Research Article - Computer Science

Multi-objective Task Scheduling in Heterogeneous Cloud Computing Environment

Sidra Khalid*(sidrak2019@namal.edu.pk**)**

Fatima Sadiqa*(fatima2019@namal.edu.pk**)**

Namal University 30-KM, Talagang Road, Mianwali, Pakistan

Received: / Accepted: / Published online: /

Abstract

For the high-performance execution of applications in a heterogeneous distributed computing system, efficient task scheduling is crucial. The main purpose of this research is to build a dynamic task scheduling algorithm for a heterogeneous cloud computing environment. We suggested a comprehensive multi-objective algorithm based on PPEFT (Parental Prioritization Earliest Finish Time) algorithm for task scheduling to minimize make span and provides efficient resource utilization to be able to address the issue of dynamic task scheduling in heterogeneous cloud computing environments. This algorithm works in two phases. First phase is parental prioritization of tasks and the second phase is about processor's assignment to these tasks. The suggested algorithm is contrasted with HEFT, CPOP and PPEFT algorithms. This approach performs noticeably superior to alternative algorithms in terms of efficiency and make span while providing dynamic task scheduling and preserving the other parameters within significant bounds. It uses different benchmark scientific procedures, such as Epigenomics and Montage.

Keywords

Cloud Computing, Dynamic Task Scheduling, Heterogeneous environment, distributed systems, Parental Prioritization, makespan, resource utilization

1. Introduction

Modern technology is greatly dependent on cloud computing. It gives boundless capacity space for any kind of data. The data is put away in different information capacity sorts. For reinforcement and reestablish purposes

data can be stored within the cloud. Cloud computing empowers organizations to spare cash by lessening the require for equipment overhauls. At long last, cloud computing makes a difference companies diminish IT costs by permitting them to outsource a few of their administrations.

It is providing different reliable and secure services to the users like Infrastructure as a Service (IaaS) like Virtual machine, Servers &Storage, Platform as a Service (PaaS) like Google compute Engine, Apache Stratos and Software as a service (SaaS) like Drop box, Google Apps. All these services are provided by Cloud Service Providers (CSPs). In the form of tasks these services are performed over different set of applications. From here, the concept of task scheduling was introduced when it was required that these services should be provided to the user timely and more securely.

Task scheduling, which involves assigning tasks to available resources, is essential to effective resource use. As a result, job scheduling for heterogeneous computing systems has attracted increasing attention.

In cloud computing, task scheduling is NP-hard to solve. The set of tasks can be broken down into smaller subtasks in task scheduling systems so they can be processed in parallel. Prioritization of supplied jobs at a specified time is caused by task scheduling. Scheduling is performed to measure the order of carrying out different jobs on different virtual machines. It also optimizes the cost, make span, energy consumption and resource utilization etc. CSPs use a variety of high-end devices deployed in various data centers to offer these services to users. Normally a data center has limited resources to handle the requests of customers, but when these requests for different services are on peak in a data center, it becomes difficult to manage them at once. Due to which multiple data centers are combined to resolve this issue of delay in service providing. This generated the concept of multi-cloud environment. Every cloud has its own way of scheduling its tasks. Thus, it is difficult to arrange the tasks in a multi-cloud environment.

Heterogeneous cloud computing involves the use of different types of computing resources, such as CPUs and GPUs, to provide cloud services. The heterogeneous nature of these resources makes task scheduling a challenging task. Dynamic task scheduling involves the distribution of computing to task with resources at runtime, based on various criteria, such as resource availability, task priority, and resource utilization. Task scheduling is crucial to achieving strong dependability, computing power, availability and scalability in a lot of studies domains, especially as the use of heterogeneous computing systems increases. A heterogeneous system is a collection of computers that are linked with varying specifications, whereas a homogeneous system is a collection of connected computers with the identical specifications (processing power, RAM, etc.). Homogeneous distributed systems are easy to develop and maintain since all processing units and their capabilities are the same, in contrast to heterogeneous distributed systems, where each node has unique processing capabilities and internal bus designs. A homogeneous distributed system can only function

effectively for jobs of the same kind. Different kind of Scientific Workflows are used for simulation of task scheduling algorithms. Epigenomics and Montage are being used for simulation of algorithms.

