HETS: Heterogeneous Edge and Task Scheduling Algorithm for Heterogeneous Computing Systems

Anum Masood*, Ehsan Ullah Munir*, M. Mustafa Rafique‡, Samee U. Khan§
*Department of Computer Science, COMSATS Institute of Information Technology, Wah Cantt - Pakistan
‡IBM Research - Ireland
§Department of Electrical and Computer Engineering, North Dakota State University, USA
anum.msd@gmail.com; ehsanmunir@comsats.edu.pk; mustafa.rafique@ie.ibm.com; samee.khan@ndsu.edu

***Abstract*—Widely used computing systems are heterogeneous in nature, comprising of interconnected resources which differ in computational capability of processing nodes and network bandwidth. Due to this diversity, an efficient heuristic is required to achieve high performance in heterogeneous computing system. In our proposed scheduling algorithm, Heterogeneous Edge and Task Scheduling (HETS), we schedule the communication between the tasks of application graph onto the network links of varying bandwidth, and schedule these tasks of different computation on the network processors after considering the computational capability of the available processors. In HETS, the prioritization is done by calculating the edge priority as well as the node priority. HETS algorithm selects the task after all its incoming edges are scheduled. The proposed algorithm minimizes the communication overhead of the application graph edges and obtains reduced schedule length in terms of the overall execution time. Performance of the proposed algorithm is studied by varying parameters of the standard task graphs as well as on real world directed acyclic graphs (DAGs) application, such as, CyberShake, Gaussian Elimination, and Montage. Extensive simulation results show the effectiveness of HETS algorithm in terms of reduced makespan and improved Schedule Length Ratio (SLR) for the given tasks.**

***Index Terms*—Bandwidth-aware scheduling, heterogeneous computing systems, task scheduling, Directed Acyclic Graph.**

I. Introduction

Heterogeneous computing systems are frequently being used for solving compute intensive applications that are sub-divided into different tasks. Generally these compute-intensive applications are represented using directed acyclic graph (DAG). Task scheduling heuristics are important for heterogeneous computing systems (HCS) as they provide high performance on these HCS by assigning the tasks to available resources with maximum computational speed.

Efficient scheduling of tasks is critical to enhance the performance of a computing system such that the overall execution time for all the scheduled task is minimized. Recently developed computer systems are different both in terms of architecture design and performance as compared to the traditional systems. Therefore the previous techniques are not fully applicable to these recent computer systems.

In scheduling, a directed graph has to be mapped onto the processors. Within an application task graph not all tasks are independent. There is precedence constraint among the dependent tasks. Typically, nodes of the graph represent tasks of a given application while the edges show the communication or transfer of data among those nodes, which have inter-task dependencies. The assignment of these tasks on the given machines or processors is known as task scheduling. Task scheduling is a NP-complete problem [18].

For task scheduling most commonly used algorithms are graph theoretic algorithms. The inputs required by the graph theoretic algorithms are application graph explained in terms of task computation time, communication cost, and precedence constraint. These algorithms are used for distributing the application graphs into tasks and allocating each of these tasks to the suitable execution nodes [7]. Numerous task scheduling heuristics were proposed on the assumptions such as tasks are independent and that there is no communication delay. These assumptions are unrealistic as there is always communication delay in parallel systems and the tasks of an application graph are mostly dependent [11].

Various heuristics are formulated for optimizing the solution for task scheduling problem. The tasks are scheduled in order to reduce the idle time on the machines and communication overhead. The scheduling of a DAG on the topological network not only involves the mapping of task nodes on the processor but also includes the mapping of the edges on the links of the network. When all the inputs are available, i.e., when parent nodes have successfully executed, only then the task is executed. The tasks are executed without being interrupted once they start. Each node has its own associated computation cost that designates the execution time of every node on the processor. In the case of homogeneous processor, the cost is the same for every type of processor.

The heuristics centered on list scheduling are used in DAG to assign priorities to tasks and for listing these tasks according to priorities in a descending order. The task of high priority is given preference over the one having low priority by the help of priority policy. A task can be assigned to any of the processors meanwhile a ready list is maintained for assigning the priorities.

