Computing (2021) 103:1353–1389
https://doi.org/10.1007/s00607-021-00935-9

REGULAR PAPER

An improved list-based task scheduling algorithm for fog computing environment

R. Madhura[1] · B. Lydia Elizabeth[2] · V. Rhymend Uthariaraj[1]

Received: 10 December 2019 / Accepted: 2 March 2021 / Published online: 27 March 2021
© The Author(s), under exclusive licence to Springer-Verlag GmbH Austria, part of Springer Nature 2021

Abstract

A high-performance execution of programs predominately depends on the efficient scheduling of tasks. An application consists of a sequence of tasks that can be represented as a directed acyclic graph (DAG). The tasks in the DAG have precedence constraints between them and each task has a different timeline on different processors. In this paper, a new list-based scheduling algorithm is proposed which schedules the tasks which are represented as a DAG structure. The main focus of this algorithm is to schedule the tasks to the suitable processing node in fog environment as the fog nodes have limited processing capacity. The assignment of tasks on the fog node should consider both the computation cost of the node and the execution finishing time of the node. The proposed algorithm has three phases. (1) the level sorting phase, where the independent tasks are identified (2) in the Task prioritization phase the proposed algorithm assigns priority to the task which has more successors so that more tasks in the next level can start their execution and (3) in the task selection phase a balanced combination of local optimal and global optimal approach is considered to assign a task to a suitable processor which further enhances the processor selection phase results in minimizing both the makespan and overall computation cost of the processors. Extensive experiments are carried out using randomly generated graphs and graphs from the real-world to analyze the performance of the proposed algorithm. The results show that the proposed algorithm outperforms all other well-known algorithms like predict earliest finish time, heterogeneous earliest finish time algorithm, minimal optimistic processing time, and SDBBATS in terms of performance matrices like average scheduling length ratio, speedup, and makespan.

Keywords Directed acyclic graphs · Makespan · List scheduling · Fog environment · Task scheduling

Mathematical Subject Classification 68W10 · 68W15

✉ R. Madhura
madhuraa@mitindia.edu

Extended author information available on the last page of the article

1 Introduction

With the realization of the internet of things, communications between smart devices to take predetermined actions/decisions have become easier. However, most of the actions/decisions are based on the processing of certain data provided by these smart devices. Due to the limited storage space and processing capabilities of these smart devices, it is not possible to analyze the massive amount of data to produce instant results. So the data are transmitted to cloud computing for processing. Since cloud computing is normally providing service to individuals and is located at the provider's convenient data centers, offloading, and processing of all IoT data may cause additional communication costs and latency. Due to this limitation, cloud computing is unable to support millions of smart devices which are spread all over worldwide. This problem is alleviated by using smaller computing facilities that may not have large computing power but, can complete smaller tasks in a very short time and can provide information to the smart devices quickly to act. Hence such computing facilities that can accept time-sensitive data from the smart devices, process, and return results are called fog computing. This term was first coined by Cisco [1] in the year 2012 to overcome the limitation of cloud computing. Fog computing is similar to Cloud Computing, but are closer to smart devices and provide time-sensitive services to the devices. Fog computing is also known as "cloud to the ground" which has been proposed to overcome the limitations of cloud computing. It serves as a virtual layer between cloud data centers and end-users to provide computation, storage, and network services. Fog computing does not replace cloud computing further fog layer works for cloud layer to provide real-time services that upheld low latency, mobility, and geographical distribution.

Besides these advantages of fog computing, some challenges have to be addressed. Among them, scheduling of IoT applications in fog processing units is to be considered as an important issue. The IoT application consists of a set of computing tasks with precedence constraints among them. This forms a workflow and can be modeled as a directed acyclic graph (DAG). The graph edges between the tasks represent the flow of data from producing tasks to consuming tasks. The main advantage of the DAG model is, it facilitates parallel execution of tasks which leads to further minimize the overall scheduling length. The key challenge is to decide how to schedule the tasks over fog processing unit to minimize the makespan and overall computation cost of the processors. Some of the issues to be considered while scheduling the tasks are (1) heterogeneity in computing systems introduces an additional degree of complexity to the scheduling problem, (2) precedence constraints should not be violated during the scheduling process, (3) the order of the tasks being processed plays a major role in a scheduling problem, (4) further while assigning the task to a processor, the scheduling technic should consider the execution cost of the processors since the fog nodes have limited processing capacities.

In this paper, a new algorithm is proposed to address the issues stated above. The proposed algorithm is based on a static list-based task scheduling problem in fog environment. The main contributions of this proposed algorithm are as follows:

- A new model is being proposed which optimizes both the task prioritization phase and processor assignment phase.
- During the processor selection phase, a balanced decision is adapted which concerns both the computation cost of the processors and the Execution finishing time of a task between n number of processors in fog layer.
- Efficient use of fog resources.