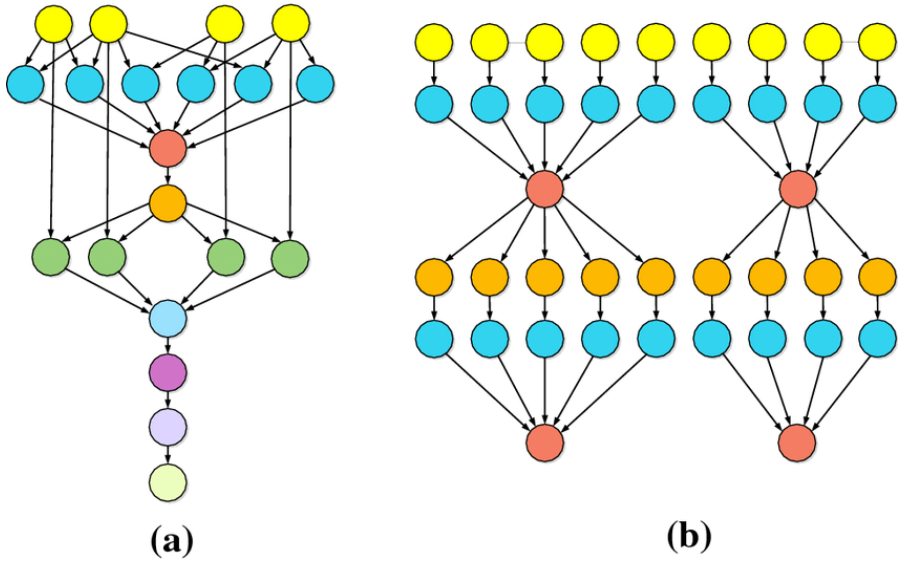

Figure1: Scientific Workflows (a) Epigenomics (b) Montage

Different problem categories can be solved by heterogeneous distributed systems with the same level of difficulty. As we know in heterogeneous cloud computing environment requirements and characteristics of all data centers and their resources are not same due to which task scheduling becomes more challenging in such environment. To resolve this issue different kind of algorithms are being developed.

Multi-objective task scheduling is main issue in heterogeneous cloud computing environment since it has a significant impact on cloud performance. Its performance also directs impact on customer's satisfaction. So, this problem should be resolved more accurately to improve its performance.

2. Related Work

Dynamic task scheduling is a crucial concern in heterogeneous cloud computing. The interest in cloud services is growing, the need for efficient and effective task scheduling algorithms is becoming more critical. Several research studies have investigated dynamic task scheduling in heterogeneous cloud computing.

Different task scheduling algorithms are used for comparative analysis of our proposed algorithm. Some of those existing task scheduling algorithms are as follow:

2.1. Min-min Algorithm

The Min-min algorithm is a simple task scheduling algorithm that aims to reduce the amount of time needed to accomplish each task. The algorithm works by selecting the work that takes the least time to complete and assigning it to the resource that is under the least load.

2.2. Max-min Algorithm

The Max-min algorithm is another simple task scheduling algorithm that aims to maximize the minimum time required to complete all jobs. The algorithm works by choosing the task that takes the longest to complete and assigning it to the resource that has the least amount of load.

2.3. Genetic Algorithm

A class of optimization algorithm called a genetic algorithm draws its inspiration from the ideas of natural selection. In the context of task scheduling, genetic algorithms aim to find the optimal allocation of tasks to resources by evolving a population of candidate solutions over time.

2.4. First Come First Service (FCFS) Algorithm

This algorithm works on FIFO (First In First Out) method. In this algorithm, tasks look for the queue where the waiting time is smallest All tasks are placed in a queue. First task from the queue is assigned to the available virtual machine. If the large task is on the front of the queue, then all other smaller tasks will have to wait for the end of execution of that larger task. The main disadvantage of this algorithm is large waiting time.

2.5. Round Robin (RR) Algorithm

This algorithm works in the form of a ring queue. Each task is assigned with a quantum of time and these sorted tasks are assigned to the available virtual machines in a circular form. If a task is not executed in the given time, then it will be interrupted and the next task will start its execution and the previous task will wait again for its turn. This algorithm continues until each task is assigned to at-least one virtual machine. This disadvantage of this algorithms is server overloading because if a task requires larger execution time, then it will not be completed in the given time and it will cause many switching until it is fully executed.

2.6. Shortest Job First (SJF) Algorithm

In this algorithm, all tasks are sorted in a queue. Small tasks are arranged in the start of the queue and long tasks are at the conclusion of the queue. The waiting time of the small tasks which are at the start of queue will

be smaller and the waiting period of larger tasks that are at the end of the queue will be larger. This will increase the waiting time of longer tasks.

2.7. Heterogeneous Earliest Finish Time (HEFT) Algorithm

The dependent jobs are scheduled using this algorithm. The algorithm is separates into two parts. Prioritization of tasks is the first step, followed by processor assignment. The initial phase involves ranking the tasks, after which processors are allocated based on these ranks.

2.8. Critical Path on A Processor (CPOP) Algorithm

The HEFT algorithm is very similar to this one. The task prioritization phase comes first, followed by the task selection phase. Priority queues are employed during the task prioritizing phase, and the crucial path to the processor is identified during the task execution phase. EFT will be employed if a critical path is not accessible. The heterogeneous cloud computing environment uses this algorithm.