There are various list scheduling algorithms proposed for finding the optimal schedule in heterogeneous computing systems [5], [6], [19]. Heterogeneous Earliest Finish Time (HEFT) and Critical Path On a Processor (CPOP) algorithms [21]; both of these are well-known list scheduling heuristics

for heterogeneous computing environment [20]. In HEFT, the upward rank of the task is calculated recursively to set the priority of the tasks. However in CPOP, the prioritization is done by adding the upward and downward ranks. Both of these algorithms use the insertion based approach.

Highest Level First with Estimated Times (HLFET) [3] allows the task with the earliest start time to be scheduled on the processor using Static Level for priority [12].

Dynamic Level Schedule (DLS) algorithm prioritize the tasks by using task's level state as well as considering their precedence constraint [16].

In Performance Effective Task Scheduling (PETS) heuristics, the communication cost is sub-divided into two parts, i.e., Data Receiving Cost (DRC) and Data Transfer Cost (DTC) [10]. SD-Based Algorithm for Task Scheduling (SDBATS) [14] considers the heterogeneity of the computing system. In SDBATS, the standard deviation of the communication cost of links between processor is considered. Performance Effective Genetic Algorithm (PEGA) uses genetic algorithm for providing the optimal schedule with low time complexity [1].

The proposed contention-aware algorithms are Bandwidth-Based Scheduling Algorithm (BBSA) and Optimal Insertion Hybrid Scheduling Algorithm (OIHSA) [9]. These algorithms solve the edge scheduling problem and optimal communication paths are selected, which provide the fastest communication for the given nodes.

This paper considers the heterogeneity of computing nodes and communication links by prioritizing edges and tasks. A task is mapped onto the processor once all the incoming edges are mapped and the data transfer from its precedence nodes is completed. The parameters, e.g., number of child node and execution time of node, are considered while selecting the processor for a given task.

The motivation behind this research work is to consider the heterogeneity not just in terms of nodes and processor but also in terms of edges between dependent nodes and the links interconnecting the processors. The proposed algorithm Heterogeneous Edge and Task Scheduling (HETS) shows effective results with minimum schedule length and overall high performance of heterogeneous computing systems. In Section 2, the problem statement is defined in the context of scheduling problem. Section 3 discusses the details of HETS algorithm. Section 4 provides results and discussion, and Section 5 concludes the paper with concluding remarks.

II. Problem Statement

The application which is to be modeled is in the form of DAG comprising of tasks which are also known as nodes of the graph. The set V represents the nodes of the graph which are shown as vertices in the graphical representation. The set E shows the communication among these nodes which are represented in the form of edges. An edge e_{u_i,v_j} belongs to the set E, while the set w_E shows the weight of the edges. The edge e_{u_i,v_j} is said to have weight $w(e_{u_i,v_j})$ that is also known as the communication cost. The weight of the nodes is represented by the set w_v. The weight of node u_i is $w(u_i)$, which is the computational cost of node u_i.

The problem comprises of two main steps, i.e., task scheduling and edge scheduling. As a pre-processing step, task priority and edge priorities are calculated.

A. Task Priority for Scheduling

The priority of the task is the sum of its computational cost and the communication cost of that node with its child nodes. Source node has highest priority while the sink node has the least $tpriority$ as it has no successor and thus no child.

$$tpriority(v_j) = cw(v_j) + avg\big(w(e_{n_i,v_j})\big) + max_{\forall n_i \epsilon succ(v_j)} tpriority(n_i) + c(v_j) \quad (1)$$

The computational cost of the given node v_j is given by $cw(v_j)$ and the communication cost is given by taking the average of all the edges communication cost having v_j as their source node $(w(e_{n_i,v_j}))$. The maximum of the task priority of the successor nodes of v_j is given by $max_{(\forall n_i \in succ(v_j))} tpriority(n_i)$. The total number of child is given by the $c(v_j)$. The density d of the network is the average value of neighbors per node (average number of child). The tasks are order in decreasing order of their $tpriority$.

