Chapter 3

Benchmarking Task Scheduling Algorithms and a Possible Framework

Many task scheduling algorithms for distributed computing on multiprocessors have been proposed in the past. These algorithms show their performance in various forms depending on the platform, applications, and assumptions. It indicates that such an available algorithm is good in one situation and may perform poorly in some other situation. It is interesting to go for significant performance assessment and comparison among these scheduling algorithms. In this chapter, we first carry out performance evaluation and comparison of task scheduling algorithms, that belong to the group of list scheduling, for randomly generated graphs and the graphs generated from real-world applications. Further, we explore possibility of a framework for benchmarking of task scheduling algorithms for distributed computing on multiprocessors. The proposed approach provides for generation of graphs through a Directed Acyclic Graph generator, then produces schedules through a scheduler which makes use of scheduling algorithms and finally analyses the results obtained by using various performance metrics. The proposed framework is general in nature.

3.1 Benchmarking

Many benchmarking studies for task scheduling algorithms are proposed in literature. Kwok and Ahmad [72] presented a taxonomy of algorithms and a set of benchmarks for the comparative study of 15 scheduling algorithms. In [73], the authors compared 11 scheduling algorithms for mapping a class of independent tasks onto heterogeneous distributed computing systems. A standard task graph set is proposed and four different scheduling algorithms are compared in [74]. Mishra et al. [75] presented a comparison of six clustering-based scheduling algorithms based on static and dynamic priorities. In [76], the authors presented a comparison of six very old list scheduling algorithms on homogeneous multiprocessors. The earlier benchmarking studies reported in the literature consider mostly the independent tasks, whereas this chapter considers dependence among tasks as depicted by the task graphs. We have used, for the purpose of comparison, the representative task graphs such as randomly generated task graphs and real-world application graphs that have been used by other researchers.

In the literature, many list scheduling algorithms are proposed with completely different assumptions. We consider six of them that are well-known and frequently cited algorithms with a common number of assumptions to provide comparison and benchmarking results. The list scheduling algorithms used in this chapter are HEFT (Heterogeneous Earliest Finish Time) [10], PETS (Performance Effective Task Scheduling) [15], LDCP (Longest Dynamic Critical Path) [9], Lookahead [50], CEFT (Constrained Earliest Finish Time) [51] and PEFT (Predict Earliest Finish Time) [7]. The results are presented using various performance metrics for two types of graphs: (1) the graphs that are generated randomly and (2) the graphs that are generated from real-world applications such as Fast Fourier Transform (FFT), Gaussian Elimination (GE), Montage workflow and Epigenomics workflow.

The following are certain assumptions that are used by the scheduling system model of list scheduling algorithms used in this chapter:

- Target computing environment is made of a set of heterogeneous processors.
- The processors are fully connected with each other.

- A processor can execute a single task at any time.
- Each task is computed by a single processor.
- The tasks used in this model are dependent tasks and can′t be preempted, once a processor starts executing them.
- After finishing the execution, the task transmits output data to all immediate children concurrently.
- The computation and communication operations can be performed concurrently by a node in the system.

3.2 Metrics

The comparison metrics used in this chapter are Schedule Length Ratio (SLR), speedup, efficiency, slack and "frequency of best schedules".

3.2.1 Schedule Length Ratio

The Schedule Length Ratio (SLR) is a commonly used metric to compare scheduling algorithms, and it is defined as the fraction of the makespan to the sum of the smallest execution cost of all tasks on the critical path of the DAG [10].

$$SLR = \frac{makespan}{\sum_{T_i \in CP_{MIN}} \min_{p_j \in P} ET(T_i, p_j)} \tag{3.1}$$

where $ET(T_i, p_j)$ denotes the execution cost of task T_i on processor p_j. In Eq. 3.1, the denominator gives the lower bound on the schedule length. Thus, the algorithm having the lowest value of SLR than the other algorithms is the best algorithm.

3.2.2 Speedup

The speedup is defined as the fraction of the sequential computation time to the parallel computation time or the makespan of the schedule.

$$Speedup = \frac{Sequential\ computation\ time}{makespan} \tag{3.2}$$

In Eq. 3.2, the numerator can be calculated by allocating all tasks to one processor which gives the minimum sum of the computation cost of the DAG.

3.2.3 Efficiency

Efficiency is defined as the speedup divided by the quantity of processors used in each run.

$$Efficiency = \frac{Speedup}{Number\ of\ processors\ used} \tag{3.3}$$

3.2.4 Frequency of best schedules

This metric is used to show that how many schedules produced by an algorithm are better, worse and equal when compared to other algorithms used in the experimentation. The results of algorithms for this metric are represented by a comparison table.

3.2.5 Slack

This metric is used to determine the robustness of the schedules generated by an algorithm when compared with the uncertainty in the tasks processing time [77, 78]. It can be expressed as follows:

$$Slack = \frac{\left[\sum_{T_i \in V} makespan - BL(T_i) - TL(T_i)\right]}{n} \tag{3.4}$$

where n is the number of tasks in the task graph, $BL(T_i)$ is the length of the largest path from task T_i to the exit task, and $TL(T_i)$ is the length of the largest path from the entry task to the task T_i. It signifies the ability of the schedule to consume delays in task computation. The slack of a task denotes the time slot where the task can be deferred without enhancing the schedule length. Schedule length and slack are contradictory metrics such that higher schedule lengths give large slack.

3.3 Algorithms considered for Benchmarking

Here, we describe six well-known and frequently cited list-based scheduling algorithms such as HEFT, PETS, LDCP, Lookahead, CEFT and PEFT to schedule tasks of an application for distributed computing on heterogeneous multiprocessors.

3.3.1 HEFT Algorithm

Topcuoglu et al. [10] proposed two algorithms named HEFT and CPOP (Critical Path on a Processor) for heterogeneous computing systems. Here, only HEFT is discussed. The authors defined two types of ranks, one is upward rank and the other is downward rank. HEFT uses upward rank while CPOP uses both the ranks. The upward rank for a task T_i is calculated as follows:

$$rank_u(T_i) = \overline{ET(T_i)} + \max_{T_j \in succ(T_i)} \left\{ \overline{CT(e_{i,j})} + rank_u(T_j) \right\} \tag{3.5}$$

where $\overline{ET(T_i)}$ represents the average execution time of task T_i, $\overline{CT(e_{i,j})}$ denotes the average communication time between tasks T_i and T_j and $succ(T_i)$ gives all direct successors of task T_i. The upward rank for the exit task is calculated as follows:

$$rank_u(T_i) = \overline{ET(T_{exit})} \tag{3.6}$$

The HEFT algorithm executes in 2 phases: (i) task prioritizing and (ii) processor selection. The first phase determines the priorities of all tasks through equations (1) and (2) and maintains a list of tasks arranged in non-increasing order of their priorities. When tasks have same upward ranks, the priority among tasks is decided

randomly. The second phase of the algorithm selects the topmost task from the sorted list say, T_i and computes EFT (Earliest Finish Time) of that task on each processor. The task T_i is allocated to the processor that provides smallest value of EFT of task T_i. To compute EFT, it uses insertion-based scheduling policy that attempts to include a task in the free time window between two tasks that are previously assigned to a processor, if the window is capable of scheduling that task and preserves the precedence constraints.

3.3.2 PETS Algorithm

The PETS (low complexity Performance Effective Task Scheduling) algorithm is given by Ilavarasan and Thambidurai [15] and executes in 3 phases: (i) level sorting, (ii) task prioritization and (iii) processor selection. The algorithm traverses the input DAG from start task(s) to end task(s) and sorts the tasks at each level in the first phase. The tasks of a level may execute concurrently. In the second phase, authors defined three attributes ACC, DTC and RPT, to compute the priority of each task. The ACC (Average Computation Cost) value of a task T_i can be calculated as follows:

$$ACC(T_i) = \frac{\sum_{j=1}^{p} ET(T_i, p_j)}{p} \tag{3.7}$$

where p denotes the number of processors and $ET(T_i, p_j)$ indicates the execution time of task T_i on processor p_j. The DTC (Data Transfer Cost) of a task T_i is defined as the quantity of communication time acquired to transmit the data and it is computed at each level as follows:

$$DTC(T_i) = \sum_{j=1}^{t} CT(e_{i,j}), i < j \tag{3.8}$$

$$DTC(T_{exit}) = 0 \tag{3.9}$$

where t denotes the number of tasks in the next level of DAG and $CT(e_{i,j})$ indicates the communication cost between task T_i and its successor. The RPT (Rank of Predecessor Task) of a task T_i gives a value which denotes the maximum rank of its

immediate predecessors and it is determined as follows:

$$RPT(T_i) = Max\Big\{rank(T_1), rank(T_2), ...rank(T_m)\Big\} \tag{3.10}$$

$$RPT(T_{entry}) = 0 \tag{3.11}$$

where T_1, T_2,...,T_m are the immediate predecessors of T_i. Now, the rank of a task T_i can be calculated as follows:

$$rank(T_i) = round\Big\{ACC(T_i) + DTC(T_i) + RPT(T_i)\Big\} \tag{3.12}$$

After computing the rank of all tasks, a priority value is given to each task at every level based on their ranks. At every level, the task which has maximum rank is given the highest priority among all tasks that is followed by the task having second highest value of rank and so on. When more than one task has same rank, the priority is decided on the basis of ACC value. Higher priority is given to the task which has minimum value of ACC. The algorithm uses the BFS (Breadth First Search) for level sorting and implements priority queue by using binary heap. The third phase will calculate the EFT of each task on all processors and allocate the tasks to the processors which provide smallest value of EFT for that task. As like HEFT, this algorithm also uses insertion-based scheduling policy when computing EFT for each task.