This paper is organized in the way such that the relevant literature papers are discussed in Sect. 2, fog architecture and system model are explained in Sect. 3, the proposed algorithm is elucidated in Sect. 4, followed by experiments and results are discussed in Sect. 5, and conclusions are given in Sect. 6.

2 Related work

The task scheduling problem is broadly classified as static scheduling and dynamic scheduling. In static scheduling, all the information about application tasks is known before whether in dynamic scheduling such information is known only at runtime. The static scheduling algorithms are classified into two types, namely heuristic-based algorithms and guided random search-based algorithms. The heuristic-based algorithms are classified into list-based algorithms [4–9, 19, 20], clustering algorithms [24–26], and task duplication–based algorithms[17, 21–23]. Some of the classification of static task-scheduling algorithms are given in Fig. 1

In this section, an elaborate survey of task scheduling algorithms, specifically List-based heuristics algorithms is presented. In comparison with clustering algorithms, list scheduling algorithms have lower time complexity and generate high-quality scheduling. Researchers have been developing many list scheduling algorithms because scheduling in a heterogeneous environment is a highly challenging one. Most of the list scheduling algorithms have two phases: the prioritizing phase followed by the Processor selection phase. In the prioritizing phase, each task is given some priority based on their computation and communication cost, and in the processor selection phase, tasks are submitted to the suitable processor to minimize the makespan. If two or more tasks have the same priority, then a task is selected randomly to resolve the tie. Some of the list-based algorithms are discussed below.

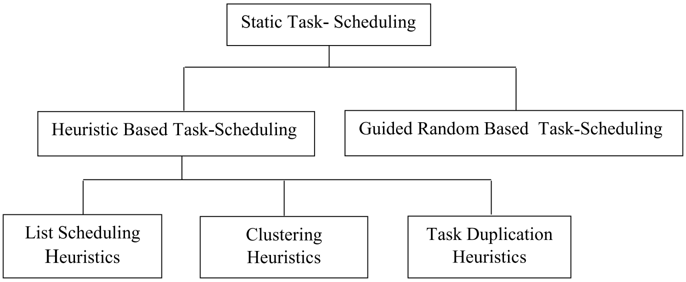
Static Task- Scheduling
Heuristic Based Task-Scheduling
Guided Random Based Task-Scheduling
List Scheduling Heuristics
Clustering Heuristics
Task Duplication Heuristics

Fig. 1 Classification of static task-scheduling algorithms

Yu-Kwong Kwokis et al. [3] designed the dynamic critical path algorithm (DCP) algorithm and it is based on the modified critical path (MCP) algorithm. It has two phases namely the node selection phase and processor selection phase. Absolute earliest start time (AEST) and absolute latest start time (ALST) are the two attributes of DCP. Based on those values the priority of the nodes is assigned. In the node selection phase, the priority of the nodes on the critical path changes dynamically. The main limitation of this approach is all the processors are homogeneous.

Haluk Topcuoglu et al. [4] have proposed two heuristic-based list scheduling algorithms for a heterogeneous environment namely (1) critical path on a processor (CPOP) algorithm (2) heterogeneous earliest finish time algorithm (HEFT). In the CPOP algorithm author tried to minimize the overall scheduling length by mapping the critical tasks to the critical processors. All the critical tasks are assigned to the same processor, which may lead to load unbalance in the schedule among the processors' results in a further increase in the scheduling length. In the HEFT algorithm priority is assigned to the tasks based on their upward rank value. The task gets its execution on the processor which produces the least earliest finishing time (EFT) which fails to consider the computation cost of the processor to execite the assigned task as an account.

Ilavarasan et al. [5] have formulated a performance effective task scheduling (PETS) algorithm. In this approach, the task with more successors gains higher priority comparatively. Then the task is assigned to the processor which has the least finishing time. Though this algorithm has a well-defined prioritization phase the task is assigned to the processor which has the least finishing time without considering the computation cost of the processors.

Bittencourt et al. [6] have proposed four different approaches to enhance the HEFT algorithm with look-ahead information. They are (1) look-ahead, (2) look-ahead with weighted average EFT, (3) look-ahead and priority list change, and (4) Look-ahead and priority list change with weighted average EFT. The authors consider one-level Lookahead which doesn't exhibit notable improvement in reducing overall scheduling length.