B. Task Scheduling Algorithm

Once the task is selected its edges are to be mapped. A task is ready for execution only when all its edges are mapped on the links of the network. The Task Start Time (TST) is calculated by adding the Available Start Time (AST) with the Actual Transfer Time (ATT) on that particular processor node p_i. Available Start Time (AST) of any processor node is calculated by using Equation 2 and Equation 3. If there is no task mapped on the processor node before then the TST will be:

$$TST_{(u_i,P_i)} = 0 \quad (2)$$

If there is already task mapped on the processor node then the TST would be the Task Finish Time (TFT) of the task u_j on processor node P_i:

$$TST_{(u_j,P_i)} = max\Big(TFT_{(u_i,P_i)}, \big(max\big(ETT_{(e_{u_j,v_j})}\big)\big)\Big) \quad (3)$$

The condition to select the maximum of the finish time of task and its edges or the last tasks u_i finish time which was executed on the same node ensures that the insertion policy is taken into account while mapping the nodes. The Estimated Transfer Time (ETT) of the edge is added in the TST instead of the $w(e_{u_j,v_j})$ as the time of communication is change from communication time to the earliest transfer time by the factor of bandwidth. The Finish Execution Time (FET) for each processor node is the sum of the TST of the task u_j on processor node P_i:

$$FET_{(u_j,P_i)} = wc_{(u_j,P_i)} + TST_{(u_j,P_i)} \quad (4)$$

Task Finish Time (TFT) is calculated to select the processor node for the particular task node u_j and the TFT is the minimum of the FET on all the processor nodes of set P.

$$TFT_{(u_j)} = min\big(FET_{(u_j,P_i)}\big) \tag{5}$$

C. Edge Scheduling

The objective of the algorithm proposed in this paper is to reduce the delays and the contention among the links of the communication of the network. The contention cannot be reduced until the edge scheduling is coupled with the task scheduling. The communication cost of any edge is represented by $w(e(u_i, v_j))$. There will be an edge present between u_i and v_j only if u_i is the predecessor ofv_j. The task v_j will be mapped on the processor when all its predecessors have successfully sent data, i.e., the edges with the v_j as its destination node are executed, and the transfer of data is completed. The start time of task node v_j will be equal to or greater than the finish time of the last edge having v_j as destination node.

$$eTST(v_j) \geq max_{\forall u_i \epsilon pred(v_i)}\Big(LFT\big(e_{(u_i,v_i)}\big)\Big) \tag{6}$$

For each task the earliest task start time $eTST$ is calculated as well as the TST on each machine (processor). The maximum of the two $(eTST)$ and (TST) is selected for the calculation of FET on each machine (processor).

1) Priority for Edge Scheduling: If there are more than two parents of the given task then a priority rank is assigned to the particular edges on the basis of their average communication cost which is given in the set w. Therefore the edges priority is calculated as:

$$\begin{aligned} epriority\Big(e_{n_i,v_j}\Big) =& avg\Big(wc_{\forall L_i \epsilon L}\big(e_{n_i,v_j}, L_i\big)\Big) + \\ & max_{\forall n_i \epsilon pred(v_i)}\Big(ETT_{\left(e_{(u_i,v_i)},L_i\right)} + \\ & TFT(n_i)\Big) \end{aligned} \tag{7}$$

Where $e_{(u_i,v_i)}$ is the edge to be mapped and L_i is the link on which its priority is to be calculated. The edge with larger (ETT) and high (TFT) of the parent node will be scheduled first. This $epriority$ based scheduling shortens the TST of the child node.

2) Edge Scheduling Proposed Algorithm: Estimated transfer time is the time required to complete the communication between the parent node u_i and the child node v_j by mapping the edge $e_{(u_i,v_i)}$ on the set of links L containing the all of the links $L_i \in L$ using its respective heterogeneous bandwidth $B_i \in B$.