3.3.3 LDCP Algorithm

The LDCP (Longest Dynamic Critical Path) algorithm is proposed by Daoud and Kharma [9] and executes in 3 phases: (i) task selection, (ii) processor selection and (iii) status update. In the first phase, DAGPs (DAGP is a Directed Acyclic Graph that corresponds to a Processor) for all processors are constructed, and selected task is identified using the key node or the parent key node. A key node is an unscheduled node having highest upward rank among the nodes of the selected LDCP, while a parent key node is an unscheduled parent of a key node which has maximum upward rank. The upward rank of node T_i in $DAGP_j$ is computed as follows:

$$Urank_j(T_i) = ET_j(T_i) + \max_{T_k \in succ_j(T_i)} \Big\{CT_j(e_{i,k}) + Urank_j(T_k)\Big\} \tag{3.13}$$

where $ET_j(T_i)$ represents the size of task T_i in $DAGP_j$; $CT_j(e_{i,k})$ denotes the communication time between T_i and its successors in $DAGP_j$ and $succ_j(T_i)$ indicates the set of direct successors of task T_i in $DAGP_j$. The second phase calculates the completion time of the selected task on all processors and allocates the tasks to the processors which provide smallest value of the completion time for that task. As like HEFT, this algorithm also uses insertion-based scheduling policy when completion time is calculated for each task. The third phase updates the status of the system such as size of nodes which discover the selected task on all DAGPs, communication costs on all DAGPs, computation constraints on all DAGPs, temporary zero weight edges on the DAGP associated with the selected processor and URank values of the nodes which are used in the identification of the scheduled tasks on all DAGPs.

3.3.4 Lookahead Algorithm

The Lookahead algorithm is given by Bittencourt et al. [50] which improves the scheduling process of the HEFT algorithm by using the information of the task and its successors. It takes decisions for scheduling that should be beneficial for both the task and its children. Like HEFT, the Lookahead algorithm executes in 2 phases: (i) task prioritizing and (ii) processor selection. Like HEFT, each task is prioritized by computing the upward rank using equation (1) and (2) and a sorted list of task is maintained in the first phase. This algorithm modifies the second phase of the HEFT algorithm by integrating the idea of Lookahead. According to this concept, the processor chosen for a particular task is the one that reduces the maximum value of EFT from all its successors on every processor. In other words, the task will be allocated to a processor that provides the minimum finish time for all its children that were assigned using HEFT. When EFT of the children of an unscheduled task with the highest rank is computed, it may be possible that some children tasks may not become ready due to dependence on other unscheduled parents. Thus, for calculating the EFT of the children tasks, the unscheduled parents are ignored.

3.3.5 CEFT Algorithm

The CEFT (Constrained Earliest Finish Time) algorithm [51] utilizes the idea of the Constrained Critical Path (CCP). A CCP is defined as a critical path having only ready tasks such that the tasks whose predecessors have finished their execution where a Critical Path (CP) denotes the largest path between the starting task(s) and the end task(s) in the DAG. The CEFT algorithm operates in 2 phases: (i) CCPs identification and (ii) processor selection. Initially, the first phase determines all critical paths in the graph and identifies ready tasks by traversing all the critical paths. Then, CCP queues are formed by inserting the ready tasks of critical paths into it. The CCP queue uses round-robin traversal to insert tasks from the next CP if there are no more ready tasks in a CP. In the second phase, finish time of constrained critical path queues is calculated, and the tasks of CCP are allocated to the processor that yields smallest value of the completion time of constrained critical path queue. The start time of task w concerning its predecessor task k when task w is allocated to the processor P_r, is calculated as follows:

$$S_{P_r}(w,k) = max\Big\{(F_k + M(w,P_r,k,P_x)), A_{P_r}\Big\} \tag{3.14}$$

where F_k denotes the actual completion time of task k, A_{P_r} represents the time upon which the processor P_r is available for computation using insertion-based scheduling policy and $M(w,P_r,k,P_x)$ presents the communication cost between tasks w and k when w is allocated to processor P_r and k is allocated to the processor P_x. The finish time for task w on processor P_r can be calculated as follows:

$$Z_{P_r}(w) = max\Big\{(S_{P_r}(w,k))_{\forall k \in pred(w)}\Big\} + T_{P_r}(w) \tag{3.15}$$

Thus, the completion time of the CCP queue Q_j on processor P_r is computed as follows:

$$E_{P_r}(Q_j) = max\Big\{(Z_{P_r}(w))_{\forall w \in Q_j}\Big\} \tag{3.16}$$

All tasks of a CCP are allocated to a processor which provides minimum completion time. Consequently, the completion time of all the tasks is updated and this process is repeated for all other CCPs which were discovered in the first phase. The

granularity of a DAG being used for assignment is greater than other scheduling algorithms.

3.3.6 PEFT Algorithm

The PEFT (Predict Earliest Finish Time) algorithm is proposed by Arabnejad and Barbosa [7], which has the low time complexity and gives remarkable improvements in makespan and efficiency. It maintains an Optimistic Cost Table (OCT) which uses a Lookahead feature without adding the complexity of the OCT computation. The OCT is denoted by a matrix and in this matrix, rows represent tasks and columns denote processors. The $OCT(T_i, p_k)$ denotes the maximum of the smallest paths from the children tasks of a task T_i to the exit task, assuming that task T_i is executed on processor p_k. The $OCT(T_i, p_k)$ value can be defined as follows:

$$OCT(T_i, p_k) = \max_{T_j \in succ(T_i)} \left[\min_{p_w \in P} \left\{ OCT(T_j, p_w) + ET(T_j, p_w) + \overline{CT(e_{i,j})} \right\} \right] \quad (3.17)$$

$\overline{CT(e_{i,j})} = 0, if p_w = p_k$

For the exit task,

$$OCT(T_i, p_k) = 0, \ for\ all\ processors\ p_k \in P \quad (3.18)$$

where $ET(T_j, p_w)$ denotes the execution cost of task T_j on processor p_w and $\overline{CT(e_{i,j})}$ indicates the average communication time between tasks T_i and T_j, that is, zero when both tasks reside on a same processor. Like HEFT, PEFT executes in 2 phases: (i) task prioritizing and (ii) processor selection. The first phase of the PEFT algorithm computes the priority of each task by calculating the mean OCT value for each task as follows:

$$rank_{OCT}(T_i) = \frac{\sum_{k=1}^{p} OCT(T_i, p_k)}{P} \quad (3.19)$$

In the second phase, the suitable processor is chosen for the computation of the current task by calculating optimistic EFT (O_{EFT}) that is the summation of the EFT value to the execution time of the largest path to the end task and is evaluated as follows:

$$O_{EFT} = EFT(T_i, p_j) + OCT(T_i, p_j) \quad (3.20)$$

where the $EFT(T_i, p_j)$ value is computed through insertion based scheduling policy. Thus, the algorithm aims to select the processor that gives a smaller finish time instead of the EFT for the current task.

3.4 Experimental Results and Discussion

Here we provide the evaluation results of the HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms and discuss the results using various performance metrics on various graphs. For this purpose, two sets of task graphs are taken into consideration: randomly generated task graphs and the task graphs derived from real-world applications. The experiments are performed on a Dell PowerEdge R420 server with CentOS (version 7.3-1611), Intel(R) Xeon(R) CPU E5-2420 v2 @ 2.20 GHz processor, 2 processor sockets having 6 execution cores per processor and 192 GB of memory.

3.4.1 Randomly Generated Application Graphs

The random application graphs are produced with a Random DAG generator [7] to benchmark and estimate the performance of the well-known list scheduling algorithms on various parameters. The DAGs generated here possess different characteristics which rely on some input parameters as mentioned below.

- n: It represents the DAG size.
- *shape* (α): It helps in determining the height and width of the graph. The height and width of the graph are maintained according to a uniform distribution by using $\frac{\sqrt{n}}{\alpha}$ and $\alpha \times \sqrt{n}$ as a mean value for height and width, respectively. The numbers obtained from the uniform distribution are rounding up to the nearest value. The height represents the levels and width represents the number of tasks that may execute concurrently in the graph.
- *density* (δ): It provides the quantity of edges exist between 2 levels of the graph. The lower density value produces a small number of edges, whereas the higher density value generates more number of edges.

TABLE 3.1: Parameters and their corresponding values used in the generation of random graphs

Parameters	**Values**
n	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500
α	0.1, 0.4, 0.8
δ	0.2, 0.8
r	0.2, 0.8
j	1, 2, 4
β	0.1, 0.2, 0.5, 1, 2
CCR	0.1, 0.5, 1, 5, 10
P	4, 8, 16, 32

- *regularity* (r): It represents the consistency of the quantity of tasks at every level of the DAG. The smaller value of regularity indicates that each level of the DAG contains unlike quantity of tasks, whereas a higher value of regularity shows that each level has alike quantity of tasks.

- *jump* (j): It shows that an edge exist between the level l and level $l + j$. A jump of 1 represents a direct link between two successive levels.

- CCR: It represents the Communication to Computation Ratio such that the summation of the communication costs of all edges is divided by the summation of the execution costs on all nodes in a DAG.