Shetti et al. [7] have introduced heterogeneous earliest finish time-no cross (HEFT-NC) algorithm with a non-cross mechanism in the processor selection phase to reduce the makespan in the CPU-GPU environment. This algorithm introduced the composite ranking function which incorporates both the difference between the computation cost and the ratio of the computation cost. Based on the rank, the priority of the task is assigned. The authors tried to balance between local optimal and global optimal results to further minimize the maskspan. This approach fails to consider the communication cost between the tasks while assigning the priorities. This algorithm can be applied between two processors only and the authors did not generalize the execution of their algorithm with N number of processors.

Arabnejad et al. [8] have introduced predict earliest finish time (PEFT) algorithm based on the look-ahead algorithm. The author effectively constructs an optimistic cost table (OCT) for every task. The OCT is a matrix in which the rows and columns indicate the number of tasks and the number of processors and elements of the OCT represents the shortest paths of a task t from entry node to exit node. Both the priority assigning and processor selection is based on the value of the OCT. The task selection is based on the EFT value of the processor which doesn't provide a globally optimal solution.

Hong et al. [9] have implemented a module deployment algorithm (MDA) in fog environment, which dynamically pushes programs to the devices. They implemented a real bed to evaluate the performance of the MDA algorithm. Even though the MDA algorithm produces a near-optimal solution, connecting smaller modules in dynamic flow is a critical issue.

Pham et al.[10] formulated a heuristic-based task scheduling algorithm in a cloud-fog computing environment to achieve the balance between the makespan and monetary cost of cloud resources. This algorithm assigns priority to the tasks based on their upward ranks and schedules them to the node which has the least EFT fails to consider the computation cost of the node as an account.

Taneja et al. [11] have developed a module mapping algorithm for the effective utilization of resources in fog-cloud infrastructure for IoT-based applications. The working model of this algorithm arranges the application modules and network nodes in ascending order as per their requirement and capacity. A key-value parameter is created based on their requirements and at each iteration, the network node searches for the eligible node based on their requirements. The main focus of this approach is to reduce the usage of resources and doesn't attempt to reduce latency.

Yang et al. [12] have proposed delay energy balance tasking scheduling (DEBTS) algorithm was proposed to address the energy and service delay problems in fog environment. Yang et al. developed an across-layer analytical framework to effectively handle the balance between energy consumption and service delay. The main limitation of this algorithm is that all the resources are homogeneous which doesn't reflect the real-world problem.

Tejaswini et al. [13] have introduced a three-layered architecture with an enhanced effective resource allocation (ERA) model. The authors elaborately described the queueing and priority models, priority assignment module, and the priority-based task scheduling algorithms to minimize overall response time and decrease the total cost in fog-cloud architecture.

Amir et al. [14] have presented a gravitational search algorithm (GSA) based on a meta-heuristic technique to optimize the distribution of the tasks over the resources to minimize the overall response time of the applications in fog environment. In this approach, the processing element (PE) in the IoT applications are modeled as DAG and each PE is distributed over the cloud-fog continuum.

Rezazadeh et al.[15] proposed a latency aware module placement (LAMP) algorithm which explores the placement of the IoT application module into devices. This approach schedules the application modules to the nearest devices as possible and the rest of the modules are assigned to the neighbor devices which has the least latency among them. Since the neighbor devices are arranged in the ascending order of their latency, the first neighbor device is selected to place the module. The main limitation of this algorithm is it fails to prioritize the modules of an application.

Shahzad Arif et al.[16] formulated the parental-based prioritization algorithm. Like other list scheduling algorithms it has two phases (1) tasks prioritization phase and (2) processor assigning phase. This algorithm facilitates the execution of successor tasks at the lower level if their parents finished their execution if they have fewer communication costs. But this idea will further delay the execution of the tasks at the current level.

Ijaz et al. [18] have proposed this algorithm namely minimal optimistic processing time (MOPT). These tasks are prioritized base on the OPT values, the OPT is a task and processor matrix from which the shortest path can be calculated to assign a task to a processor. The MOPT has entry task duplication which may increase the workload on the devices.

Munir et al. [19] have proposed standard deviation based algorithm for task scheduling (SDBATS). The tasks are prioritized based on the standard deviation values of their computation and communication costs. In the processor assignment phase, the tasks are assigned to the processor which has the least EFT. The limitation of this algorithm is, it incorporates an entry-level task duplication strategy which may increase the workload on the processors.

The related works discussed above concludes that there is an explicit need for a new algorithm that should address the limitations of the task scheduling problems. Some of the issues to be considered while scheduling the tasks are (1) Computation cost of a processing units should be considered while assigning a task. (2) Precedency between the tasks should not be violated. (3) The rank of the tasks should reflect their significance. To resolve these issues a new list based static task scheduling algorithm is proposed in this paper. The performance of the proposed algorithm is evaluated by conducting extensive experiments. The results show that the proposed algorithm outperforms all other well-known algorithms.

