated DAG is longer and has low parallelism while a shorter DAG with high parallelism results if α is kept high. Heterogeneity among the processors is controlled through the range parameter β. A significant variation can be produced in the computation times of the tasks on different processors with large value of β.

5.2. Comparison Metrics

The comparison among the proposed techniques and the existing ones is made on the basis of different performance and cost metrics. A little description of these metrics is given below.

- *Make Span*: Is the main performance metric which gives the overall completion time for all the tasks in a given graph.
- *SLR*: As different task graphs with diverse attributes are generated and studied, SLR is computed in which schedule length is normalized to some lower bound. For an algorithm, SLR value is the ratio of its makespan and sum of minimum computation costs of tasks on the CP, i.e.

$$SLR = \frac{makespan}{min(CompCost\_on\_CP)} \tag{13}$$

- *Speedup*: Is the third metric used for the comparison purpose of the algorithms which is obtained by dividing the sequential execution times of the graphs by their parallel execution times.
- *Efficiency*: Is the ratio of speedup and the number of processors used.

5.3. Experimental Results

5.3.1. Randomly Generated Task Graphs

The quality of the algorithms with respect to different attributes of the graphs mentioned above is evaluated by generating diverse set of random task graphs and the results are shown below. From the results, it can apparently be seen that the performance of the proposed algorithms is considerably well as compared with the existing algorithms.

In Figure 3, average makespan for the proposed techniques and the existing ones is compared against different values of the shape parameter α. For each value of α, 100 task graphs were randomly generated and their makespans were obtained for the proposed and the existing algorithms selected here for the comparison purpose. The results show that the average makespan for both the proposed algorithms is less compared with HEFT, PETS and CPOP. For α=1, the percentage improvement in makespan of HMCT and LPS compared with HEFT is 8% and 10% respectively.

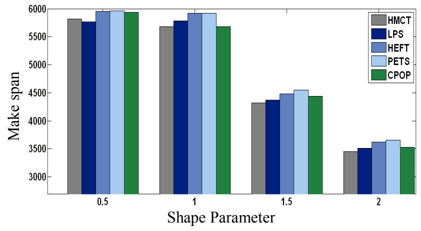


Figure 3. Average make span for varying α

Figure 4, 5 and 6 compare the algorithms for the average value of SLR obtained against varying α values, number of nodes and varying CCR values respectively. Again, 100 task graphs were generated for each case and the results were compared. The performance of an algorithm is considered best if it has smallest SLR value. On this basis, it can be seen from the Figures below that HMCT and LPS outperform the existing algorithms in most of the cases.

Figure 4 shows that the average SLR value increases when the parallelism among the DAGs is increased i.e., the value of α is increased. The performance ranking of the algorithms for longer DAGs (when α is 0.5 and 1.0) is HMCT, LPS, PETS, HEFT and CPOP. As the α reaches to 2.0, length of the generated DAG becomes much shorter and the performance of CPOP becomes comparable with HMCT and LPS.

As the number of nodes increases in Figure 5, the difference of average SLR value of the proposed algorithms with the existing ones increases which shows that the performance of HMCT and LPS for large applications with more number of tasks is better as compared to small applications. The maximum percentage improvement of the proposed algorithms in this scenario is approximately 9%.

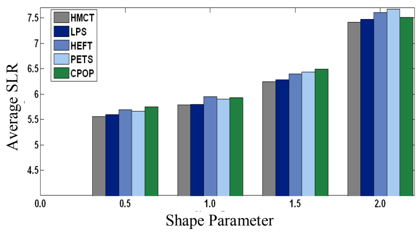


Figure 4. Average SLR for varying α.

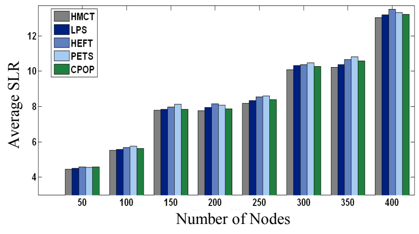


Figure 5. Average SLR for varying no of nodes.

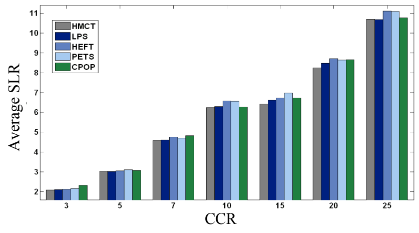


Figure 6. Average SLR for varying CCR.

