Efficient Scheduling Strategy for Task Graphs in Heterogeneous Computing Environment

Samia Ijaz[1], Ehsan Ullah Munir[1], Waqas Anwar[2], and Wasif Nasir[1]
[1]Department of Computer Science, COMSATS Institute of Information Technology, WahCantt, Pakistan
[2]Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan

Abstract: *Today's multi-computer systems are heterogeneous in nature, i.e., the machines they are composed of, have varying processing capabilities and are interconnected through high speed networks, thus, making them suitable for performing diverse set of computing-intensive applications. In order to exploit the high performance of such a distributed system, efficient mapping of the tasks on available machines is necessary. This is an active research topic and different strategies have been adopted in literature for the mapping problem. A novel approach has been introduced in the paper for the efficient mapping of the DAG-based applications. The approach that takes into account the lower and upper bounds for the start time of the tasks. The algorithm is based on list scheduling approach and has been compared with the well known list scheduling algorithms existing in the literature. The comparison results for the randomly synthesized graphs as well as the graphs from the real world elucidate that the proposed algorithm significantly outperforms the existing ones on the basis of different cost and performance metrics.*

Keywords: *Directed acyclic graphs, task scheduling, task prioritization, makespan.*

Received August 9, 2011; accepted December 30, 2011; published online August 5, 2012

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 [1]. In order to fully exploit such Heterogeneous Distributed Computing Systems (HDCS), applications running on them are decomposed into different number of tasks that may have or have no dependencies among themselves [11].

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 a very critical aspect. Mapping of tasks to machines should be done so as to lessen the overall time for the execution of the application. The scheduling problem becomes more complex in a HDCS due to the multiformity of not only the resources on which tasks are to be executed but also due to varying speeds of intercommunicating links between them as time required for the single task execution on different resources or for transferring same amount of data will be different. Generally, the task scheduling problem is modeled by Directed Acyclic Graph (DAG) in which application tasks are shown through the graph nodes and dependencies for the data among different tasks are depicted through edges. Communication costs are marked on the edges and computation costs are labeled on the nodes. The task scheduling problem addressed here is a static model as different properties of the application for example, execution times of all the tasks on distinct resources and inter-task communication costs are known in advance.

To achieve promising results from the multifarious distributed resources and eminent speed networks, efficient strategies for the scheduling of the applications have prime importance and therefore, this area is an active topic of research. Plethora of algorithms exists in the literature for solving the problem of task scheduling but, being *NP*-complete [4], finding near optimal solution for the problem requires more efficient scheduling strategies. High speedup and efficiency can be attained only if the mapping of tasks on machines is done appropriately as it can truly exploit system parallelism.

List scheduling strategy has been adopted in the research work proposed in the paper. Proposed work has been compared with the work in [14, 15] in terms of different performance and speed metrics such as efficiency, speedup, schedule length ratio and makespan. The results obtained through extensive simulation analysis affirm that the proposed algorithm surpasses the previous ones quite significantly.

The paper is organized as follow: The task scheduling problem has been addressed in section 2. Section 3 gives a brief review of the related work in the field. Proposed algorithm is demonstrated in section 4 with experimental investigation being shown in section 5. Finally, section 6 concludes the paper.

2. Task Scheduling Problem

The basic elements of a scheduling system are an application program, certain environment to run the application on and some strategy for the scheduling of the application. Generally, a DAG is used for this purpose, with a set V of nodes representing n number of tasks of the application and a set E of edges showing 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 the situation is different and both are being executed on different processors then the communication cost, $C_{i,j,}$ between the two dependent tasks is computed using the following relation:

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

For the sake of simplicity, data transfer cost is assumed to be 1.0 in this case. A task graph of Figure 1 has been chosen for the thorough elaboration of the proposed algorithm. Computation cost matrix is given in Table 1.