3 Fog architecture

Figure 2 represents the architecture of fog computing model. The framework of fog computing consists of a three-level hierarchy where the bottom-most layer consists of smart devices which may be wireless sensors, smartphones, home appliances, etc.

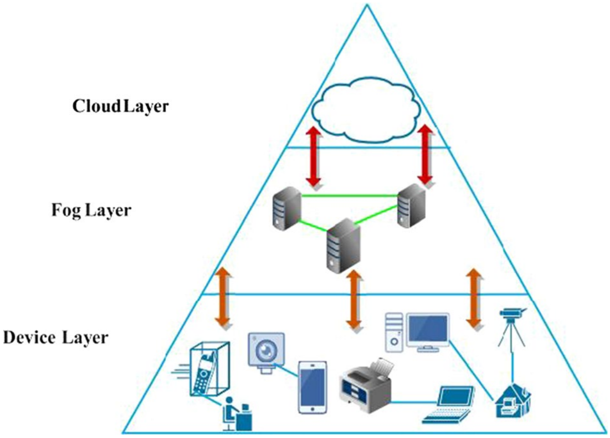

Fig. 2 Hierarchal architecture of Fog computing

These smart devices produce a huge amount of data and these data which require time-sensitive execution are transmitted to the fog layer and the rest are transmitted to the cloud layer for further analysis. The middle layer is the fog layer and it is between the terminal device layer and cloud computing layer. The fog devices are the primary component of the fog layer and are deployed on the edge of the network to provide faster execution of the user data. The devices can be routers, switches, gateways, etc. that can provide networking, computing, storage, and capabilities. They are also connected to cloud layer to get the benefits of their computing capabilities for further processing of the data. The uppermost layer is the cloud layer, the cloud data center, cloud computing, and storage devices are deployed in the cloud layer to provide huge data storage capacity, high computing ability, and processing of a large variety of data. The hierarchal architecture of fog computing is given in Fig. 2.

3.1 Scheduling model

Some of the important terminologies used in the scheduling model are listed in Table 1 given below.

3.2 Application Model

An application can be represented by a Directed Acyclic Graph G = (N, E) as shown in Fig. 4, where, N represents a set of n_i nodes, where i = (1…n) and each node n_i ϵ N represents an application task which has different execution times on different processors and E represents the set of communication edges between the tasks. Each edge e(i,k) ϵ E has a non-negative weight $\bar{c}_{i,k}$ which represents the amount of data transferred from task n_i to n_k. Data is a n x n matrix of communication data, where

Table 1 The symbols and notations

N	Task set in the in graph G
n_i	Task i in N
e(i,k)	Edge from ni to k in G
$W_{i,j}$	The computation cost of n_i
$C_{i,j}$	The communication cost of $e_{i,j}$
p_i	The ith processor in processor set P
n_{entry}	It is the entry task
n_{exit},	It is the exit task
pred(n_i)	Predecessor set of n_i
succ(n_i)	Successor set of n_i
EST(n_i, p_j)	The earliest start time of task n_i on processor P_j
EFT(n_i, p_j),	The earliest finish time of task n_i on processor P_j
AFT	Actual finishing time
Makespan	The actual finishing time of the exit task

data $_{i,\,k}$ is the amount of data required to be transferred from task n_i to task n_k. An application model of fog computing is shown in Fig. 3.

The target computing environment consists of a set of $P=\{p_j: j=0, m-1\}$ of m heterogeneous processors, each of which is a fog node. It is assumed that the nodes are fully connected. All the tasks in the given application are non-preemptive. W is a $n \times m$ computation cost matrix, where, n represents the number of tasks and m represents the number of processors in the system. $w_{i,j}$ gives the computational time of the task n_i on the processor p_j. The matrix B has the size of $m \times m$ which stores the data transferring rates between the processors. The communication cost of transferring data from the task n_i to task n_k is through edge e(i, k) is given below

$$\overline{c}_{i,k} = \overline{L} + (\mathrm{data}_{i,k})/\overline{B} \tag{1}$$

where, $\mathrm{data}_{i,k}$ is the amount of data required to be transferred from task n_i to task n_k, B represents the average transferring rate among the processors and $c_{i,k}$ is assumed to be zero when both n_i and n_k are scheduled on the same processor. L represents average communication startup time. In this paper, both the communication cost and computation cost refer to the time of communication and time of computation.