- *heterogeneity* (β): It gives the range of percentage of execution costs on processors. This is a factor for processor speeds. When β has a higher value, there is a considerable difference in the execution costs of a task on various processors, whereas when β has a lower value, the computation costs assigned to a task on all processors have similar value. Uniform distribution with the range $[0, 2 \times \overline{ET}_{DAG}]$ is used to select the average value of execution cost for a task in that range, where $\overline{ET}_{DAG}$ is the randomly generated average execution cost of a graph. The execution cost of task T_i for each processor p_j is randomly set in the following range:
$\overline{ET(T_i)} \times \left(1 - \frac{\beta}{2}\right) \leq ET(T_i, p_j) \leq \overline{ET(T_i)} \times \left(1 + \frac{\beta}{2}\right)$

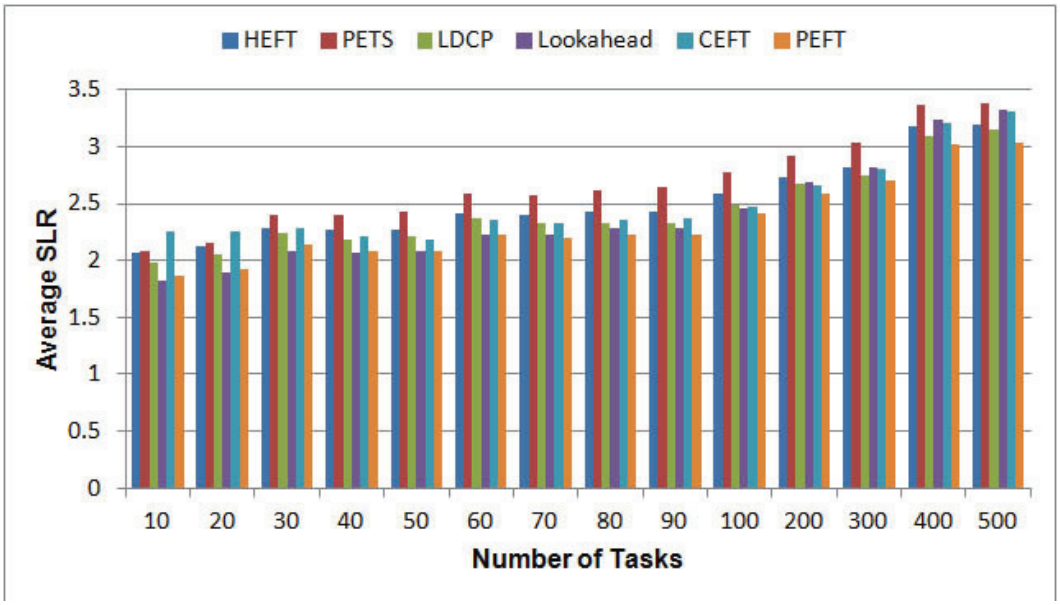

FIGURE 3.1: Average SLR results for random graphs with respect to the DAG size.

In the experiments, we used parameters given in Table 3.1 to generate random graphs. In Table 3.1, P represents the number of processors used in the experiment. Using these parameters′ combinations, 50,400 DAGs are generated. For each DAG, five different random graphs are created with the same arrangement but with different computation and communication costs on nodes and edges, respectively. Thus, totally 252,000 random DAGs are utilized in this work.

Fig. 3.1 presents the average SLR for all scheduling algorithms with respect to the various DAG sizes. For DAG sizes up to 40 tasks, the Lookahead algorithm gives the best results. For DAG size 50 and 60 tasks, PEFT and Lookahead algorithms provide similar results. For DAG sizes bigger than 60 tasks, the PEFT algorithm gives best results as it outperforms the other scheduling algorithms. The Lookahead algorithm takes scheduling decisions for a particular task by analyzing the impact of assignment of its children. When a number of children tasks of a current task increase, the load of the processor changed significantly, and the algorithm takes more time for making decisions which result in the reduction in performance for randomly generated DAGs. The PEFT algorithm uses the concept of OCT which is computed before starting the scheduling process and assumes that children tasks of a current task are computed on a processor that provides the smallest finish time of the task without considering processor availability. This reduces the total

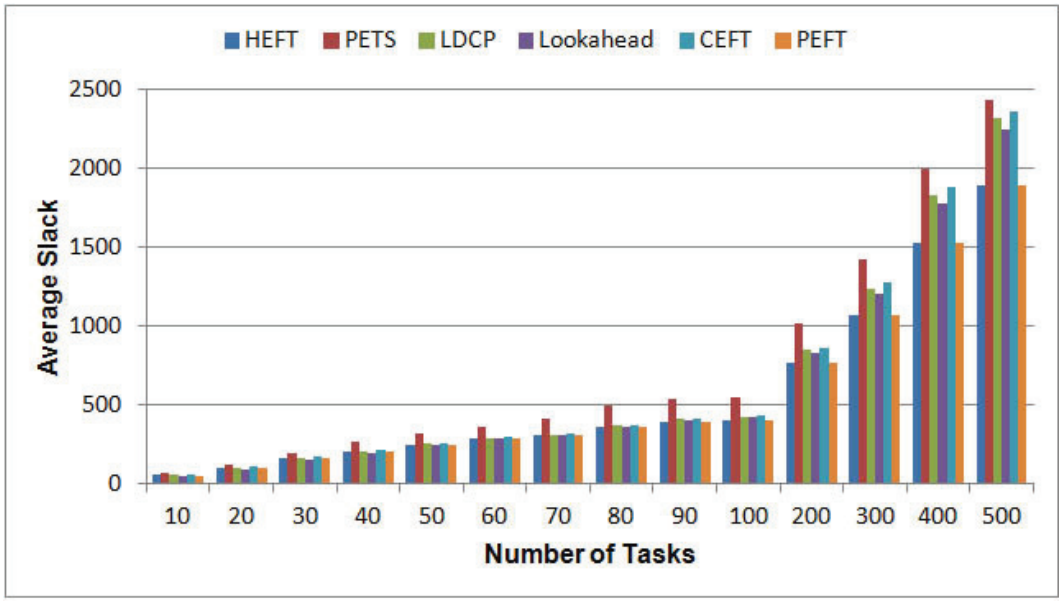

FIGURE 3.2: Average slack results for random graphs with respect to the DAG size.

time taken by PEFT to make scheduling decisions as the optimistic cost of children tasks of a current task is already known in advance. The PETS algorithm performs worst for all DAG sizes except for the DAG size 10 and 20 tasks in which CEFT gives worst results. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 2.513, 2.669, 2.440, 2.393, 2.504 and 2.338 respectively. Thus, the order of algorithms for average SLR results with respect to DAG sizes is: $PEFT < Lookahead < LDCP < CEFT < HEFT < PETS$.

Fig. 3.2 presents the average slack for all scheduling algorithms with respect to the various DAG sizes. For DAG sizes featuring up to 100 tasks, all algorithms except PETS provide similar results. For DAG sizes larger than 100 tasks, slack level of all algorithms except HEFT and PEFT continuously increases. For all DAG sizes, PETS performs worst than the other algorithms. It is observed from results that PEFT keeps equal slack level as HEFT for all DAG sizes even it generates smaller schedules with equal robustness as generated by HEFT. The mean of average slack results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 553, 727, 626, 610, 642 and 553 (the fractional part of the numbers are converted to their nearest integer values), respectively. Thus, the order of algorithms for average slack results with respect to DAG sizes is: $PEFT = HEFT < Lookahead < LDCP < CEFT < PETS$.

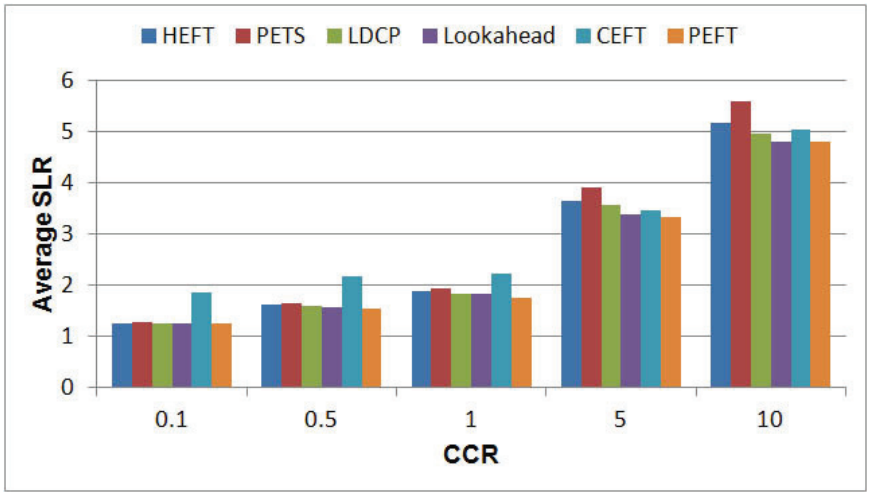

FIGURE 3.3: Average SLR results for random graphs with respect to the CCR.

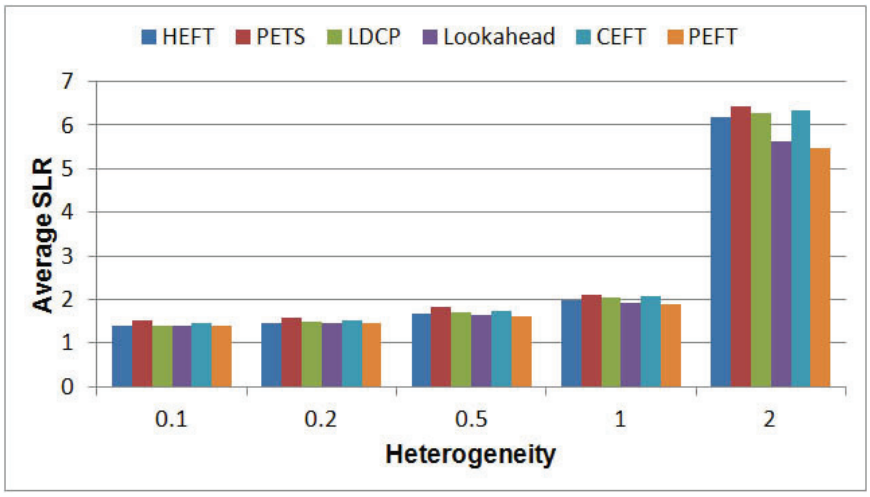

FIGURE 3.4: Average SLR results for random graphs with respect to the heterogeneity.