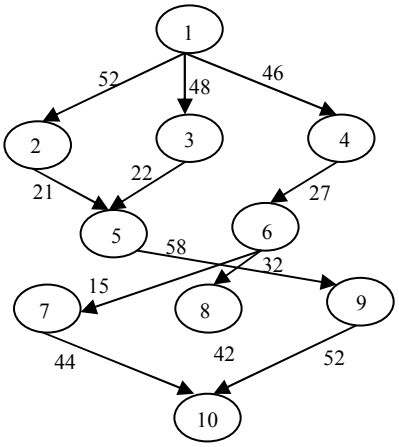

Figure 1. Directed acyclic graph.

Table 1. Computation cost matrix.

Task	P1	P2	P3
1	17	19	8
2	7	6	7
3	21	29	14
4	8	10	11
5	33	30	50
6	11	13	11
7	11	6	15
8	12	20	14
9	26	31	27
10	13	12	13

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 figured out using the equations 3 and 4. An important consideration here is that a task can be scheduled to execute only if the execution of its parent tasks has finished and once a task has been executed, *EST* and *ECT* values become the *AST* (actual start time) and *ACT* (actual completion time) respectively for the task.

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

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

where t_p belongs to the immediate predecessor set of task t_i, the time when processor p_j will be free from the execution of already scheduled tasks and is available for the execution of the task t_i is depicted by *avail[j]* and $C_{i,j}$ is the communication cost needed for transferring the output of the parent task to its currently executing child task (t_i in this case). The internal max block in equation 3 is returning the time when all the necessary information for t_i has reached the processor p_j. When *EST* value for t_i on p_j has been computed, the earliest time by which the computation of t_i can be completed can be found out using equation 4. Here, the computation time of t_i ($W_{i,j}$) is added up in its earliest start time on p_j ($EST(t_i, p_j)$) to get $ECT(t_i, p_j)$. Finally, after all the tasks are mapped on appropriate processors, the actual completion time for the end task gives the schedule length (or the makespan) for the entire application, i.e.

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

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

3. Related Work

High performance of the HDCS demands for the efficient scheduling strategies for an application. Because of its fundamental significance, extensive study has been made in the field and bunch of algorithms exist in the literature. The classification of the algorithms has been done as: list scheduling algorithms [7, 8, 12, 14], guided random algorithms [5], cluster based [9] and task duplication algorithms [2, 3, 6].

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 [14]. Few of the algorithms in this category that exist in literature are Modified Critical Path

(MCP) [16], Mapping Heuristics (MH) [7], Levelized Min Time (LMT) [8], Dynamic Critical Path for Grids (DCP-G) [13], Heterogeneous Earliest Finish Time (HEFT) [14], Critical Path on a Processor (CPOP) [14] and Performance Effective Task Scheduling (PETS) [15]. A brief description of some algorithms is given below.

3.1. Mapping Heuristics (MH)

The computation costs for a task, in case of MH, is the ratio of the total count of instructions that have to run in the task and the processor speed. Static upward ranks are computed for the tasks, on the basis of which priorities are assigned. The main drawback associated with the MH algorithm is that it does not follow insertion based strategy.

3.2. Levelized Min Time (LMT)

The two phase algorithm initially performs task prioritization and then the selection of processor phase takes place. 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 allots the tasks to the quickest 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 (DCP-G)

This algorithm considers the lower and higher limits for the starting time of a task and generates a critical path on this basis. It follows Min-Min algorithm strategy as a task is allotted to a processor that finishes its execution fastest. The emphasis here is to minimize the critical path length.

3.4. Heterogeneous Earliest Finish Time (HEFT)

One of the most well-known scheduling algorithms, HEFT, prioritizes the tasks considering their upward ranks which are computed using average of execution times of tasks and mean communication costs among the processors of two successive tasks. A processor that yields minimum finish time for a task is chosen for its execution.

3.5. Critical Path on a Processor (CPOP)

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

3.6. Performance Effective Task Scheduling (PETS)

PETS algorithms has three phases. It performs level-wise sorting of the tasks before task prioritization and processor selection phases. Firstly, grouping of independent tasks is done in a way that their concurrent execution can be performed. Processor selection phase for PETS is same as for HEFT and CPOP.