Figure 6 shows the cases for CCR values of 3, 5, 7, 10, 15, 20 and 25. Average value of SLR shows an increasing trend with the increase in value of CCR. For small values of CCR (3, 5), there is slight difference in the values of HMCT, LPS and HEFT. But as the CCR value increases, the performance of HMCT and LPS improves quite well. When CCR is larger, communication costs received and transferred by a task dominate the rank assignment function while for small CCR values, scheduling list order is not affected as dominating factor here is average computation cost. The proposed algorithms show approximately 8% improvement in the results compared with the existing well known algorithms.

Speedup and efficiency comparison of the Algorithms for varying number of nodes and number of processors respectively is also made and given in the Figures 7 and 8, respectively. It is clear from Figure 7 that there is great improvement in the speedup obtained through HMCT and LPS compared with HEFT, PETS and CPOP. This improvement rises as the number of nodes are increased which shows that our proposed Algorithms outperform the other reported algorithms. The maximum speedup improvement of HMCT and LPS here is almost 10%.

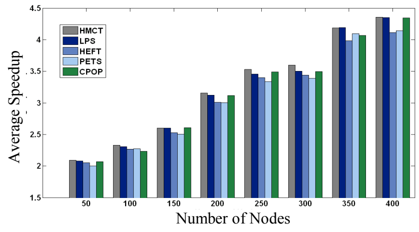


Figure 7. Average speedup comparison.

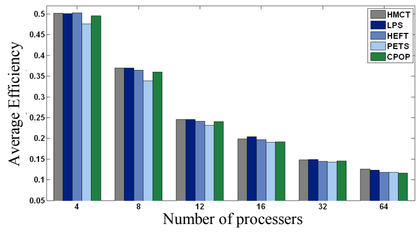


Figure 8. Average efficiency comparison.

5.3.2. Task Graphs of Real World Problems

In another experiment, task graph of a real world problem, molecular dynamics code [11], has been taken. The task graph is an irregular one. The number of tasks in the graph and the structure of the application are defined already, only the CCR values have been used to evaluate the quality of the proposed algorithms with respect to average SLR. Figure 9 shows that the proposed algorithms outperform the existing ones.

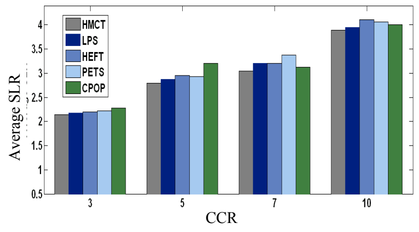


Figure 9. Average SLR for varying CCR.

6. Conclusions

Optimal mapping of the tasks in an HDCS requires efficient strategies for task scheduling problem in order to obtain near optimal results. In this paper, two new task scheduling algorithms have been proposed and extensively been tested against different comparison metrics and compared with the existing well known algorithms. Diverse set of task graphs with varying characteristics have randomly been generated and used for the comparison of the existing and proposed algorithms. The results of the comparative analysis are evident of the fact that the proposed algorithms perform significantly better in most of the cases.

References

[1] Bajaj R. and Agrawal D., "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Transaction on Parallel and Distributed Systems*, vol. 15, no. 2, pp. 107-118, 2004.

[2] Basker S. and SaiRanga P., "Scheduling Directed A-cyclic Task Graphs on Heterogeneous Network of Workstations to Minimize Schedule Length," *in Proceedings of International Conference on Parallel Processing Workshops*, pp. 97-103, 2003.

[3] Dhodhi K., Ahmad I., and Yatama A., "An Integrated Technique for Task Matching and Scheduling onto Distributed Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1338-1361, 2004.

[4] Dogan A. and Ozguner F., "LDBS: Duplication based Scheduling Algorithm for Heterogeneous Computing Systems," *in Proceedings of International Conference on Parallel Processing*, pp. 352-359, 2002.

[5] Ijaz S., Munir E., Nisar W., and Anwar W., "Efficient Scheduling Strategy for Task Graphs in Heterogeneous Computing Environment," *the International Arab Journal of Information Technology*, vol. 10, no. 5, pp. 486-492, 2013.

[6] Ilavarasan E., Thambidurai P., and Mahilmannan R., "Performance Effective Task Scheduling Algorithm for Heterogeneous Computing System," *in Proceedings of the 4th International Symposium on Parallel and Distributed Computing*, Lille, pp. 28-38, 2005.