Fig. 3.3 presents the average SLR for all algorithms with respect to the CCR. For CCR value 0.1 and 0.5, all algorithms except CEFT algorithm give similar results. For CCR value 0.1, 0.5 and 1, CEFT provides worst results, whereas for CCR value 5 and 10, PETS gives worst results. For all CCR values, the PEFT algorithm performs better than other algorithms. The Lookahead algorithm presents similar results to that of PEFT algorithm for all CCR values. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 2.714, 2.870, 2.639, 2.566, 2.948 and 2.536, respectively. Thus, the order of algorithms for average SLR results with respect to CCR is: $PEFT < Lookahead < LDCP < HEFT < PETS < CEFT$.

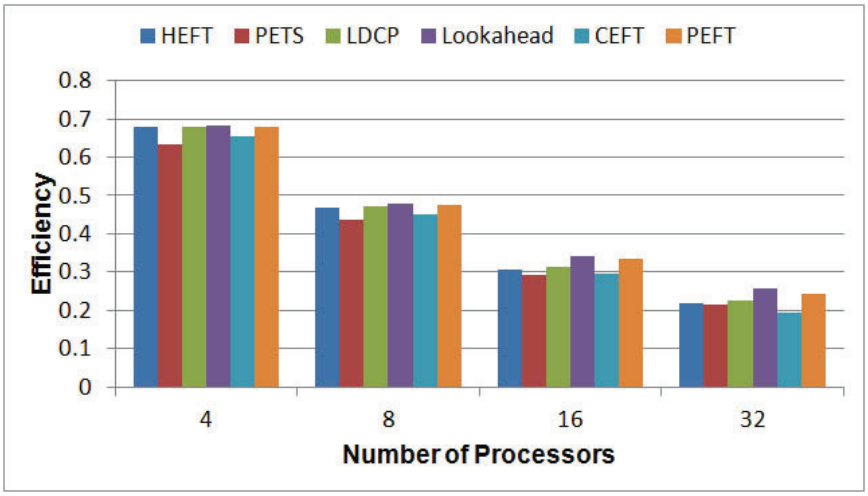

FIGURE 3.5: Efficiency of algorithms for random graphs with respect to the number of processors.

Fig. 3.4 presents the average SLR for all algorithms with respect to the heterogeneity factor. For smaller values of heterogeneity or when the computation costs for a particular task are approximately same on all processors, all algorithms perform nearly similar. The performance of the algorithms varies when the value of heterogeneity becomes larger. From results, it is clear that PEFT has smallest average SLR for all heterogeneity values and after PEFT, Lookahead offers better results than the other algorithms in terms of SLR. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 2.534, 2.690, 2.586, 2.410, 2.628 and 2.360, respectively. Thus, the order of algorithms for average SLR results with respect to heterogeneity is: $PEFT < Lookahead < HEFT < LDCP < CEFT < PETS$.

Fig. 3.5 presents the efficiency of all algorithms with respect to the various numbers of processors. From results, it is clear that Lookahead has the highest value of efficiency among all algorithms for all processor quantities. After Lookahead, PEFT presents better solutions than the other algorithms in terms of efficiency. For a lower number of processors, HEFT, LDCP, Lookahead and PEFT algorithms give nearly similar results. For processor quantity 4, 8 and 16, PETS gives worst results, while for processor quantity 32, CEFT gives the worst result. The average value of efficiencies for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 0.417, 0.394, 0.423, 0.440, 0.399 and 0.432, respectively. Thus, the order of algorithms for efficiency is: $PETS < CEFT < HEFT < LDCP < PEFT < Lookahead$.

TABLE 3.2: Pair-wise Schedule Length Comparison of the Scheduling Algorithms.

Algorithms	Schedules	Algorithms					
		HEFT	PETS	LDCP	Lookahead	CEFT	PEFT
HEFT	better	*	90%	20%	29%	32%	24%
	equal		0%	9%	6%	16%	3%
	worse		10%	71%	65%	52%	73%
PETS	better	10%	*	9%	12%	24%	8%
	equal	0		0%	0%	8%	0%
	worse	90%		91%	88%	68%	92%
LDCP	better	71%	91%	*	34%	46%	31%
	equal	9%	0%		9%	21%	4%
	worse	20%	9%		57%	33%	65%
Lookahead	better	65%	88%	57%	*	74%	29%
	equal	6%	0%	9%		12%	7%
	worse	29%	12%	34%		14%	64%
CEFT	better	52%	68%	33%	14%	*	18%
	equal	16%	8%	21%	12%		0%
	worse	32%	24%	46%	74%		82%
PEFT	better	73%	92%	65%	64%	82%	*
	equal	3%	0%	4%	7%	0%	
	worse	24%	8%	31%	29%	18%	

Table 3.2 shows the pair-wise comparison of schedule lengths in the percentage of better, equal, and worse results generated by all scheduling algorithms. For instance, when Lookahead is compared with PETS, it accomplishes better scheduling in 88% of runs, equivalent schedules in 0% of runs and worse schedules in 12% of runs.

3.4.2 Real-world Application Graphs

Besides the randomly generated DAGs, the performance of the algorithms is evaluated for the real-world applications, specifically Gaussian Elimination [10, 11, 39], Fast Fourier Transform [10, 40], Montage workflow [42–44] and Epigenomics workflow [44, 45]. All of these applications are well known and used in real-world problems.

TABLE 3.3: Configuration parameters for Gaussian Elimination

Parameters	Values
Matrix size	5, 10, 15, 20, 25, 30
CCR	0.1, 0.5, 1, 5, 10
Number of processors	2, 4, 8, 16, 32

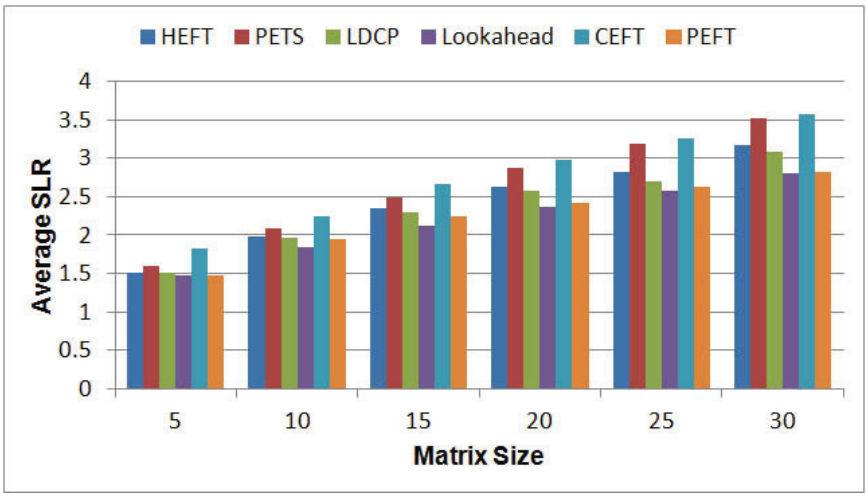

FIGURE 3.6: Average SLR for Gaussian Elimination graphs with respect to the matrix size.

3.4.2.1 Gaussian Elimination

The configuration of Gaussian Elimination is known; hence, we use different values for the matrix size, CCR and number of processors as given in Table 3.3 to perform experiments.

Fig. 3.6 presents the average SLR for all algorithms with respect to the matrix size. For lower matrix sizes, HEFT, LDCP and PEFT produce similar schedules, but when matrix size increases, PEFT leads and gives better schedules than those two algorithms. It is observed from the results that the Lookahead algorithm generates best schedules and CEFT gives worst schedules among all algorithms for all matrix sizes. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 2.408, 2.626, 2.356, 2.191, 2.760 and 2.257, respectively. Thus, the order of algorithms for average SLR results with respect to the matrix size is: $Lookahead < PEFT < LDCP < HEFT < PETS < CEFT$.

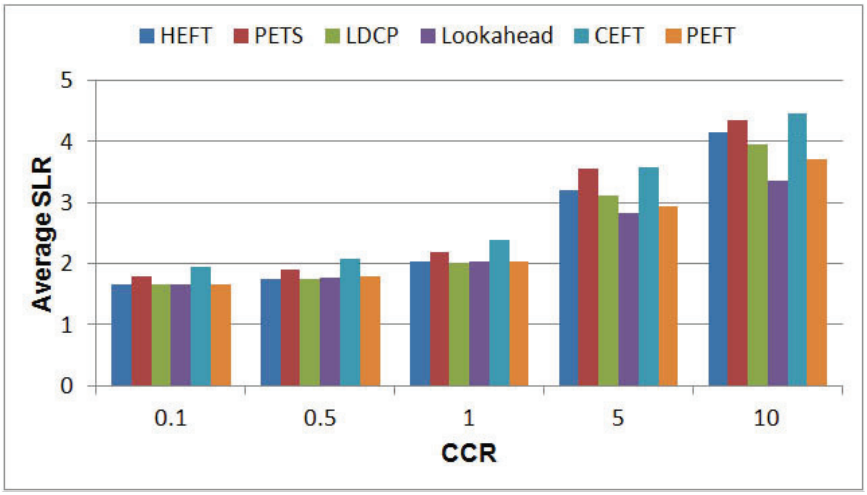

FIGURE 3.7: Average SLR for Gaussian Elimination graphs with respect to the CCR.