3. Problem Definition

In heterogeneous distributed systems of cloud computing (HeDSCC), a parallel application's planning for dynamic work is described by a DAG. It is presented as a tuple and comprises n tasks of set T and the number of edges e. (T, E). Each job *ti* T in a parallel processing, and the communication link among *ti* and *tj* is symbolized by an edge (*ti, tj*). If a (*ti, tj*) E, then the execution of *ti* T must be complete before the execution of *tj*, T may begin. The job *tj* is a child job of edge (*ti, tj*) while *ti* is a parental task. A DAG task is referred to as an initial task if task *ti* does not have a parent, and a final task if task *tj* does not have a child.

So, each edge (*ti, tj*) has a value (*CCij*) associated with it that represents the cost of communication across jobs (*ti, tj*). The HeDSCC have m processors of set P, each with a different set of calculation capabilities. The cost of the computation, m*n, is contained in the matrix C, where m represents the total amount of processors and n, the number of tasks.

Since each processor is assumed to be fully linked, each element of matrix *CTij* C displays the computing complexity of task *ti* on processor *pj*. The autonomous conversation unit is utilized for the execution of concurrent jobs on various processors and aids in processor communication. The amount of data communicated on the edge (*ti, tj*) when two tasks are arranged on multiple processors equals the communication cost (*CCij*) from task (*ti*) to task (*tijs*). The communication cost between tasks *ti* and *tj* will be considered to be zero if they are carried out on the same processor.

Only once the parent task has finished running and all necessary data has been made available to the processor can the child task begin running. In order to execute jobs in parallel, each task must be given to the processor in a schedule that allows for a minimum overall makespan (runtime).

4. The Proposed Algorithm Dynamic Parental Prioritization Earliest Finish Time (DPPEFT)

The DPPEFT algorithm is suggested for scheduling interdependent jobs in a heterogeneous setting. The DPPEFT algorithm comprises two stages: the task prioritization stage and the processor assignment stage for task execution. The step of prioritizing tasks assigns a ranking to each task based on parental priority. The task is given to the processors during the processor selection step based on the earliest finish time.

4.1. Task Prioritization Phase

The task prioritizing phase (TTP) is a critical stage in the task scheduling process. In this stage, each task's priority is determined based on ranks determined by parental prioritizing. By ranking each task, a list of tasks is created. A DAG's ranks are determined top-down, beginning with the first job. The order of tasks that depend on the original job is then determined. Combining mean computation time and communication costs (CCs), the rank ratings of each task are determined.

If the job on the next layer has a lower rank value and is not relying on any other tasks for which order value is not determined, the parental prioritization has the advantage of scheduling it ahead of the tasks on the current layer in the DAG. The maximum communication cost of the parent job, the rank value, and the average computation time (*CTi*) of all processors of the given tasks can be used to calculate the rank of the node *ni*. To allocate to the processor, all of the estimated scores are arranged in a queue after being filtered by descending order.

The definition of task *ni* is used to determine its rank as

$$rank\ (n_i)=\text{CTi}+\left(CC_{ij}+rank(n_j)\right) \quad (1)$$

where *ni* is the active node, *CTi* is the task's mean computation time, par(*ni*) are the task's parent tasks, and *CCij* is the edge's communication cost (*ti* , *tj*).Since ranks are determined by moving from task to task in the tasks graph, it begins with the first task. Initial mission n_{ini},*s* rank is equivalent to

$$rank\ (n_{ini}) = CT_{ini} \quad (2)$$

Where CT_{ini} is the initial task's average computation time and n_{ini} is the first node.

4.2. Processor Assigning Phase

According to the tasks set in the tasks prioritizing phase, the processor allocates each task during this step. Most task scheduling algorithms take into account the earliest accessible time, which is the moment the processor finished running. However, the DPPEFT algorithm takes into account the placement strategy, which assigns jobs to the processor based on their earliest start time (EST) and earliest finish time (EFT). The task should be started as soon as possible after EST. One may figure out the EST of node *ni* by

$$EST\ (n_i\ , p_j) = \max[\text{available}(p_j), (\text{CT of}\ p_j + CC_{ij})] \qquad (3)$$

where *ni* is the active node, *pj* is the processor's total processing time, avail(*pj*) is the processor's availability, and *CCij* is the communication cost. Because computation time on each processor begins at zero, the EST for the starting task is considered to be zero for each CPU.

EFT can be determined by using

$$EFT\ (n_i\ ) = \text{CT}_\text{i} + \ EST\ (n_i\ , p_j) \qquad (4)$$

where *CTi* is the average processing time across all processors. EFT, which takes into account EST and each processor's computation for task *ni*, is the earliest finish time of jobs on each processor. Task *ni* will be assigned on the processor with the lowest EFT value, which is considered to be the idle time slot for the task.