$$ETT_{(e_{u_i,v_j},L_i)} = \frac{wc\big(e_{u_i,v_j}, L_i\big)}{b_i} \tag{8}$$

If there is no edge mapped on the link before then the AST will be zero. If there is already an edge mapped on the particular link then AST would be the Link Finish Time (LFT) of the edge $e_{(u_i,v_i)}$ on link L_i:

$$AST_{(e_{u_i,v_j},L_i)} = LFT_{\left(e_{x_i,u_j},L_i\right)} \tag{9}$$

Finish Transfer Time (FTT) for each link is the sum of the AST of the link L_i with the ETT of the edge $e_{(u_i,v_i)}$ on the link L_i:

$$FTT_{(e_{u_i,v_j},L_i)} = AST_{(e_{u_i,v_j},L_i)} + ETT_{(e_{u_i,v_j},L_i)} \tag{10}$$

Actual Transfer Time (ATT) is calculated to select the link for the particular edge e_{u_i,v_j}.

$$ATT_{(e_{u_i,v_j})} = min\Big(FTT_{(e_{u_i,v_j},L_i)}\Big) \tag{11}$$

Link Finish Time (LFT) is sum of TFT of parent and ATT.

$$LFT_{(e_{u_i,v_j,L_i})} = ATT_{(e_{u_i,v_j})} + TFT_{u_i} \tag{12}$$

In some case the parent node is already finished by the time the link is available. In such case the condition applied is:

$$LFT_{(e_{u_i,v_j,L_i})} = \begin{cases} ETT_{(e_{u_i,v_j,L_i})} + \\ \quad TFT_{u_i} \ if \ LFT_{(e_{u_i,v_j,L_i})} < TFT_{u_i} \\ ATT_{(e_{u_i,v_j,L_i})} \ if \ LFT_{(e_{u_i,v_j,L_i})} \geq TFT_{u_i} \end{cases} \tag{13}$$

For mapping we need three things, the link on which the edge is to be selected, the link start time and link finish time. The link start time can be calculated by the equation below:

$$LST_{(e_{u_i,v_j},L_i)} = LFT_{(e_{u_i,v_j},L_i)} - ETT_{(e_{u_i,v_j},L_i)} \tag{14}$$

Pseudo-code of HETS is given in Algorithm 1. The algorithm performance is mainly calculated in terms of $makespan$. The formula for $makespan$ is:

$$makespan = max(TFT, LFT) \tag{15}$$

Time complexity of HETS in terms of t number of tasks and e number of edges is $O(t \times e)$.

Algorithm 1: HETS Algorithm.

Input: Given Application Graph $G(V, E)$ to be mapped on network topology.
Output: Scheduling of $G(V, E)$ on networks computing nodes P and communication links L.

```
// Prioritization of the tasks.
```
Sort nodes $u_i \in V$ into priority list in accordance to the $tpriority$ criteria. Arrange the nodes in non-increasing order of $tpriority$.
```
// Mapping of Edges and Tasks.
```
forall the *each* $u_j \in V$ *do* **do**
  forall the *each* $e_{(i,j)} \in E$ *do* **do**
    Calculate $epriority$ for all the edges having given node u_j as destination node. Arrange the edges in non-increasing order of $epriority$. By Calculating ETT of edge on each link L_iL considering the bandwidth B_iB of link L_i, LFT calculated for each edge $e_{(i,j)}$.
  end forall
  Schedule task $u_j \in V$ on the computing nodes $p_i \in P$ with minimum TFT.
end forall
Calculate $makespan$ for the mapping of G on network topology.

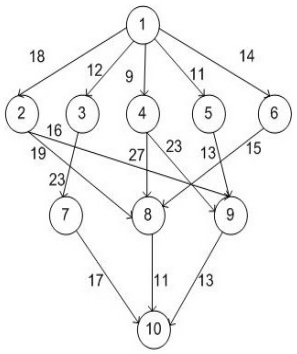

Fig. 1: DAG input graph

$Edges$	e_i	L_1	L_2	L_3
$e_{1,2}$	e_1	18	12	16
$e_{1,3}$	e_2	12	17	6
$e_{1,4}$	e_3	9	11	15
$e_{1,5}$	e_4	11	13	15
$e_{1,6}$	e_5	14	11	18
$e_{2,8}$	e_6	19	7	16
$e_{2,9}$	e_7	16	17	10
$e_{3,7}$	e_8	23	18	9
$e_{4,8}$	e_9	27	20	21
$e_{4,9}$	e_{10}	23	14	12
$e_{5,9}$	e_{11}	13	10	8
$e_{6,8}$	e_{12}	15	16	19
$e_{7,10}$	e_{13}	17	13	19
$e_{8,10}$	e_{14}	11	14	12
$e_{9,10}$	e_{15}	13	8	17