[7] Iverson M., Ozguner F., and Follen G., "Parallelizing Existing Applications in a Distributed Heterogeneous Environment" *in Proceedings of 4th Heterogeneous Computing Workshop*, pp. 93-100, 1995.

[8] Kafil M. and Ahmed I., "Optimal Task Assignment in Heterogeneous Distributed Computing Systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42-51, 1998.

[9] Kim S., "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures," available at: ftp://ftp.cs.utexas.edu/pub/ techreports/ tr88-04. pdf, last visited 1988.

[10] Mahamat H. and Azween A., "A New Grid Resource Discovery Framework," *the International Arab Journal of Information Technology*, vol. 8, no. 1, pp. 99-107, 2011.

[11] Munir E., Mohsin S., Hussain A., Nisar M., and Ali S., "SDBATS: A Novel Algorithm for Task Scheduling in Heterogeneous Computing Systems," *in Proceedings of the 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum*, Cambridge, USA, pp. 43-53, 2013.

[12] Rahman M., Venugopal S., and Buyya R., "A Dynamic Critical Path Algorithm for scheduling Scientific Workflow Applications on Global Grids," *in Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing*, Bangalore, pp. 35-42, 2007.

[13] Topcuglou H., Hariri S., and Wu M., "Performance Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transaction on Parallel and Distributed Systems,* vol. 13, no. 3, pp. 260-274, 2002.

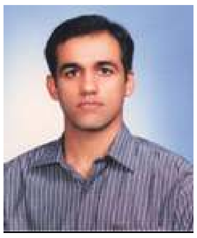
[14] Wu M. and Gajski D., "Hypertool: A Programming Aid for Message Passing Systems," *IEEE Transaction on Parallel and Distributed Systems,* vol. 1, no. 3, pp. 330-343 1990.

Ehsan Munir received his PhD degree in computer software and theory from Harbin Institute of Technology Harbin, China in 2008. He completed his MS degree in computer science from Barani Institute of Information Technology, Pakistan in 2001. Currently, he is an associate professor and head in the Department of Computer Science at COMSATS Institute of Information Technology, Pakistan. His research interests are task scheduling algorithms in heterogeneous parallel and distributed computing.

Samia Ijaz received her BS degree in computer science and Engineering from University of Engineering and Technology, Pakistan, in 2005. She has recently completed her MS in computer sciences from COMSATS Institute of Information Technology, Pakistan, in 2011. Her research interests include computer networks, task scheduling algorithms in heterogeneous computing and image processing.

Sheraz Anjum received PhD degree in Engineering microelectronics and solid-state electronics from Institute of Microelectronics, Graduate University of Chinese Academy of Sciences, China, in 2008, the degree of MSc computer engineering from University of Engineering and Technology, Pakistan, in 2005 and MSc in electronics from Quaid-E-Azam University, Pakistan, in 1999. Currently he is working as Associate Professor at the Department of Electrical Engineering, COMSATS Institute of Information Technology, pakistan. His research interests include but not limited to digital system design, design and analysis of networks on chip architectures and algorithms, reconfigurable architectures, multi-processor heterogeneous computing and design of advance DSP architecture.

Ali Khan completed his BS degree in electrical engineering from University of Engineering & Technology, Pakistan in 2003 and PhD degree in information and communication engineering from Harbin Institute of Technology, Harbin, China (PRC) in 2008. He is an Assistant Professor in Electrical Engineering Department at COMSATS Institute of Information Technology, Pakistan. he is also Head of Wireless Sensor Networks Research Group and supervising graduate and post graduate research in the areas of mobile ad-hoc networks, distributed computing, WSN applications, MAC layer issues and mote design.



Waqas Anwar received his PhD degree in computer science from Harbin Institute of Technology Harbin, China in 2008. He is currently an assistant professor in the Department of Computer Science at COMSATS Institute of Information Technology, Pakistan. His areas of research are NLP and computational intelligence.

Wasif Nisar received his PhD degree candidate in computer science from Institute of Software, GUCAS China in 2008. He received his BSc degree in 1998 and MSc degree computer science in 2000 from University of Peshawar, Pakistan. His research interest includes software estimation, software process improvement, distributed systems, databases and CMMI-based project management.