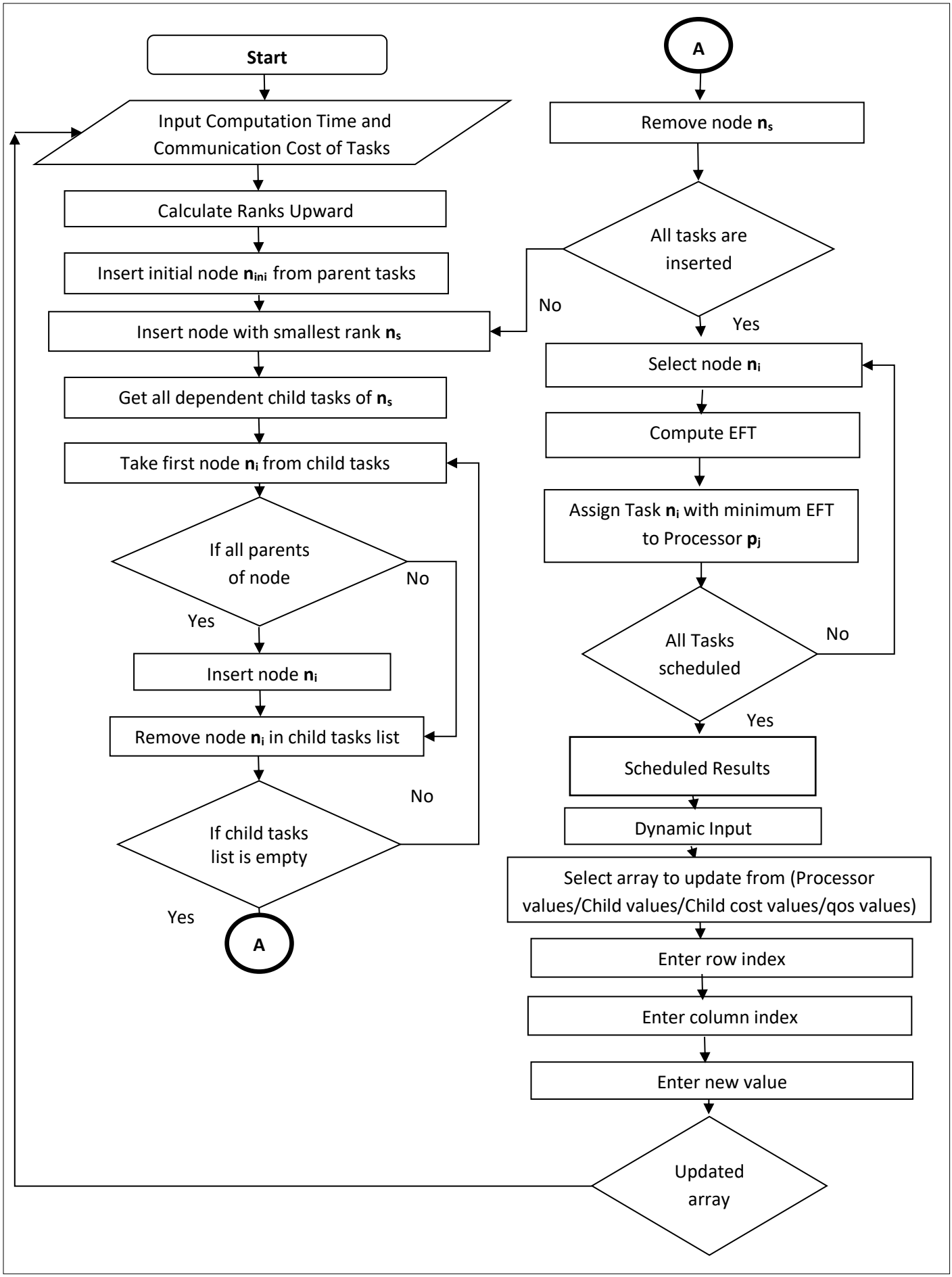
Start
Input Computation Time and Communication Cost of Tasks
Calculate Ranks Upward
Insert initial node n_{ini} from parent tasks
Insert node with smallest rank n_s
Get all dependent child tasks of n_s
Take first node n_i from child tasks
If all parents of node
No
Yes
Insert node n_i
Remove node n_i in child tasks list
If child tasks list is empty
No
Yes
A
A
Remove node n_s
All tasks are inserted
No
Yes
Select node n_i
Compute EFT
Assign Task n_i with minimum EFT to Processor p_j
All Tasks scheduled
No
Yes
Scheduled Results
Dynamic Input
Select array to update from (Processor values/Child values/Child cost values/qos values)
Enter row index
Enter column index
Enter new value
Updated array