Some of the important terminologies referred to in this paper are $n_{entry,}$ n_{exit}, *pred*(n_i), *succ*(n_i), EST (n_i,p_j), EFT (n_i,p_j), AFT, and makespan. If a task in the Directed Acyclic Graph (DAG) has no predecessor task is called an entry task and it is represented as n_{entry}. If it has more than one entry node, a dummy node with zero weight and communication edge can be added to the graph. The task without

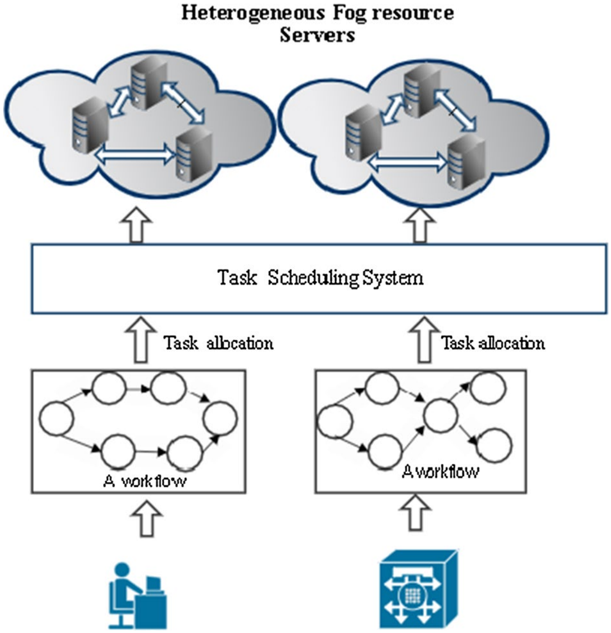

Fig. 3 An application model of fog computing

a successor task is called exit task and it is represented as n_{exit}. If a DAG has more than one exit node, then the same procedure which is followed in the entry task has to be followed. *pred*(n_i) and *succ*(n_i) represent immediate predecessors and successors of the task n_i. The entry node is the first node and the exit node is the last node that is, no predecessor for the entry node and no successor for the exit node.

EST ($n_{i,}p_j$) represents the earliest starting time and EFT ($n_{i,}p_j$) represents the earliest finishing time of the task n_i, on processor $p_{j,}$ respectively. For entry task

$$\mathrm{EST}\left(n_{entry}, p_j\right) = 0, \tag{2}$$

In Fig. 4, the entry task n_1 starts its execution from zero time. So the EST($n_{1,}$ p_1) = 0.

From the entry node, EST and EFT values are calculated recursively using the formula given below

$$\mathrm{EST}(n_i, p_j) = \max\{avail\left[j\right], \max_{n_m \in pred\left(n_i\right)} (\mathrm{AFT}(n_m) + c_{m,i}\}, \tag{3}$$

$$\mathrm{EFT}(n_i, p_j) = w_{i,j} + \mathrm{EST}(n_i, p_j), \tag{4}$$

where avail [j] is the earliest available time on the processor p_j to execute the task n_i. Here the max refers to the maximum of the available time of all the data requested by the task n_i on the processor p_j from all its predecessors. *pred*(n_i) is a set of immediate predecessor tasks of task n_i. The $c_{m,\ i}$ is the communication cost between the

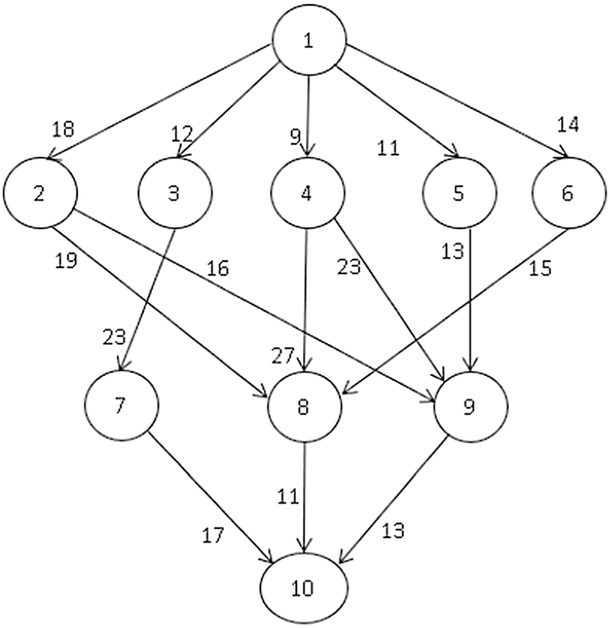

Fig. 4 An example DAG diagram with ten tasks along with its communication cost [29]

predecessor task to the task n_i. For example, in Fig. 4 the task n_8 can start its execution only if its immediate predecessor tasks $n_{2,}$ $n_{4,}$ and n_6 have finished their execution. The Earliest Start Time of task n_8 depends on the maximum of the finishing time of its predecessor along with its communication time and availability of the processor.