TABLE I: Communication Cost of Edges

$Nodes$	P_1	P_2	P_3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	16	15	11
8	5	11	14
9	18	12	20
10	23	7	21

TABLE II: Computational Cost of Nodes

$Nodes$	$tpriority$
1	113.000
3	84.000
4	84.000
2	81.000
5	73.000
6	67.333
7	46.667
8	38.667
9	46.333
10	16.667

TABLE III: Priority of Nodes

III. HETS Algorithm

The mapping of edges on the links of the computing systems and of vertices on the machines available is done in HETS algorithm. The HETS priority ranking is done by calculating $tpriority$ (using Equation 1) for each vertex and priority queue is generated. All the vertices are arranged in accordance with the non-increasing order of their $tpriority$. The $epriority$ calculation (using Equation 7) is the second part of the mapping phase. These edges are arranged in accordance with the non-increasing order if their $epriority$. The edge with high $epriority$ is to be assigned first. The edges are mapped on the links available such that the scheduling phase is repeated until all the nodes are mapped on the network topology.

LFT is calculated for each link afterwards TFT is calculated. The $makespan$ is calculated using maximum of all the edges LFT and vertices TFT. DAG input graph is shown in Figure 1. Table 1 and Table 2 show the computational and communication cost, respectively. Table 3 shows the calculated $tpriority$ and bandwidth of the links is given in Table 4. Figure 2 shows graphical representation mapping of the DAG input graph. The graphs shows the mapping of nodes on processors. The node $n1$ of given DAG is mapped onto processor $P3$ having Task Start Time ($TST_(n_1, P_3)$) value 0 and Task Finish Time ($TFT_{(n_1,P_1)}$) value 9. Edge $e2$ between the nodes $n1$ and $n3$ is mapped onto link $L3$ having Link Start Time ($LST_{(e2_{n_1,n_3},L_3)}$) value 9 and Link Finish Time ($LFT_{(e2_{n_1,n_3},L_3)}$) value 10.

$Link$	b_i	$Bandwidth$
L_1	b_1	2 GB/s
L_2	b_2	4 GB/s
L_3	b_3	6 GB/s

TABLE IV: Bandwidth of Links

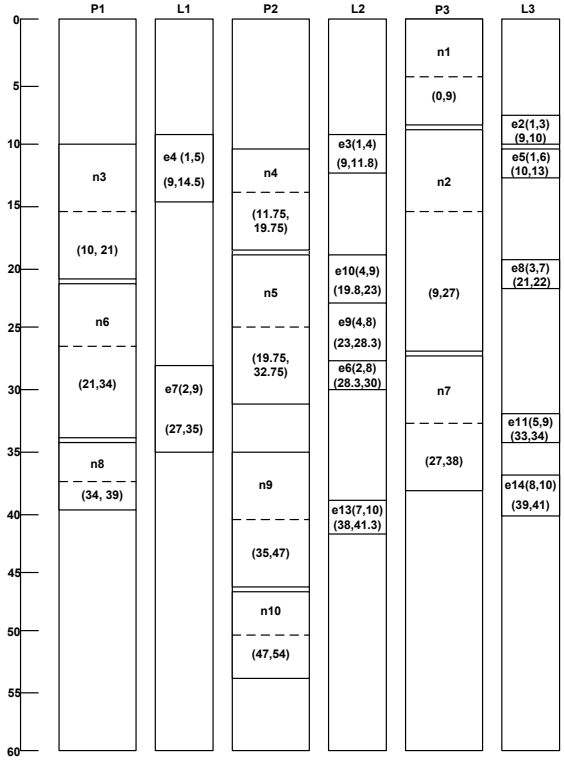

Fig. 2: Mapping using HETS (Makespan = 54)

IV. Results and Discussion

For evaluating HETS, extensive experimentation was carried out by using the randomly generated DAGs and real world application graphs, such as, Montage, Gaussian Elimination, and CyberShake. The sample graphs from Graph partitioning Archive [22] were also used for evaluation of the HETS algorithm. The communication and computation cost matrices for the given DAG were randomly generated by ETC generation method [14].