Figure2: DPPEFT Flowchart

5. Experimental results and Discussion

Here we are comparing the effectiveness of our suggested algorithm DPPEFT against widely recognized scheduling algorithms like HEFT, CPOP and PPEFT. On Cloud Sim 3.0.3, a simulation environment is utilized and Workflow Sim 1.0 with graphs produced using a python module to examine the performance of job scheduling methods. With random and manually applied DAGs, several sets of dependent task graphs are formed. We also conducted experiments utilizing conventional scientific procedures. To produce the schedules of the tasks that are given, each task scheduling algorithm is applied to the tasks DAG.

5.1. Performance Comparison Metrics

The following performance indicators are used to compare algorithms:

- **Makespan**

The makespan is the most popular comparative metric for a single DAG. The make span is the amount of time it takes for the final task in the input DAG to finish running. The execution time of the exit node is, in other words, the algorithms make span for a DAG.A definition of an algorithm's make span is

makespan=maxAFT (last node)

Where AFT (last node) is the input graph's exit node's actual finish time. The longest finish time across all exit nodes is taken into account when there are many exit nodes in the graph.

- **Scheduling length ratio (SLR)**

The makespan is represented by the Scheduling Length Ratio (SLR), which is the lower bound normalised to it.As outlined by the SLR,

$$SLR = \frac{makespan}{min\{CP_{i,j}\}}$$

The Critical Path Including Communication (CP*min*) represents the critical path tasks' lowest possible cost, taking into account communication expenses. When an SLR is lower, the method is preferable since there is no makespan less than the CP*min*.

Step	EFT			Task Selected	Assigned Processor
	P0	P1	P2		
1	32	34	27	t0	P2
2	28	14	23	t9	P1
3	21	27	30	t7	P0
4	47	53	52	t1	P0
5	44	38	46	t8	P1
6	35	43	39	t6	P2
7	21	27	30	t5	P0
8	45	47	53	t2	P0
9	37	38	35	t4	P2
10	44	39	48	t3	P1

Table1: Schedule produced by DPPEFT algorithm in each iteration

5.2. Performance Comparison

The performance of our proposed algorithm DPPEFT is compared with HEFT, CPOP and PPEFT algorithms on the basis of makespan and average SLR.

Figure3 and **Figure4** show the total execution time (makespan) of each algorithm for different cases based on the number of tasks being executed. In each case the overall execution time of DPPEFT algorithm is less than HEFT, CPOP and PPEFT algorithms.

In **Figure5** Average SLR of DPPEFT, PPEFT, HEFT and CPOP algorithms is calculated. The average Schedule Length Ratio (SLR) of our proposed algorithm (DPPEFT) is smaller as compared to other algorithms based on the number of tasks being executed.

Figure6 shows the comparison of Average SLR of 12 tasks for HEFT, CPOP, PPEFT and DPPEFT algorithms.

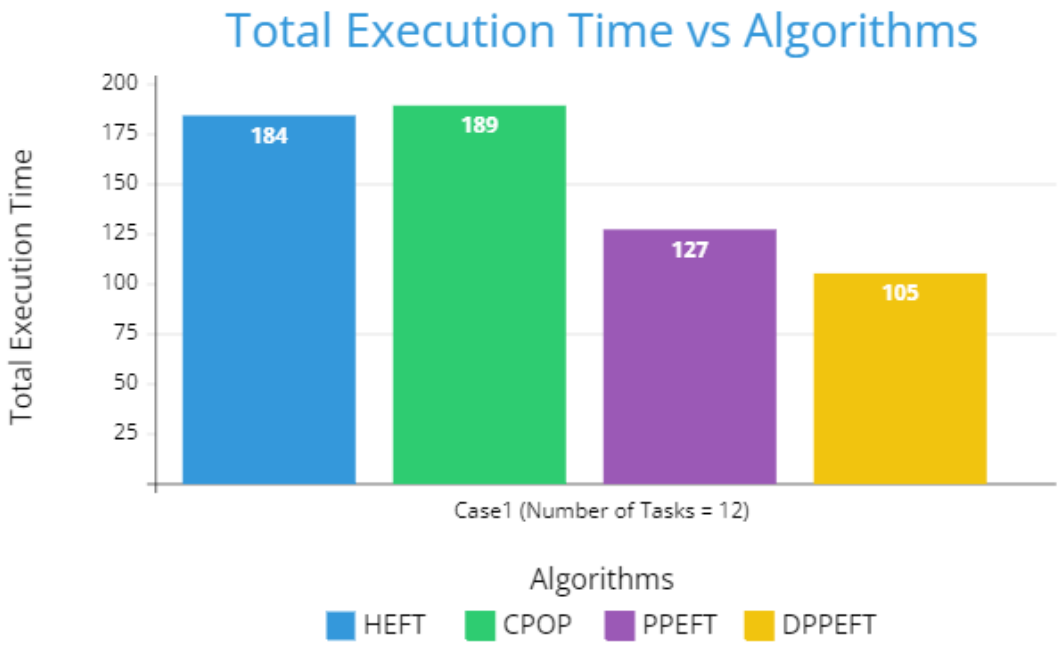

Figure3: Total Execution Time for each Algorithm (1 case)