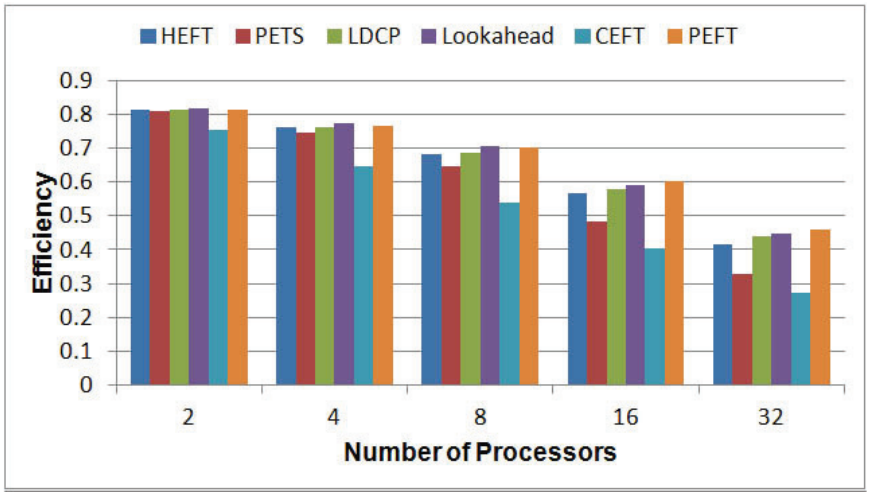

FIGURE 3.8: Efficiency of algorithms for Gaussian Elimination graphs with respect to the number of processors.

Fig. 3.7 presents the average SLR for all algorithms with respect to the CCR. For CCR up to 1, HEFT, LDCP, Lookahead and PEFT produce similar schedules, but when CCR value goes beyond 1, Lookahead leads and provides best schedules among all algorithms. It is observed from the results that the CEFT gives the worst results in terms of SLR for all CCR values. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 2.557, 2.759, 2.500, 2.331, 2.896 and 2.429, respectively. Thus, the order of algorithms for average SLR results with respect to the CCR is: $Lookahead < PEFT < LDCP < HEFT < PETS < CEFT$.

TABLE 3.4: Configuration parameters for Fast Fourier Transform

Parameters	Values
Input points	4, 8, 16, 32, 64
CCR	0.1, 0.5, 1, 5, 10
Number of processors	2, 4, 8, 16, 32

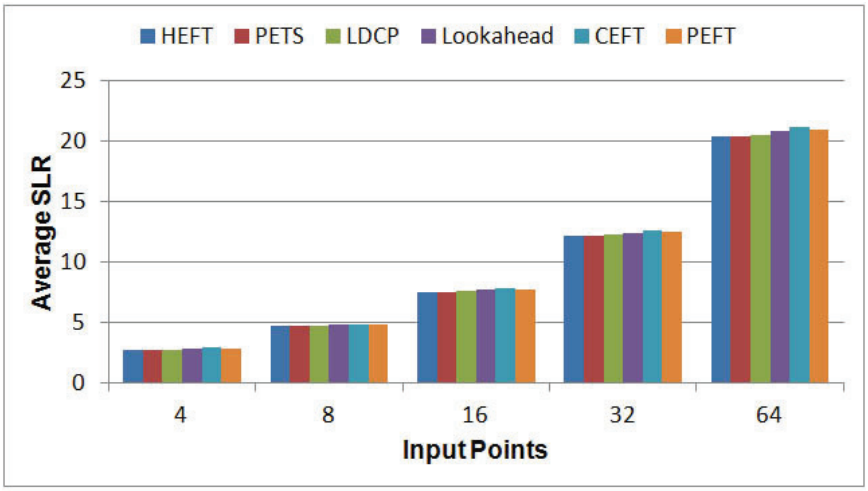

FIGURE 3.9: Average SLR for FFT graphs with respect to the input points.

Fig. 3.8 presents the efficiency of algorithms with respect to the numbers of processors. For processor quantity 2, all algorithms perform similarly except CEFT. For processor quantity 4-32, Lookahead and PEFT provide similar and better results among all algorithms where Lookahead algorithm has the highest efficiency when the number of processors is 4 and 8, whereas PEFT has the highest efficiency when the number of processors is 16 and 32. For all processor quantities, CEFT has the lowest value of efficiency among all algorithms. The average value of efficiencies for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 0.648, 0.604, 0.656, 0.668, 0.524 and 0.667, respectively. Thus, the order of algorithms for efficiency is: $CEFT < PETS < HEFT < LDCP < Lookahead < PEFT$.

3.4.2.2 Fast Fourier Transform

The configuration of FFT is known; hence, we use different values for input points, CCR and number of processors as given in Table 3.4 to perform experiments.

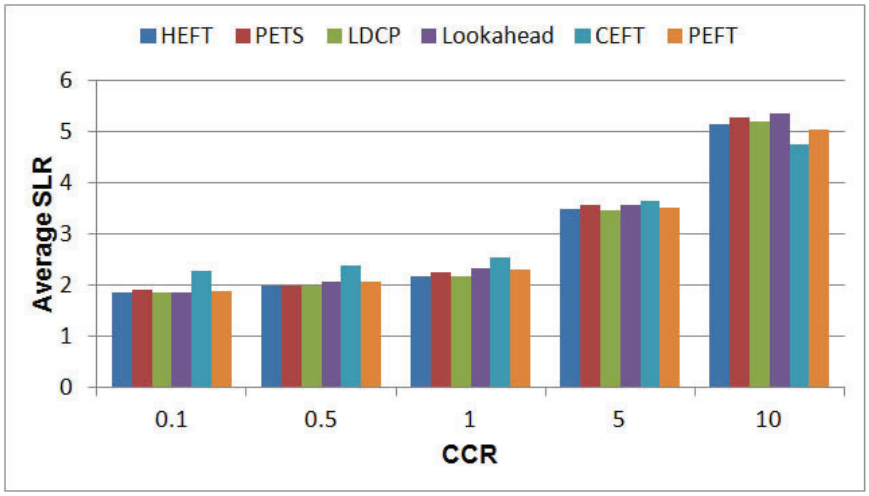

FIGURE 3.10: Average SLR for FFT graphs with respect to the CCR.

Fig. 3.9 presents the average SLR for all algorithms with respect to the input points. For all input points, CEFT and PEFT yielded the worst schedules, though all algorithms generate similar schedules. From experiments, it is observed that when each path in the DAG is a critical path, HEFT and PETS produce better and similar results than the other algorithms for all input points. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 9.506, 9.506, 9.550, 9.720, 9.856 and 9.775, respectively. Thus, the order of algorithms for average SLR results with respect to the input points is: $HEFT = PETS < LDCP < Lookahead < PEFT < CEFT$.

Fig. 3.10 presents the average SLR for all algorithms with respect to the CCR. For CCR up to 5, CEFT produces worst results, while HEFT and LDCP generate similar and better schedules than the other algorithms. PEFT and Lookahead also yielded similar schedules for CCR value up to 5. CEFT generates best schedules whereas Lookahead gives worst schedules in terms of SLR for CCR value 10. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 2.929, 2.996, 2.927, 3.032, 3.116 and 2.963, respectively. Thus, the order of algorithms for average SLR results with respect to the CCR is: $LDCP < HEFT < PEFT < PETS < Lookahead < CEFT$.

Fig. 3.11 presents the efficiency of algorithms with respect to the number of processors. For a number of processors up to 8, HEFT, PETS, LDCP, Lookahead and

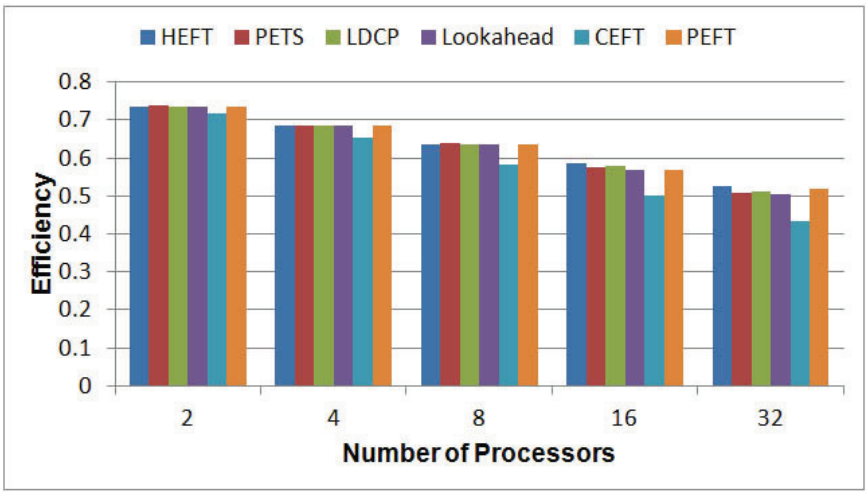

FIGURE 3.11: Efficiency of algorithms for FFT graphs with respect to the number of processors.

TABLE 3.5: Configuration parameters for Montage workflow

Parameters	**Values**
Number of tasks	25, 50
CCR	0.1, 0.5, 1, 5, 10
Heterogeneity	0.1, 0.2, 0.5, 1, 2
Number of processors	2, 4, 8, 16, 32, 64

PEFT provide similar results in terms of efficiency, but when the number of processors increases, HEFT leads other algorithms and gives best results. For all processor quantities, CEFT has the lowest efficiency among all algorithms. The average value of efficiencies for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 0.633, 0.629, 0.629, 0.625, 0.578 and 0.628, respectively. Thus, the order of algorithms for efficiency is: $CEFT < Lookahead < PEFT < PETS = LDCP < HEFT$.

3.4.2.3 Montage Workflow

The structure of Montage workflow application is known; thus, we use different values of configuration parameters as given in Table 3.5, for experiments on this workflow.