The ETC matrix takes into account both processors heterogeneity as well as the tasks heterogeneity. It mainly contains the execution times of all tasks on available processors. The communication cost matrix depicts the dependencies among the tasks of the application graph. The various parameters include number of processors, number of nodes, CCR (communication to computation ratio), which handles the execution time vales in ETC and the shape of the DAG is controlled by the alpha (α) parameter. Lesser the value of alpha, less will be the parallelism but if the value of alpha is unit, i.e., 1, then the DAG will be balanced. Most commonly used performance metric for the comparison of the heuristics effectiveness is the schedule length or makespan, which is the overall execution

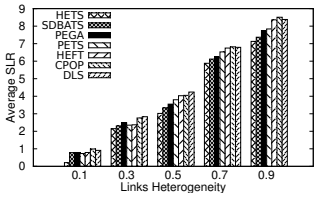

Fig. 3: DAG Input Graph average SLR for different link heterogeneity values.

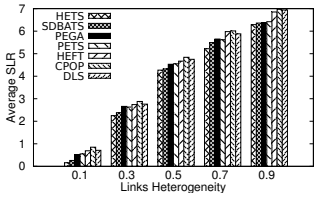

Fig. 4: Montage Graph average SLR for different link heterogeneity values.

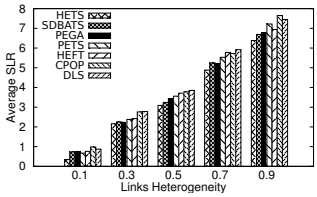

Fig. 5: CyberShake Graph average SLR for different link heterogeneity values.

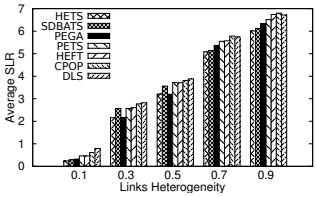

Fig. 6: Gaussian Elimination Graph average SLR for different link heterogeneity values.

time of application task graph.

Multiple scheduling algorithms, i.e., SDBTAS, PEGA, PETS, HEFT, CPOP, and DLS, were implemented in a simulation environment to evaluate the performance of HETS using real world application graphs such as Montage Graph, CyberShake Graph and Gaussian Graph. DAGs of varying size, from 5 to 250 nodes, were also used for testing of the proposed algorithm with the existing algorithms. Heterogeneity was considered in terms of task, processor, and links.

The total processors or machines available in network topology are given by P. The total number of nodes which are to be mapped onto the processors are N. The links are represented by L and the edges of the given application graph are represented by E. The throughput of the reconfiguration network in terms of communication and computational throughput workload is given by communication throughput t_{comm} and computational throughput t_{comp}. The system throughput, i.e., t_{sys}, is the minimum of the communication and computational throughput. Results are conducted on the following parameters: Processors (P); Nodes (N); Links (L); Edges (E); Communication throughput t_{comm}, Computational throughput t_{comp}; System throughput t_{sys}.

Computation as well as the communication costs of the real world DAGs were calculated using the ETC generation method based on the Coefficient of Variation [2], [15].

A. Comparison Parameters

Performance comparison used the following parameters.

1) Makespan: Makespan is the completion time of the whole application graph. The makespan is calculated as:

$$makespan = max(LFT, TFT) \tag{16}$$

2) Schedule Length Ratio (SLR): The schedule length of the output schedule is considered as the main performance comparison metric. SLR is defined as the ratio of makespan to the sum of Critical Path (CP) tasks computational costs. The size of the task graph varies therefore the schedule length is normalized to lower bound which is referred to as Schedule Length Ratio. The schedule length ratio is calculated as:

$$SLR = \frac{max(TFT, LFT)}{\sum_{v_j \in CP_{MIN}} min(cw(v_j))} \tag{17}$$

The graphs input were divided into two types: DAG graph inputs and real world application graphs.