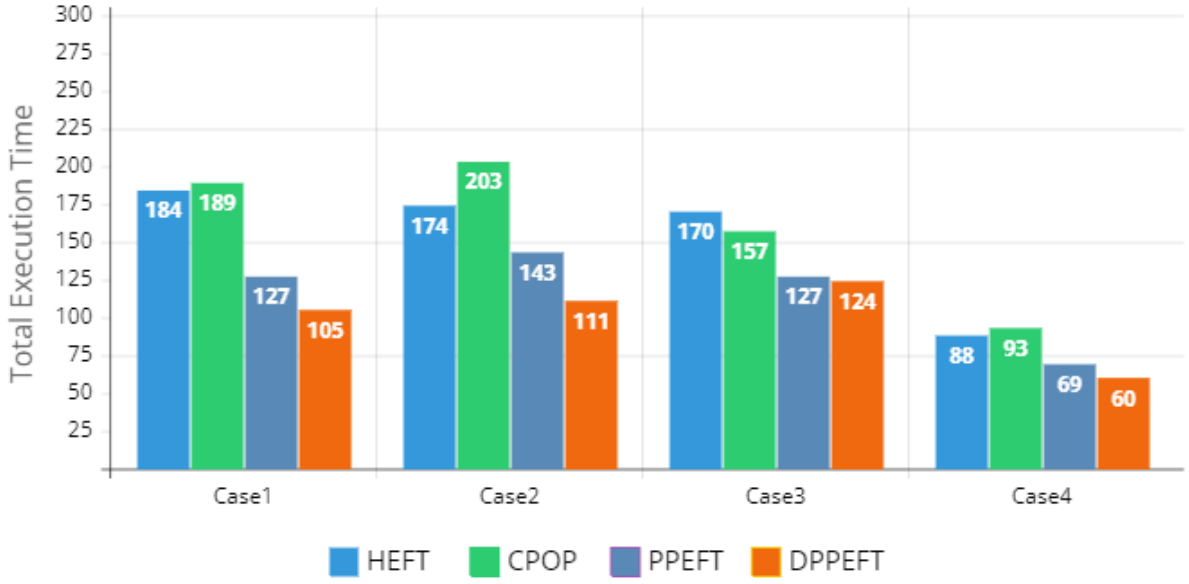

Figure4: Total Execution Time of each algorithm (4 cases)