4. Proposed Work

4.1. Minimal Latest Start Time (MLST)

MLST is the proposed algorithm in the paper. The time limits for beginning the scheduling of a task are taken into account. The objective behind the approach is that a task whose latest time to start scheduling has approached must be put for scheduling. Delaying the execution of the task even if its start time has reached its maximum limit will result in an increase in the execution time of the entire application. The time limits have been considered for scheduling in the earlier proposed algorithms in the literature such as in [13, 16, 17], but the novelty introduced in the paper is that an extra phase has been adopted that performs level-wise task sorting. The algorithm has also task prioritization and processor selection phases. For assigning priorities, the lower and upper bounds for the start time of a task have been considered and a task whose latest start time among all the tasks is minimal is assigned a higher rank. The prioritization of the tasks is done level wise such that a task at a lower level has higher rank than a task at a higher level. DAG is traversed in such a way that independent tasks at each level are grouped in a way that their concurrent execution is possible. Absolute Earliest Start Time (AEST) of the tasks is computed recursively starting from the entry node [17]. AEST for the entry node is 0 and for the other tasks, it is figured out using the following relation,

$$AEST(t_i) = max_{t_p \in pred(t_i)}\{AEST_{t_p} + w_{t_p} + C_{t_p,t_i}\} \quad (6)$$

where t_p is the immediate predecessor of the task t_i.

When AEST values for all the tasks are known, Absolute Latest Start Time (ALST) for each task is computed recursively by traversing the DAG upward. Equations 6 and 7 are used for the calculation of ALST.

$$ALST_{t_{end}} = AEST_{t_{end}} \quad (7)$$

$$ALST_{t_i} = min_{t_s \in succ(t_i)}\{ALST_{t_s} - C_{i,s}\} - w_{t_i} \quad (8)$$

where t_s is the set of immediate successors of the task t_i.

Finally, the prioritization of the tasks is done level wise on the basis of ALST of the tasks. A task that has minimal ALST among all the tasks at a level has higher priority (rank). For the processor selection phase, EST and ECT for a task on each processor is reckoned using the equations 2, 3 and 4 and the processor which gives the least ECT for a task is assigned the task for execution. The MLST works on insertion based approach in which scheduling of a task is allowed between already scheduled tasks if there is an idle time slot available, provided the priority constraints are not violated.

Table 2. Computed attributes for MLST.

Tasks w.r.t Priority	Processors						Final Processor
	P1		P2		P3		
	EST	ECT	EST	EST	ECT	EST	
1	0	17	0	19	0	8	P3
3	26	67	26	75	8	32	P3
2	90	97	90	96	32	39	P3
4	74	77	74	79	39	45	P3
5	63	96	63	93	45	95	P2
6	99	110	99	112	45	56	P3
9	171	177	93	104	171	178	P2
8	118	130	118	138	56	70	P3
7	61	72	104	110	70	85	P1
10	176	179	116	118	176	179	P2

The summary of the proposed algorithm is given in Figure 2.

Input: DAG, Number of Processors
Output: Scheduled Tasks, Makespan
1. *Beginning from the entry node of the DAG, do*
   a. *Calculate the absolute earliest start time (AEST) for all the tasks by task graph traversal in top-down manner.*
2. *Beginning from the end node of the DAG, do*
   a. *Compute the absolute latest start time (ALST) for all the tasks by task graph traversal in bottom-up manner.*
3. *Beginning from the first level, set the priorities of the tasks in the non-increasing order of their ALST on every level such that a task at higher level has higher rank.*
4. *While there are tasks that have not been scheduled yet, do*
   a. *Select the highest priority task*
   b. *Compute ECT of the task on each processor by following the insertion based approach.*
   c. *Allot the task to the processor that gives least ECT*
   end
   end
end

Figure 2. MLST algorithms.

The algorithm has been explained through the DAG of Figure 1. There are 10 tasks in the graph and the tasks are to be executed on an appropriate processor from a set of 3 processors. Table 2 displays the attributes of the tasks required for the proposed algorithm such as average computation costs, AEST, ALST, ranks based on the ALST and the priorities of the tasks according to their rank value.