B. Results on DAG Task Graph

The number of trails was repeated for the DAG as the ETC is generated randomly. To ensure that the results are not biased and that the selection of the DAG is purely random the trails for each size of DAG were repeated from 50 to 100. Random DAGs were generated based on number of parameters for simulation of proposed algorithm. DAG graphs were generated to get DAG of varying sizes, i.e., nodes n varying from 5 to 250. These DAG graphs were also tested by varying the DAG specific properties such as α, CCR, maximum out-degree d_{out}, and other topology parameters such as number of processors P, and number of links L.

Following set of parameters are use in our evaluation:

- Set of nodes, n = {5, 10, 25, 50, 100, 150, 200, 250}.
- Communication to computation ratio, CCR = {0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 2.00}.
- Set of Processors in network, P = {3, 6, 9, 12}.
- Set of Links in network, L = {3, 6, 9, 12, 15}.
- Set of shape parameter, α = {0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 2.00}.
- Out degree for a node, d_{out} = {2, 3, 4, 5, 6}.

For the evaluation of our algorithm, the set of link heterogeneity values, i.e., V_{link} $(0.1, 0.3, 0.5, 0.7, 0.9)$ were chosen as described previously. The average SLR of the HETS algorithm was compared with the average SLR of HEFT, SDBATS, PEGA, PETS, CPOP, and DLS algorithms was evaluated by fixing the number of processors and varying the V_{link} values. Figure 3 shows the performance results for average SLR for different V_{link} values. Average SLR value produced by HETS is shorter than HEFT, SDBATS, PEGA, PETS, CPOP, and DLS algorithms.

C. Results on Real World Application Task Graphs

Real world application task graphs of Montage, CyberShake and Gaussian Elimination were generated to get graphs of varying sizes. These graphs were also tested by varying the parameters of the topology like the number of processors P and the number of links L.

1) Performance Results on Montage Task Graph: Montage Task Graph is an astronomy related task graph [4]. Montage Application task graph is used for creating image mosaics in astrophysics [17].

To analyse the effectiveness of our proposed algorithm, we conducted several experiments using the Montage task graph. The structure of Montage task graph is quite regular. Montage

task graph depends on the multiple input files to assemble. The results were obtained on three Montage task graphs with 25, 50, and 100 tasks, where each task graph was comprised of 5, 10, and 100 input files, respectively.

The parameters of Montage task graph are taken for the performance evaluation of the HETS algorithm are communication and computation cost heterogeneities. For the evaluation of our algorithm, different values for V_{link} were chosen as described previously. The average SLR of the HETS algorithm was compared with the average SLR of HEFT, SDBATS, PEGA, PETS, CPOP, and DLS algorithms was evaluated by fixing the number of processors and varying the V_{link} values. Figure 4 shows the performance results for average SLR for different V_{link} values. Average SLR value produced by HETS is shorter than HEFT, SDBATS, PEGA, PETS, CPOP, and DLS algorithms. This shows the effectiveness of HETS algorithm.

2) Performance Results on CyberShake Task Graphs: CyberShake is used for characterizing the earthquakes [8]. CyberShake application task graph is a real world application for seismic hazard analyzing [13]. For analyzing the effectiveness of HETS algorithm, we conducted several experiments using the CyberShake task graph. The performance metric used was average SLR. Heterogeneity was considered in terms of communication and computation cost heterogeneities.

Figure 5 shows the performance of HETS algorithm in comparison to other studied algorithms. The graph shows the average SLR obtained by fixing the number of processors and varying the V_{link} values. The average schedule length ratio of HETS was less than the remaining heterogeneous algorithms. The main difference between HETS and other algorithm used for performance comparison is that HETS considers not only the task heterogeneity but also takes into account the heterogeneity of edges. The edges are scheduled on the fastest available link therefore overall contention of network is reduced and the delays are minimized which in turn reduce the schedule length.

3) Performance Results on Gaussian Elimination Task Graph: The communication cost heterogeneity V_{link} of Gaussian elimination task graph [21] is varied and results were obtained using average SLR as performance metric. The performance results are shown in Figure 6. Our evaluation shows that HETS algorithm outperformed HEFT, SDBATS, PEGA, PETS, CPOP, and DLS.