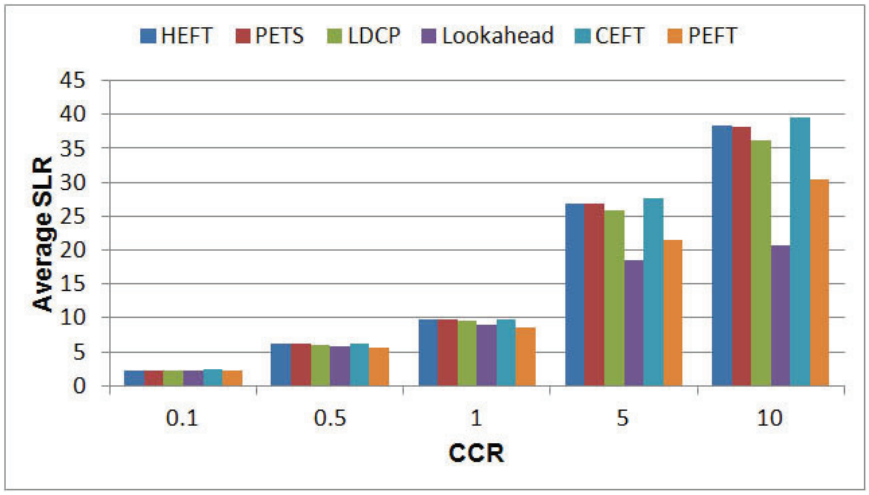

FIGURE 3.12: Average SLR for Montage workflow graphs with respect to the CCR.

Fig. 3.12 presents the average SLR for all algorithms with respect to the CCR. For computation-intensive DAGs, all algorithms produce similar results, while for communication-intensive DAGs, Lookahead gives best results in terms of SLR. When CCR value is 1, Lookahead and PEFT give similar results where PEFT yielded the smallest schedules followed by Lookahead. For all CCR values, HEFT and PETS produce similar schedules and CEFT give worst schedules. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 16.707, 16.676, 16, 11.286, 17.151 and 13.703, respectively. Thus, the order of algorithms for average SLR results with respect to CCR is: $Lookahead < PEFT < LDCP < PETS < HEFT < CEFT$.

Fig. 3.13 presents the average SLR for all algorithms with respect to the number of processors. For all processor quantities, HEFT, PETS, LDCP and CEFT produce similar results in which LDCP gives smaller schedules and CEFT provides higher schedules than the other three algorithms. For a smaller number of processors, Lookahead and PEFT produce similar and better results than the other algorithms, but when the number of processors increases, Lookahead yielded the smallest schedules among all algorithms, followed by PEFT. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 10.255, 10.255, 10.0123, 7.561, 10.449 and 8.637, respectively. Thus, the order of algorithms for average SLR results with respect to the number of processors is: $Lookahead < PEFT < LDCP < HEFT = PETS < CEFT$.

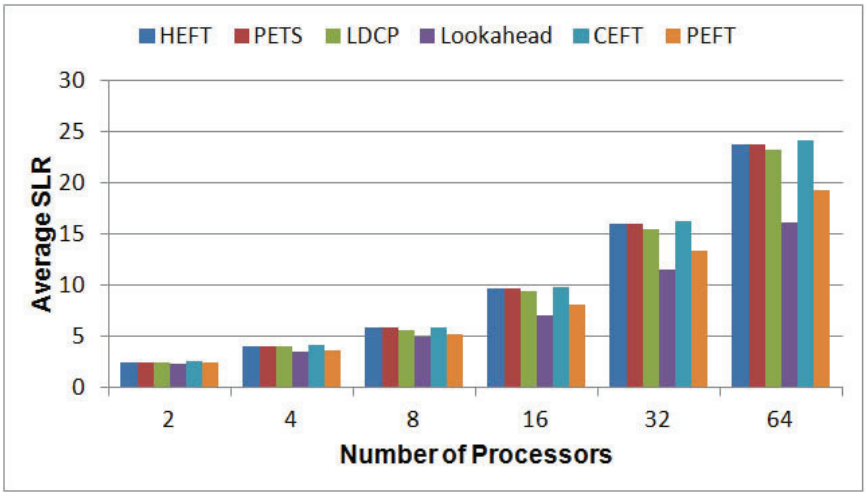

FIGURE 3.13: Average SLR for Montage workflow graphs with respect to the number of processors.

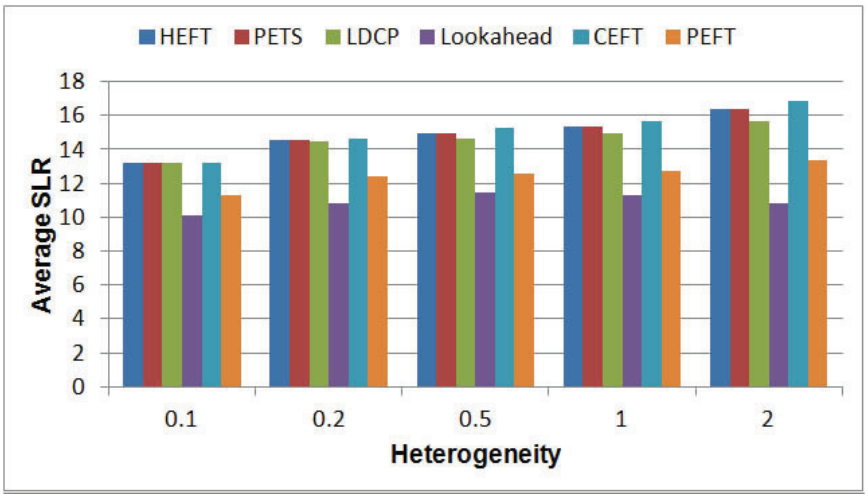

FIGURE 3.14: Average SLR for Montage workflow graphs with respect to the heterogeneity.

Fig. 3.14 presents the average SLR for all algorithms with respect to the heterogeneity factor. For heterogeneity value up to 0.2, all algorithms except Lookahead and PEFT give similar results in terms of SLR while for heterogeneity value greater than 0.2, only HEFT and PETS maintain the same level. For all heterogeneity factors, Lookahead produces smallest schedules and PEFT gives the second best result after Lookahead in terms of SLR. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 14.914, 14.903, 14.587, 10.914, 15.119 and 12.454, respectively. Thus, the order of algorithms for average SLR results with respect to the heterogeneity is: $Lookahead < PEFT < LDCP <$

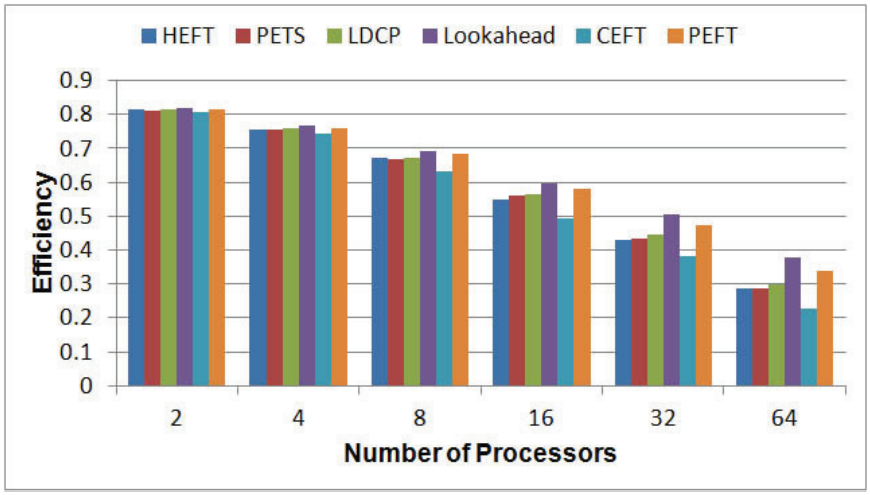

FIGURE 3.15: Efficiency of algorithms for Montage workflow graphs with respect to the number of processors.

$PETS < HEFT < CEFT$.

Fig. 3.15 presents the efficiency of algorithms with respect to the number of processors. For processor quantity up to 4, all algorithms provide similar results in terms of efficiency. HEFT, PETS and LDCP algorithms have similar efficiency for processor quantity 8-64. For all processor quantities, Lookahead has higher values of efficiency whereas CEFT has lower values of efficiency among all algorithms. The average value of efficiencies for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 0.585, 0.586, 0.593, 0.627, 0.547 and 0.609, respectively. Thus, the order of algorithms for efficiency is: $CEFT < HEFT < PETS < LDCP < PEFT < Lookahead$.

3.4.2.4 Epigenomics Workflow

The structure of Epigenomics workflow application is known; thus, we use different values of configuration parameters as given in Table 3.6, for experiments on this workflow.