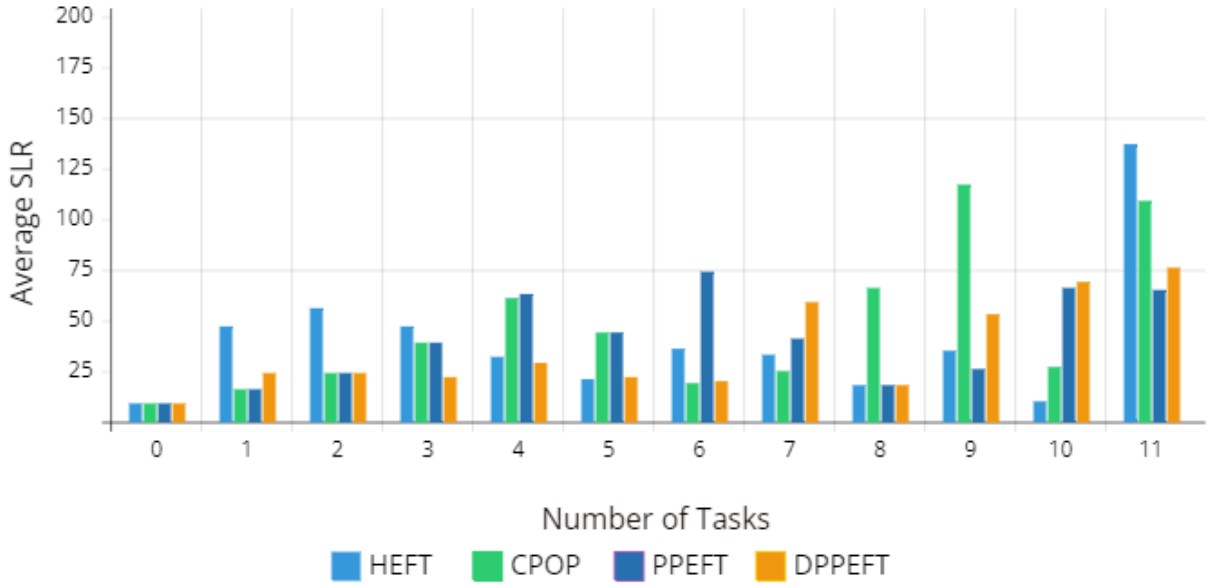

Figure5: Average SLR of 12 tasks

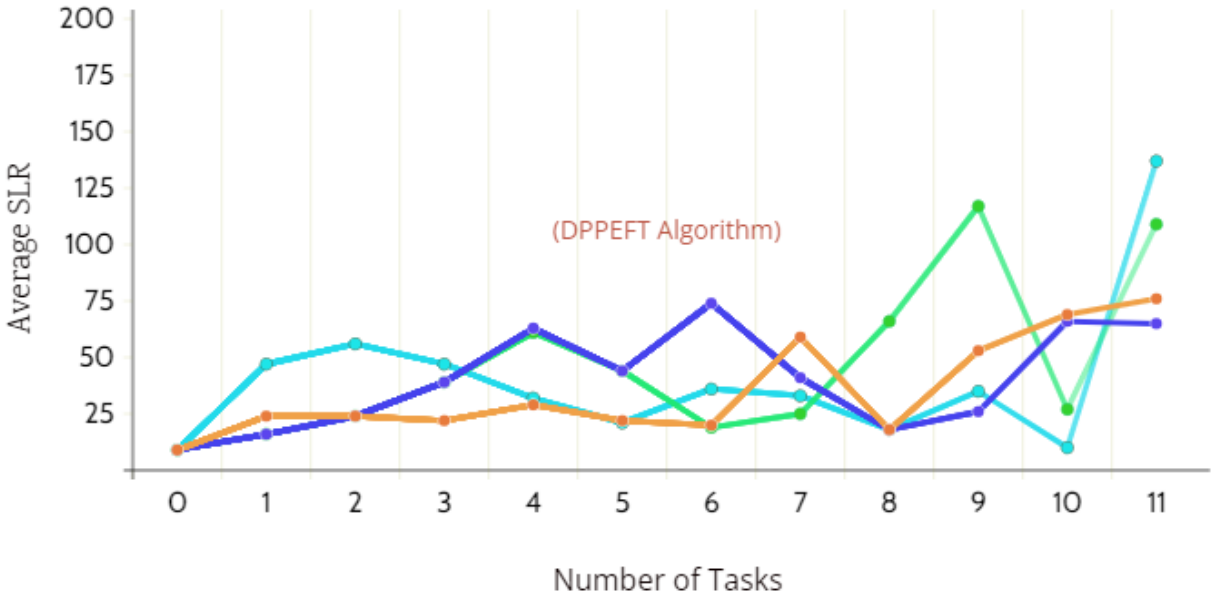

Figure6: Average SLR with 12 Tasks

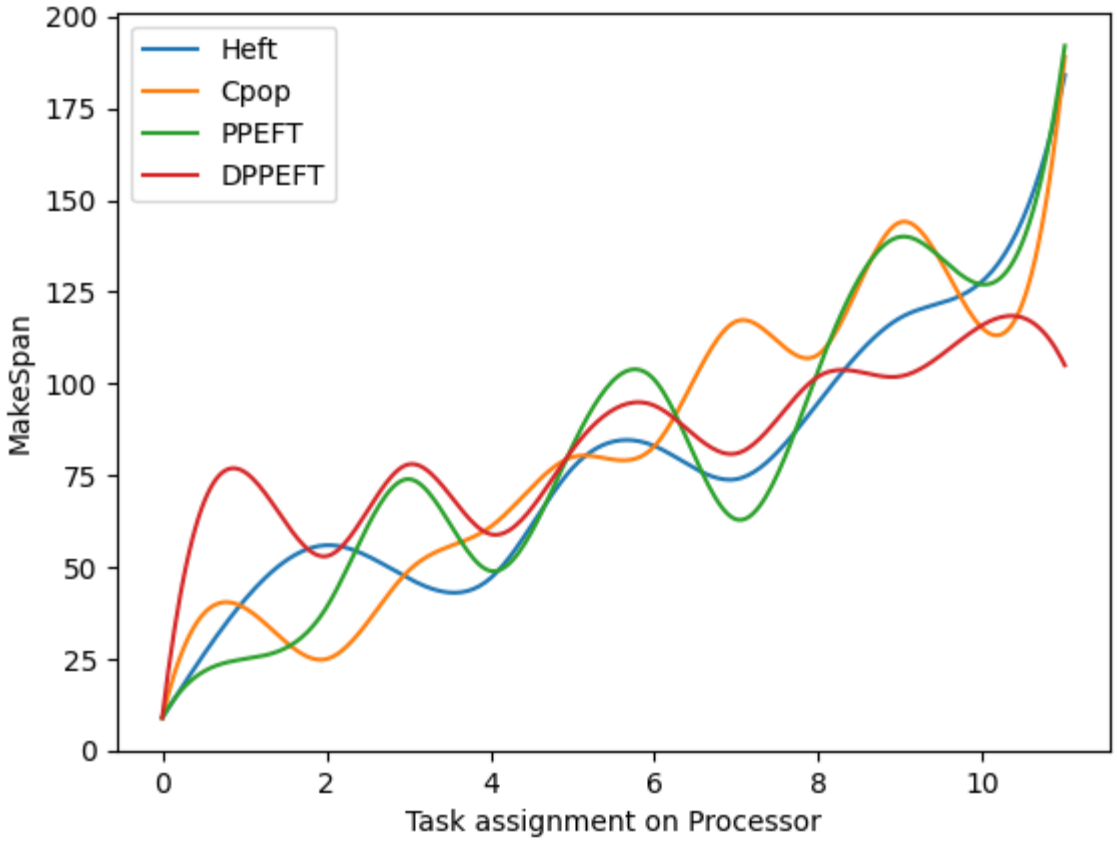

Figure7: Task Assignment on Processor with Makespan

These results show that the total execution time of DPPEFT algorithm is less than the CPOP, HEFT and PPEFT algorithms. Average SLR of each algorithm is also compared based on the number of tasks in each case because with an increase in the number of tasks, the SLR value also changes. **Figure7** shows the assignment of tasks to different processors based on their makespan (execution time).

6. Conclusion & Future Work

In this paper, we introduce DPPEFT, a dynamic task scheduling technique for heterogeneous computing systems. Task prioritizing and processor assignment are the two phases of this novel technique. In this algorithm we are taking input dynamically. In the task prioritizing phase, tasks are planned based on parental priority and descending ranks. DPPEFT schedules jobs using the parental prioritization queue (PPQ). PPQ is used to schedule lower communication cost dependent tasks from the next row of the task graph (DAG) before the present job. The list of tasks derived

from the PPQ that require the least amount of computing time is assigned to the processor during the processor assignment phase. As a result, DPPEFT produces results for work scheduling which are more effective than those of other task scheduling algorithms. Our newly developed algorithm DPPEFT's performance is compared to that of the widely recognized algorithms HEFT, CPOP and PPEFT. In terms of makespan and resource utilization the DPPEFT algorithm produces better results. For future we want to extend this DPPEFT algorithm to respond better in terms of network load and efficiency.