Finally, EST and ECT values for the tasks on every processor and the selected processor through MLST are displayed in Table 3. The schedule length for the selected DAG obtained when executed through MLST algorithm is 133 while HEFT, CPOP and PETS give the schedule lengths of 151, 149 and 169 respectively.

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

Task	Avg_CC	AEST	ALST	Rank	Task Priority
1	14.667	0	0	0	1
2	6.667	66.667	78.333	78.333	3
3	21.333	62.667	62.667	62.667	2
4	9.667	60.667	144.000	144.000	4
5	37.667	106.000	106.000	106.000	5
6	11.667	97.333	180.667	180.667	6
7	10.667	124.000	227.000	227.000	9
8	15.333	141.000	224.333	224.333	8
9	28.000	201.667	201.667	201.667	7
10	12.667	282.667	281.667	281.667	10

5. Results and Discussion

In this section, the proposed technique has been evaluated through its 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. Task graphs of real world application have also been considered for comparative analysis of the proposed and existing algorithms. Matlab simulator has been used for the comparative study of the proposed and existing work.

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), communication to Computation Cost Ratio (CCR), shape/height parameter of the graph (α), out degree of a node 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= {100, 150, 200, 250, 300, 350, 400}
No_of_Processors = {4, 8, 12, 16, 64}
α = {0.5, 1.0, 1.5}
CCR = {0.1, 1, 5, 10, 15, 20, 25, 30}
Out_degree = {1, 2, 3, 4, 5}
β = {0.25, 0.5, 0.75, 1.0}

Height of a DAG is generated from a uniform distribution randomly that has $\sqrt{n}/\alpha$ as mean and width from $\sqrt{n}\times\alpha$. 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 diverse 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:

- *Makespan:* Is the main performance metric which gives the overall completion time for all the tasks in a given graph.
- *Schedule Length Ratio (SLR):* As different task graphs with diverse attributes are generated and studied, schedule length ratio 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.

$$SLR = \frac{makespan}{min(CompCost\_on\_CP)} \quad (9)$$

- *Speedup:* Is the third metric employed for the evaluation purpose of the algorithms which is incurred by dividing the sequential execution times of the graphs by their parallel execution times.
- *Efficiency:* Is the speedup and makespan ratio of the graph.

5.3. Randomly Generated Task Graphs

The quality of the algorithm on the basis of different graph attributes described above has been evaluated by generating the randomly generated task graphs with diverse features and then executing through the proposed and existing algorithms. Comparison of the algorithms has been made and shown below. The results demonstrate that the existing algorithms (HEFT, CPOP and PETS) are outperformed by the MLST algorithm. For the experimental evaluation of the MLST algorithm, 700 graphs with diverse attributes have been produced. In 63% cases, the MLST algorithm surpasses the HEFT algorithm, in 65% cases PETS algorithms is outperformed and 70% scenarios show better results compared with CPOP.

The algorithms have been compared on the basis of makespan they produce when task graphs with different shapes are executed through them. Longer graphs with low parallelism to smaller graphs with high parallelism were generated through random graph generator. For each value of shape parameter (α), 100 task graphs were generated. The comparison has been displayed in Figure 3 which shows that the MLST algorithm completely outperforms HEFT, CPOP and PETS especially for longer graphs which have low parallelism.

The proposed algorithm has also been analyzed by average SLR produced for different graph structures and the results are shown in Figure 4. Again, 100 graphs were generated for each value of the shape parameter (α). The results affirm the fact that MLST performs significantly well compared with HEFT, CPOP and PETS.

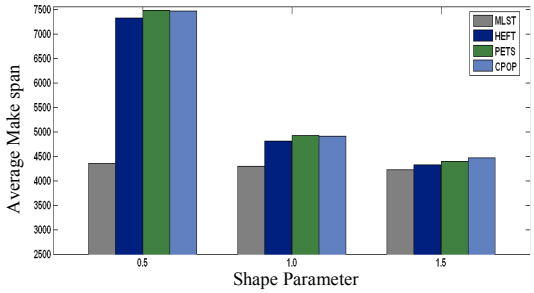

Figure 3. Average makespan for varying α.