If the last task n_L is assigned to the processor P_j, then the availability of the processor is the execution time taken by the processor to finish that task. In Fig. 4, EST of the task n_1 is zero because task n_1 is the entry task and the execution time of task $n_1$on the processor p_3 is 9 ms. The Earliest Finishing Time of task n_1 is the summation of the execution time of task n_1 on the processor p_3 and Earliest Start Time of task n_1 on the processor $p_{3.}$

$$\begin{aligned}\mathrm{EFT}(\mathrm{n}_1,\mathrm{p}_3) &= \mathrm{w}_{1,3} + \mathrm{EST}(\mathrm{n}_1,\mathrm{p}_3)\\ &= 9 + 0\\ &= 9\,\mathrm{ms}\end{aligned}$$

After finishing the last task the processor is ready to start to execute another task which is scheduled for that processor. The scheduling length of the graph will be the finishing time of the exit task and its equation is given below

$$Makespan = \max\left\{\mathrm{AFT}\left(n_{exit}\right)\right\}. \tag{5}$$

where AFT referred to the actual finish time of the task. If all the tasks in the graphs are scheduled, then the makespan will be the actual time to finish the last task n_{exit}.

As it follows the insertion-based policy, the earliest ideal time slot between two already-scheduled tasks of the processors is effectively utilized to minimize the scheduling length. The tasks are selected based on priority. If a task has more than one predecessor task, then it has to wait until all the predecessors finish their executions. A sample application of 10 tasks with its communicational cost is given in Fig. 4. The tasks are represented as nodes and communications are represented as edges. The communication cost of task n_1 to its successor task n_2 is 18 ms and all other communication costs are given in Fig. 4. The computational costs of the tasks on different processors are given in Table 2 [29]. For example, if the task n_1 is submitted to the processor p_1, then the processor p_1 need 14 ms to finish its execution, on the other hand, if the task n_1 is submitted to the processor p_2, then the processor p_2 needs 16 ms to finish its execution.

4 The proposed algorithm

As it is based on the static scheduling problem, the communication cost, computation cost, and task dependencies are known in advance. Making use of this information a new algorithm is being proposed and the objective function of the proposed algorithm is to minimize both the overall scheduling length of an application and the overall computation cost of the processors where the tasks are assigned to gets their

Table 2 Computation time matrix of Fig. 5 (time in ms)

Task	Processor P_1	Processor P_2	Processor P3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	16	15	11
n_8	5	11	14
n_9	18	12	20
n_{10}	23	7	21

execution. The proposed algorithm has three phases. They are (1) the level sorting phase, (2) the task prioritization phase and (3) the processor selection phase. These phases are explained below.

4.1 The level sorting phase

The given DAG is traversed from top to bottom and the tasks which are independent of each other are sorted in each level so that the tasks at each level can be executed in parallel. In the given DAG G = (N, E), level l_1 has entry task n_1. The tasks n_k in the level l_i, such that the edges e(j, k) represents the task n_j is in the level less than $l_{i\text{-}1}$. For example, the graph in Fig. 4 has an entry task is always in the level l_1, next to the task n_2, task n_3, task n_4, task n_5, and task n_6 are in the level l_2, task n_7, task $n_{8,}$ task $n_{9,}$ are in the level l_3 and exit task n_{10} is in the level $l_{4.}$ The tasks in each level are independent of each other so that they can be executed in parallel. On the other hand task, n_3 and n_7 cannot be executed in parallel as task n_7 dependent on the outcome of task $n_{3.}$

4.2 Task prioritization phase

This is the second phase of the proposed algorithm. In this phase, the tasks are ordered according to their priority. The prioritization phase is the most significant phase of an algorithm, as the efficiency of the scheduling algorithm mostly depends on this phase. So in this phase, the available information is effectively utilized in assigning priority to a task to minimize the overall scheduling length of an application. To calculate the priority of the tasks, the proposed algorithm considers three attributes: (1) *cumulative execution cost (CEC), data transfer cost (DTC),* and *rank of predecessor task (RPT).* Based on these attributes values, the tasks are prioritized and scheduled to the processors.

To calculate the priority of a given task, first, the *CEC*(n_i) is calculated for each task in the given graph. It is the summation of the execution cost of the processors for the given task. The *CEC*n_1 of a given task is defined in Eq. (6)

$$CEC(n_i) = \sum_{j=1}^{m} w_{i,j} \tag{6}$$

where, $w_{i,\,j}$ represents the computation cost of the task n_i on the processor p_j.

The next attributes to be calculated are *DTC* and rank of predecessor task (*RPT*) [5]. These three attributes of a given task are summed up and assigned as the rank of that task. The idea behind adding these three attributes is, the task with more successors gets higher priority so that its successors need not have to wait for a longer time to start its execution.