V. Conclusion

In this paper, we propose a novel scheduling algorithm, Heterogeneous Edge and Task Scheduling (HETS), for heterogeneous computing systems that incorporates heterogeneity aspect, such as, dependencies of tasks and network contention, of any given application task graph. The proposed algorithm maps the nodes on the heterogeneous processors and edges on the heterogeneous links using heuristic approach. We compare the performance of HETS with different existing scheduling algorithms, i.e., SDBTAS, PEGA, PETS, HEFT, CPOP, and DLS. We use real world application graphs, such as, CyberShake, Montage, and Gaussian Elimination as well as different graph benchmarks for evaluating HETS algorithm. Extensive evaluation with different input graphs has shown improved performance using HETS algorithm both in terms of schedule length and throughput. In future, we plan to extend HETS algorithm for dynamic task graphs to incorporate real-time variations in the available computation and network resources.

References

[1] S. G. Ahmad, E. U. Munir, and W. Nisar. PEGA: A performance effective genetic algorithm for task scheduling in heterogeneous systems. In *Proc. IEEE HPCC-ICESS*, 2012.

[2] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen. Task execution time modeling for heterogeneous computing systems. In *Proc. IEEE HCW*, 2000.

[3] R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *IEEE Proc. HCW*, 1998.

[4] G. B. Berriman, E. Deelman, J. Good, J. C. Jacob, D. S. Katz, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su. Generating complex astronomy workflows. In *Workflows for e-Science*, pages 19–38. Springer, 2007.

[5] T. D. Braun, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *JPDC*, 61(6):810–837, 2001.

[6] L. D. Briceno, M. Oltikar, H. J. Siegel, A. Maciejewski, et al. Study of an iterative technique to minimize completion times of non-makespan machines. In *Proc. IEEE IPDPS* , 2007.

[7] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE TPAMI*, 18(4):377–388, 1996.

[8] R. Graves, et al. Cybershake: a physics-based seismic hazard model for southern california. *Pure and Applied Geophysics*, 168(3-4):367–381, 2011.

[9] J.-J. Han and D.-Q. Wang. Edge scheduling algorithms in parallel and distributed systems. In *Proc. ICPP*, 2006.

[10] E. Ilavarasan and P. Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer sciences*, 3(2):94–103, 2007.

[11] S. Jin, G. Schiavone, and D. Turgut. A performance study of multiprocessor task scheduling algorithms. *The Journal of Supercomputing*, 43(1):77–97, 2008.

[12] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[13] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya, et al. Scec cybershake workflowsautomating probabilistic seismic hazard analysis calculations. In *Workflows for e-Science*, pages 143–163. Springer, 2007.

[14] E. U. Munir, S. Mohsin, A. Hussain, M. W. Nisar, and S. Ali. SDBATS: a novel algorithm for task scheduling in heterogeneous computing systems. In *Proc. IEEE IPDPSW*, 2013.

[15] E. U. Munir, L. J.-Zhong, S. S.-Fei, Z. Z.-Nian, and R. Qaisar. A new heuristic for task scheduling in heterogeneous computing environment. *Springer JZUS-A*, 9(12):1715–1723, 2008.

[16] G. C. Sih, E. Lee, et al. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE TPDS*, 4(2):175–187, 1993.

[17] G. Singh, E. Deelman, G. Mehta, K. Vahi, M.-H. Su, G. B. Berriman, J. Good, J. C. Jacob, D. S. Katz, A. Lazzarini, et al. The pegasus portal: web based grid computing. In *Proc. ACM SAC*, 2005.

[18] O. Sinnen. Reducing the solution space of optimal task scheduling. *Computers & Operations Research*, 43:201–214, 2014.

[19] X. Tang, K. Li, G. Liao, and R. Li. List scheduling with duplication for heterogeneous computing systems. *JPDC*, 70(4):323–329, 2010.

[20] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *Proc. HCW*, 1999.

[21] H. Topcuoglu, S. Hariri, and M.-y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE TPDS*, 13(3):260–274, 2002.

[22] C. Walshaw. The graph partitioning archive, 2002.