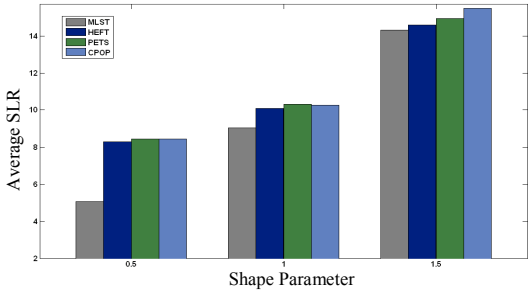

Figure 4. Average SLR for varying α

In another experiment, algorithm quality has been verified by changing the number of nodes and taking the average SLR of the schedule for 100 task graphs for each value of number of nodes. The experiments have been conducted for 100, 150, 200, 250, 300, 350 and 400 number of nodes. Figure 5 shows that as the number of nodes rises, the quality of the algorithm compared with the reported algorithms also improves. Similar experiments have been conducted for average SLR when the CCR is changed and selected from the range of values given in the above section. For computation-intensive graphs, average SLR produced by MLST is comparable with HEFT but for communication-intensive graphs, i.e., the graphs for which CCR is high, MLST outperforms rest of the algorithms. The results are shown in Figure 6.

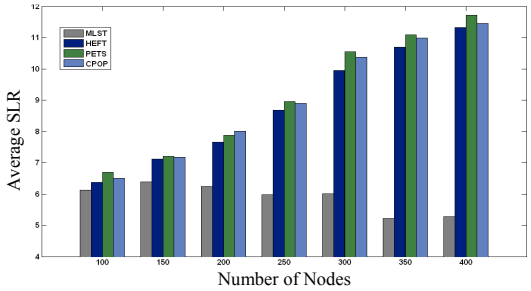

Figure 5. Average SLR for varying no of nodes.

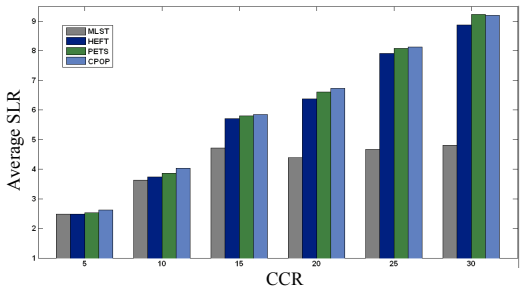

Figure 6. Average SLR for varying CCR.

Further, speedup and efficiency comparison of the algorithms on the basis of varying number of nodes and number of processors, respectively, have been made. Number of nodes is same as used for the average SLR comparison. The number of processors has been taken from 4, 8, 12, 16 and 64. The results of Figures 7 and 8 elucidate that MLST algorithm outperforms the other reported algorithms for average speedup and average efficiency.

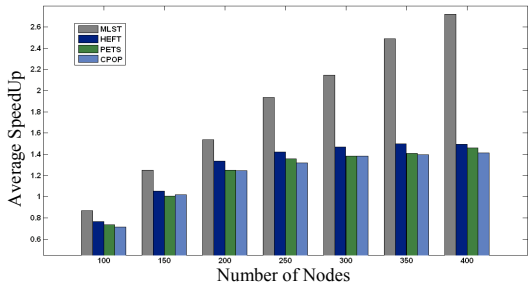

Figure 7. Average speed up comparison.

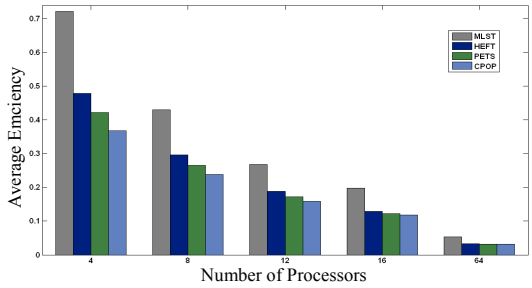

Figure 8. Average efficiency comparison.