Fig. 3.16 presents the average SLR for all algorithms with respect to the CCR. It is observed from the results that HEFT, PETS and LDCP produce similar results in terms of SLR for all CCR values and computation intensive DAGs, Lookahead, CEFT and PEFT give results equivalent to the other three algorithms. For CCR

TABLE 3.6: Configuration parameters for Epigenomics workflow

Parameters	Values
Number of tasks	24, 46
CCR	0.1, 0.5, 1, 5, 10
Heterogeneity	0.1, 0.2, 0.5, 1, 2
Number of processors	2, 4, 8, 16, 32, 64

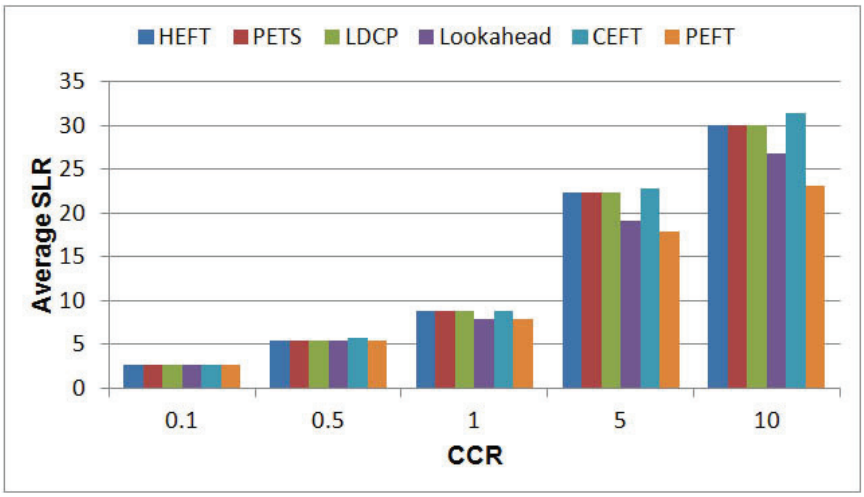

FIGURE 3.16: Average SLR for Epigenomics workflow graphs with respect to the CCR.

value 1, Lookahead and PEFT produce similar and better results than the other algorithms. When the DAGs are communication intensive, the PEFT leads other algorithms and gives best schedules, while CEFT produces worst schedules. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 13.836, 13.836, 13.836, 12.333, 14.256 and 11.347, respectively. Thus, the order of algorithms for average SLR results with respect to CCR is: $PEFT < Lookahead < HEFT = PETS = LDCP < CEFT$.

Fig. 3.17 presents the average SLR for all algorithms with respect to the number of processors. For all processor quantities, HEFT, PETS, LDCP and CEFT produce similar results in which CEFT gives higher schedules than the other three algorithms and HEFT, PETS and LDCP produce same results in terms of average SLR. For the number of processors up to 8, Lookahead and PEFT give same schedules, whereas for the number of processors greater than 8, PEFT produces the best result among all algorithms. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead,

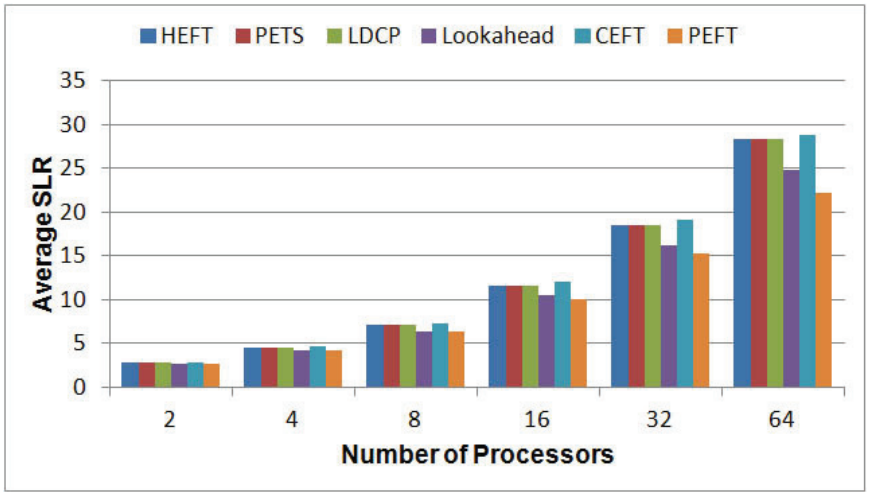

FIGURE 3.17: Average SLR for Epigenomics workflow graphs with respect to the number of processors.

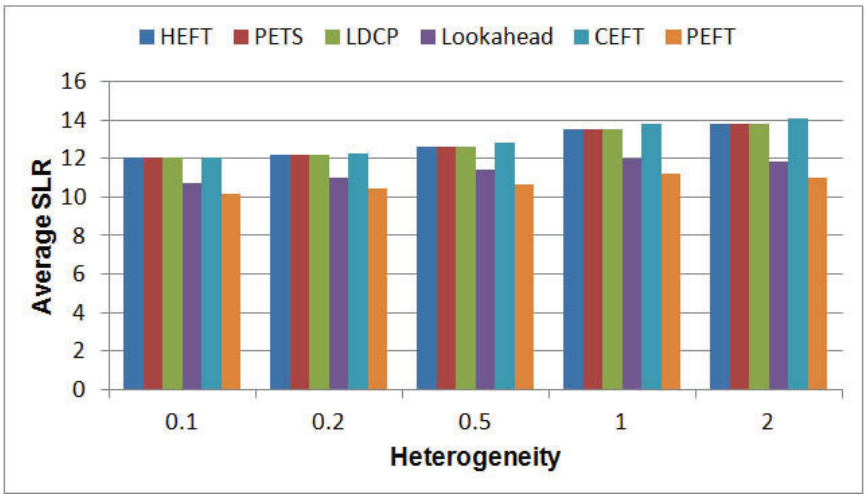

FIGURE 3.18: Average SLR for Epigenomics workflow graphs with respect to the heterogeneity.

CEFT and PEFT algorithms is 12.112, 12.112, 12.112, 10.744, 12.443 and 10.118, respectively. Thus, the order of algorithms for average SLR results concerning the number of processors is: $PEFT < Lookahead < HEFT = PETS = LDCP < CEFT$.

Fig. 3.18 presents the average SLR for all algorithms with respect to the heterogeneity factor. It is evident from the results that HEFT, PETS and LDCP produce same results in terms of SLR and PEFT produces smallest schedules for each heterogeneity value used in the experiments. CEFT is the worst algorithm, and

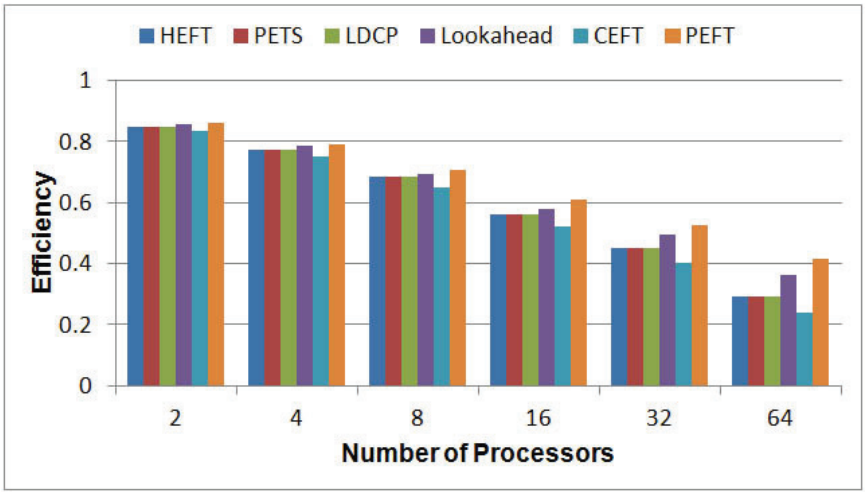

FIGURE 3.19: Efficiency of algorithms for Epigenomics workflow graphs with respect to the number of processors.

Lookahead is the second best algorithm in terms of average SLR. The mean of average SLR results for HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 12.845, 12.845, 12.845, 11.389, 13.009 and 10.703, respectively. Thus, the order of algorithms for average SLR results with respect to the heterogeneity is: $PEFT < Lookahead < HEFT = PETS = LDCP < CEFT$.

Fig. 3.19 presents the efficiency of algorithms with respect to the numbers of processors. It is observed from the results that for all processor quantities, HEFT, PETS and LDCP present same results regarding efficiency and for processor quantity up to 8, PEFT and Lookahead give similar results. CEFT gives the smallest value of efficiency, while PEFT gives the highest value of efficiency for all number of processors. The average value of efficiency of HEFT, PETS, LDCP, Lookahead, CEFT and PEFT algorithms is 0.601, 0.601, 0.601, 0.630, 0.568 and 0.652, respectively. Thus, the order of efficiency of algorithms with respect to the number of processors is: $CEFT < HEFT = PETS = LDCP < Lookahead < PEFT$.

The above comparison of task scheduling algorithms can be formalized in the form of a framework. We are proposing such a possible framework in the next section.

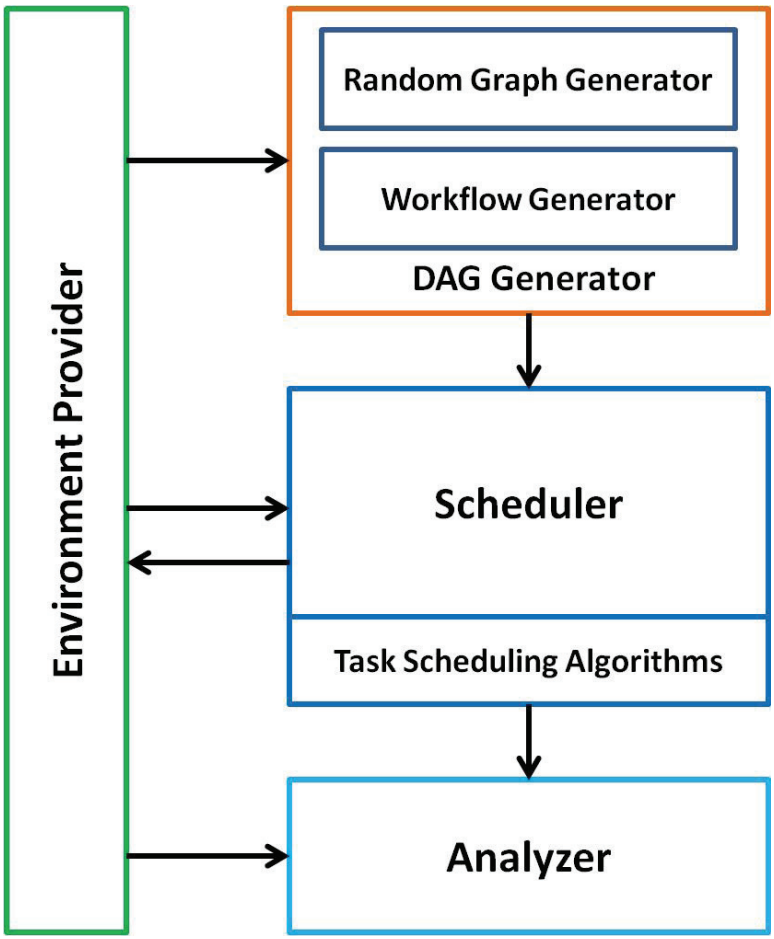

FIGURE 3.20: Proposed framework for benchmarking of task scheduling algorithms.