7. References

[1] Task scheduling for heterogeneous computing system,
https://link.springer.com/article/10.1007/s11227-016-1917-2

[2] Parental Prioritization-Based Task Scheduling in Heterogeneous Systems,
https://link.springer.com/article/10.1007/s13369-018-03698-2

[3] Task Scheduling Optimization in Cloud Computing Based on Genetic Algorithms,

https://www.techscience.com/cmc/v69n3/44156

[4] Task scheduling and resource allocation in cloud computing using a heuristic approach,
https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-018-0105-8

[5] Multi-objective Task Scheduling in Cloud Environment using Decision Tree Algorithm,
https://ieeexplore.ieee.org/document/9745131

[6] Task Scheduling and Resource Allocation in Cloud Computing using a Heuristic Approach,
https://doi.org/10.1186/s13677-018-0105-8

[7] Task Scheduling in Cloud Computing based on Meta-heuristics: Review, Taxonomy, Open Challenges, and Future Trends,
https://www.sciencedirect.com/science/article/abs/pii/S221065022100002X

[8] Enhanced Max-min Task Scheduling Algorithm in Cloud Computing,
https://www.academia.edu/download/34344253/min_max_algo.pdf

[9] Task scheduling mechanisms in cloud computing: A systematic review,
https://onlinelibrary.wiley.com/doi/full/10.1002/dac.4302

[10] A Priority based Job Scheduling Algorithm in Cloud Computing,

https://www.researchgate.net/profile/Shamsollah-Ghanbari/publication/257726215_A_Priority_Based_Job_Scheduling_Algorithm_in_Cloud_Computing/links/5a5fa392a6fdcc21f4858947/A-Priority-Based-Job-Scheduling-Algorithm-in-Cloud-Computing.pdf

[11] Hybrid Job Scheduling Algorithm for Cloud Computing Environment,

https://link.springer.com/chapter/10.1007/978-3-319-08156-4_5