5.4. Task Graphs of Real World Problems

In another experiment, task graph of a real world problem, molecular dynamics code [10], has been taken. The task graph is an irregular one. The number of tasks in the graph and the application structure are defined already, only the CCR values have been changed to evaluate the quality of the proposed algorithm with respect to average SLR. The performance of the algorithm has been displayed in Figure 9 which explains that the MLST algorithm outperforms the other reported algorithms significantly well. Again, graphs with diverse features have been generated and the MLST algorithm outperforms the HEFT and PETS in around 60% cases and CPOP in 70% cases.

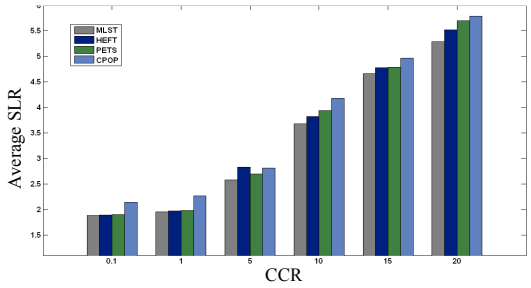

Figure 9. Average SLR comparison for molecular dynamics task graph.

6. Conclusions

In order to obtain the near optimal results for the problem of task scheduling in an HDCS, efficient strategies for the optimal mapping of the tasks are required. A novel task scheduling algorithm has been introduced in the paper and extensively been tested for different comparison metrics. The comparison of the algorithms has been made against the well known existing algorithms in the literature on the basis of these cost and performance metrics. Diverse set of task graphs with varying features have randomly been generated and used for the experimental purpose along with the task graph of a real world application. The comparative analysis explains that the performance of the proposed algorithm is significantly well in most of the cases.

References

[1] Braun S., Siegel D., Maciejewski J., Beck A., Boloni N., Maheswaran L., Reuther M., Robertson I., Theys P., and Yao D., "Characterizing Source Allocation Heuristics for Heterogeneous Computing Systems," *Advances in Computers*, vol. 63, pp. 91-128, 2005.

[2] Basker S. and SaiRanga C., "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] Bajaj R. and Agrawal P., "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Transaction on Parallel and Distributed System*, vol. 15, no. 2, pp. 107-118, 1998.

[4] Daoud I. and Kharma N., "A High Performance Algorithm for Static Task Scheduling in Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 4, pp. 399-409, 2008.

[5] 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, 2002.

[6] 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.

[7] EI-Rewini H. and Lewis G., "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," *Journal of Parallel and Distributed Computing*, vol. 9, no. 2, pp. 138-153, 1990.

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

[9] Kafil M. and Ahmed I., "Optimal Task Assignment in Heterogeneous Distributed Computing Systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42-50, 1998.

[10] Kim J. and Browne C., "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures," *in Proceedings of International Conference on Parallel Processing*, pp. 1-8, 1988.

[11] Munir U., Li Z., Shi S., Zou Z., and Rasool Q., "A New Heuristic for Task Scheduling in Heterogeneous Computing Environment," *Journal of Zhejiang University Science A*, vol. 9, no. 12, pp. 1715-1723, 2008.

[12] 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.

[13] 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.

[14] Topcuglou H., Hariri S., and Wu Y., "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.

[15] 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. 28-39, 2005.

[16] 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.

[17] Yang H., Lee P., and Chung C., "Improving Static Task Scheduling in Heterogeneous and Homogeneous Computing Systems," *in Proceedings of IEEE International Conference on Parallel Processing*, Xi'an, pp. 45-45, 2007.

Samia Ijaz received her BSc degree in computer science and engineering from University of Engineering and Technology, Lahore, Pakistan, in 2005. She has recently completed her MSc 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.

Ehsan Ullah Munir received his PhD degree in computer software and theory from Harbin Institute of Technology Harbin, China in 2008. He completed his MSc 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 include task scheduling algorithms in heterogeneous parallel and distributed computing.

Waqas Anwar received his PhD degree in computer science from Harbin Institute of technology Harbin, China in 2008. Currently, he is an associate professor in the Department of Computer Science at COMSATS Institute of Information Technology, Abbottabad, Pakistan. His research interests include 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 and MSc degrees in computer science from University of Peshawar, Pakistan, in 1998 and 2000, respectively. His research interest includes software estimation, software process improvement, distributed systems, databases, and CMMI-based project management.