3.5 A Possible Framework for Benchmarking

Having carried out a comparison of task scheduling algorithms, it is plausible to explore possibility of a framework for benchmarking of task scheduling algorithms. The word "framework" ascribes a skeleton structure of the methodology being proposed in this work for the purpose of benchmarking. This framework will be helpful for people of the area to follow our method for benchmarking task scheduling algorithms for heterogeneous computing systems. The architecture of framework contains four modules as shown in Fig. 3.20: Environment Provider, DAG Generator, Scheduler and Analyzer.

Environment Provider This module provides specification regarding the computing environment of the system to all other modules of framework and takes into

consideration the communications among tasks. The specification involves:

- The number of available processors in the system.
- The type of processors used and their computing power.
- Topological information of the environment such as how the processors are interconnected with each other and how the communications among processors take place.
- Whether the capabilities of the hardware design of the processors can handle computation and communication, concurrently.

DAG Generator This module takes the input describing the computing environment and generates input graphs for scheduler. It has two sub-modules: random graph generator and workflow generator.

- The random graph generator produces random graphs which have characteristics based on input parameters such as a number of nodes in the graph, a factor which determines the height and width of the graph and a factor that decides the density of the edges between levels. This sub-module assigns the computation and communication costs according to the CCR and heterogeneity factors which reflect the behavior of target computing environment. These parameters have already been discussed in subsection 3.4.1, in detail.
- The workflow generator produces graphs derived from real-world computational workloads such as Fast Fourier Transform, Gaussian Elimination, Montage workflow and Epigenomics workflow. These graphs have fixed structures. Thus, this generator needs fewer parameters than random graph generator to produce graphs.

Scheduler This module performs the role of receiving input graphs and allocates tasks of the graph to available processors. The Scheduler produces different schedules according to task scheduling algorithms used. The Scheduler module workflow is:

1. The Scheduler takes information about the computing environment and receives the input graphs from the DAG Generator.

2. Next, the Scheduler extracts the information regarding tasks and their dependencies from graphs.

3. Now, the Scheduler generates schedules by applying list scheduling algorithms one by one. The module is expandable enough to add potential new algorithms. The Scheduler uses Environment Provider module to handle communication among tasks. In general, to produce schedules, the Scheduler,

   (a) prioritizes the tasks first and

   (b) then identifies processors to assign tasks according to their priorities.

4. Additionally, the Scheduler evaluates the performance of algorithms on different scheduling criteria such as makespan, schedule length ratio, speedup, efficiency, the frequency of best schedules and slack.

5. Finally, the Scheduler transfers all the results and information about graphs to the Analyzer.

Analyzer This module is used for the analysis of the results obtained from the Scheduler. The functions performed by the Analyzer are:

- It receives information about the graphs along with their results regarding schedules, makespan and schedule length ratio, and speedup of algorithms, etc., from the Scheduler.
- It takes information about the computing environment such as number of processors from Environment Provider module.
- It performs analysis of the results received from the Scheduler and compares the performance of scheduling algorithms on different performance criteria.

3.5.1 Advantages and Limitations of the Proposed Framework

For understanding the applicability of task scheduling algorithm, an appropriate benchmarking is required. The proposed framework will be able to cover the following, which have not been tackled earlier:

- The proposed framework is general in nature. The applicability of framework is not limited only for performance assessment and comparison of list scheduling algorithms; it can similarly be used for other types of scheduling heuristic algorithms such as clustering-based and duplication-based algorithms or combination thereof.
- It provides results for real-world application graphs such as Fast Fourier Transform task graphs, Gaussian Elimination task graphs, Montage workflow and Epigenomic workflow.
- It provides results for randomly generated task graphs as considered by other researchers.
- The framework may also be used for other computing environments like cluster computing, grid computing and cloud computing.

The proposed framework does not differentiate among types of processing units such as CPUs, GPUs, APUs, FPGAs and MICs, and it works for the computing environment in which processors are fully connected.

3.6 Summary

In this chapter, we evaluated and compared the performance of task scheduling algorithms for heterogeneous computing systems. We used six well-known list scheduling algorithms such as HEFT, PETS, LDCP, Lookahead, CEFT and PEFT for this purpose. The analysis of results is performed on different performance metrics like scheduling length ratio and efficiency. for randomly generated graphs and the graphs generated from real-world applications such as FFT, Gaussian Elimination, Montage and Epigenomics workflows.

It is concluded from the results that for randomly generated application graphs, Lookahead produces smaller schedules for small-sized graphs and PEFT gives smaller schedules for bigger graphs. All algorithms except CEFT produce similar schedules for computation-intensive graphs, whereas for communication-intensive graphs,

TABLE 3.7: Summary of the results of benchmarking for comparison of task scheduling algorithms.

Task graph	**Parameters**	**Order of algorithms**(from lowest value to highest value)
Random	Average SLR versus number of tasks	$PEFT < Lookahead < LDCP < CEFT < HEFT < PETS$
	Average slack versus number of tasks	$PEFT = HEFT < Lookahead < LDCP < CEFT < PETS$
	Average SLR versus CCR	$PEFT < Lookahead < LDCP < HEFT < PETS < CEFT$
	Average SLR versus heterogeneity	$PEFT < Lookahead < HEFT < LDCP < CEFT < PETS$
	Efficiency versus number of processors	$PETS < CEFT < HEFT < LDCP < PEFT < Lookahead$
GE	Average SLR versus matrix size	$Lookahead < PEFT < LDCP < HEFT < PETS < CEFT$
	Average SLR versus CCR	$Lookahead < PEFT < LDCP < HEFT < PETS < CEFT$
	Efficiency vs. Number of Processors	$CEFT < PETS < HEFT < LDCP < Lookahead < PEFT$
FFT	Average SLR versus input points	$HEFT = PETS < LDCP < Lookahead < PEFT < CEFT$
	Average SLR versus CCR	$LDCP < HEFT < PEFT < PETS < Lookahead < CEFT$
	Efficiency versus number of processors	$CEFT < Lookahead < PEFT < PETS = LDCP < HEFT$
Montage	Average SLR versus CCR	$Lookahead < PEFT < LDCP < PETS < HEFT < CEFT$
	Average SLR versus number of processors	$Lookahead < PEFT < LDCP < HEFT = PETS < CEFT$
	Average SLR versus heterogeneity	$Lookahead < PEFT < LDCP < PETS < HEFT < CEFT$
	Efficiency versus number of processors	$CEFT < HEFT < PETS < LDCP < PEFT < Lookahead$
Epigenomics	Average SLR versus CCR	$PEFT < Lookahead < HEFT = PETS = LDCP < CEFT$
	Average SLR versus number of processors	$PEFT < Lookahead < HEFT = PETS = LDCP < CEFT$
	Average SLR versus heterogeneity	$PEFT < Lookahead < HEFT = PETS = LDCP < CEFT$
	Efficiency versus number of processors	$CEFT < HEFT = PETS = LDCP < Lookahead < PEFT$

Lookahead and PEFT produce similar and better schedules than the other algorithms. In terms of efficiency, Lookahead has the highest value and PEFT has the second highest value among all algorithms. When the execution costs for a particular task are almost same on all processors, all algorithms produce similar schedules, but when the computation costs are different, PEFT gives the best results in terms of SLR. HEFT and PEFT offer the best and same level of robustness of the schedules generated by them when compared to uncertainty in the tasks computation time.

For real-world application graphs or when the structure of the graph is known, Lookahead and PEFT perform better than the other algorithms, whereas HEFT, PETS and LDCP produce similar results for all real-world application graphs except for FFT. For Gaussian Elimination, Lookahead and PEFT give similar and better results for all performance metrics. When each path in the graph is a critical path, all algorithms produce similar schedules, but HEFT gives the best performance among all algorithms. The Lookahead and PEFT give the best performance, including schedule length ratio and efficiency, for Montage workflow and Epigenomics workflow, respectively. Overall, the results illustrated the effectiveness of framework for benchmarking of algorithms on heterogeneous computing systems. Summary of the results of benchmarking and comparison of task scheduling algorithms is shown in Table 3.7.

We have further explored possibility of a framework for benchmarking of task scheduling algorithms for heterogeneous computing systems. Such a framework provides for collection of activities that need to be taken up for the benchmarking purpose, including random graph generator and workflow generator as part of DAG generator, a scheduler and an analyzer under the control of an environment provider.

A state forward task scheduling method may be based on merging the nodes of a task graph whenever there is a possibility of running together the associated tasks together on one and the same processor - this is done when heavily communicating tasks need to be brought together, this work is known as edge zeroing and such an algorithm is known edge zeroing algorithm [3]. We extend this idea and proposed a clustering-based task scheduling algorithm that we named as Edge Priority Scheduling (EPS) for multiprocessors environments, in the next chapter.