The *DTC* is data transfer cost. The DTC of a task n_i is the amount of communication cost required to transfer the data from task n_i to all its immediate successor tasks. It is computed using Eq. (7) given below

$$DTC(n_i) = \sum_{j=1}^{n} c_{i,j} \quad i < j \tag{7}$$

where n is the number of immediate successors of a task n_i and *DTC* is zero for exit task.

The *RPT* of task n_i is calculated by taking the highest rank of all its immediate predecessor tasks.

$$RPT(n_i) = Max\{rank(n_1), rank(n_2) \ldots rank(n_h)\} \tag{8}$$

where, $n_{1,}n_{2\ldots}n_h$ are the immediate predecessors of task n_i. As the entry task has no predecessor task its *RTP* value is considered as zero. To calculate the rank of a task, the maximum rank of predecessor tasks is considered as one of the parameters. Based on the values of the *CEC*, *DTC*, and *RPT*, the rank is calculated for each task n_i.

$$rank(n_i) = CEC(n_i) + DTC(n_i) + RPT(n_i) \tag{9}$$

For example, to calculate the rank of the task n_1, *CEC*(n_i)is calculated as shown in Eq. (7),

CEC(n_i)=14 + 16 + 9 =39. The *DTC* is the communication cost incurred to transfer data to their immediate successors that are from n_1 to n_2, n_3, n_4, $n_{5,}$ and n_6. In this given example *DTC* of task n_1 is 64 that is (18 + 12 + 9 + 11 + 14) and *RPT* is the rank of the predecessor task, as task n_1 has no predecessor, its *RPT* value is 0. The above-calculated values are summed up and assigned as the rank of the task n_1. The rank of the task n_1 is 103. Next, ranks of the tasks n_2, n_3, $n_{4,}$ $n_{5,}$ and n_6 are calculated using Eq. (9). Now to calculate the rank of the task n_8 three attributes have to be summed up: (1) the *CEC* of the task n_8 − 30, (2) the *DTC* value of task n_8 − 11, and (3) *RPT* value of the task n_8 − 232. To calculate the RPT value of task n_8, the highest rank of its immediate predecessor tasks should be considered. The task

n_8 receives input from tasks n_2, n_4, n_6. The calculated rank of task n_2, task $n_{4,}$ and task n_6 are 188, 191, and 156 respectively. The *RPT* is the maximum rank value of its immediate predecessors. Therefore, the *RPT* value of the task n_8-191, as the task n_4 has the maximum rank value when compared to the task n_2 and n_6. The ranks are arranged in descending order at each level and assigned, the task with the highest rank gets higher priority at each level. The scheduling order of the proposed algorithm is {$n_{1,}$ $n_{4,}$ $n_{2,}$ $n_{3,}$ $n_{6,}$ $n_{5,}$ $n_{9,}$ $n_{8,}$ $n_{7,}$ and n_{10}}. The calculated values are shown in Table 3

4.3 Processor selection phase

In this phase, the tasks are assigned to their suitable processors to further minimize the overall scheduling length. The proposed algorithm follows the non-crossover technique [7] in phase for processor selection. This technique enhances the processor selection phase and facilitates reducing the makespan. Most of the list scheduling algorithms do not follow the non-crossover technique. For example, the HEFT algorithm assigns the task to the processor which has the least EFT (local optimal). On the other hand, it doesn't consider the computation cost of the processor. This leads to a cross-over of tasks from one processor to another processor to gets its execution, often resulted in increased makespan. This limitation can be overcome by the non-crossover approach. The non-crossover approach is a technique that will decide whether a task has been executed on the processor which has the least EFT (local optimal) or on the processor which has the least computation cost (global optimal) to further reduce the makespan. The non-crossover decision is made based on the cross-threshold value. The proposed algorithm assigns the cross_threshold value from 0.0 to 0.3 range of value as this value shows consistent performance over a huge variety of graphs [7]. If the cross_threshold value is fixed to 1, then the algorithm resembles more like HEFT that is more cross over between processors on the other hand opting zero, no cross over between the processors.

The entry task is assigned to the processor which has the least EFT. After it finished its execution the task which has the next highest priority is taken from the ReadyTask-List and their EFT values for all the processors are calculated. Now the computation cost of that task on the processor which produces the least EFT is compared with the computation cost of the same task on all other processors. If both the finishing time and the computation cost of a given task are low on the processor then the task is assigned to that processor. In contrast, the cross-threshold value is calculated. Based on the resulted value, cross-over the decision has to be made. If the cross_threshold value is between 0.0 and 0.3 then the task has to be executed on the processor which has the least EFT, otherwise, the task gets its execution on the processor which has the least computation cost.

The Cross-Threshold value is the ratio of $weight(n_i)$ to $weigh_{abstract}$. The $weight(n_i)$ is calculated using Eq. (10). It is the ratio of the absolute time difference of the computation cost of a given task on different processors to the speedup value of that task. The speedup value is the ratio of the execution time on the slower processor to the faster processor. The $weight_{abstract}$ is the ratio of the time difference

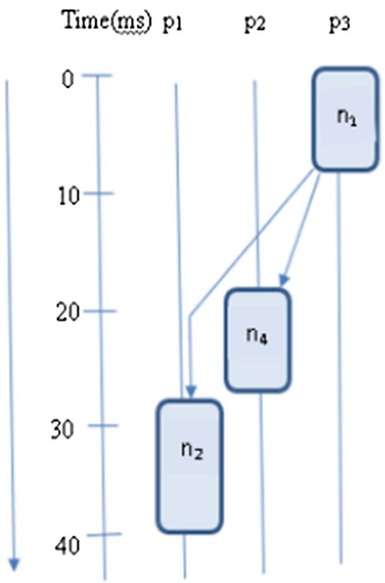

Fig. 5 Scheduling diagram of tasks n_1, n_4, n_2

Table 3 Prioritization of the tasks given in Fig. 4 by the proposed algorithm

Level (l_i)	Task	*CEC*(n_i)	*DTC*(n_i)	*RPT*(n_i)	*rank*(n_i)	Priority
1	n_1	39	64	0	103	1
2	n_2	50	35	103	188	2
2	n_3	43	23	103	169	3
2	n_4	38	50	103	191	1
2	n_5	35	13	103	151	5
2	n_6	38	15	103	156	4
3	n_7	42	17	169	228	3
3	n_8	3o	11	191	232	2
3	n_9	50	13	191	254	1
4	n_{10}	51	0	254	305	1

between the EFT values of the task on the processors to the speedup value of the EFT. The $weight_{abstract}$ is calculated as shown in Eq. (11).

$$Weight\left(n_i\right) = \frac{\mathrm{w}(\mathrm{n_i}, \mathrm{p_j}) - \mathrm{w}(\mathrm{n_i}, \mathrm{p_k})}{\mathrm{w}(\mathrm{n_i}, \mathrm{p_j})/\mathrm{w}(\mathrm{n_i}, \mathrm{p_k})} \tag{10}$$

Here w(ni, pj) is defined as the computation cost of task ni on processor pj.

$$Weight_{abstract} = \frac{\mathrm{EFT}(\mathrm{n_i}, \mathrm{p_j}) - \mathrm{EFT}(\mathrm{n_i}, \mathrm{p_k})}{\mathrm{EFT}(\mathrm{n_i}, \mathrm{p_j})/\mathrm{EFT}(\mathrm{n_i}, \mathrm{p_k})} \tag{11}$$

Here EFT(n_i, p_j) is defined as the earliest finishing time of task n_i on processor p_j.

Next, the cross-threshold value is computed based on the following equation given below.

$$\text{Cross\_threshold} = \frac{weight\left(n_i\right)}{weight_{abstract}} \tag{12}$$

The cross over of Tsk from one processor to another processor is based on the value of cross_threshold value.

The processor selection phase starts its execution from the entry task. A sample scheduling diagram is given in Fig. 5. As all the processors are available for execution the entry task starts its execution from zero. The EST value is calculated from Eq. (3) and EFT value is calculated from Eq. (4). The EFT value of the entry task on all three processors is calculated as, EFT $(n_{1,}\ p_1) = 14$ and EFT $(n_{1,}\ p_2) = 16$, and EFT$(n_{1,}\ p_2) = 9$. Since the processor p_3 has the least EFT, Processor p_3 is selected for executing task n_1. Now the next task to be executed in the priority list is task n_4 and its EST $(n_{4,}\ p_1) = 18$, EST $(n_{4,}\ p_2) = 18$, EST $(n_{4,}\ p_3) = 9$, EFT $(n_{4,}\ p_1) = 31$ and EFT $(n_{4,}\ p_2) = 26$ and EFT $(n_{4,}\ p_3) = 26$ values are calculated. Now the finishing time of both processors p_2 and p_3 are the same but the computation time of the p_2 is less when compared to p_3. so task n_4 is assigned to p_2. Since both the finishing time and execution time of processor p_1 is high when compared to p_2 and p_3, p_1 is not considered for execution. The next task in the priority queue is task n_2. To schedule, task n_2 its EST and EFT are calculated. They are EST $(n_{2,}\ p_1) = 27$, EST $(n_{2,}\ p_2) = 27$, EST $(n_2, p_3) = 9$, EFT $(n_2, p_1) = 40$, EFT $(n_2, p_1) = 46$ and EFT $(n_2, p_3) = 27$. Now the decision has to be made for assigning task n_2 on a suitable processor because p_3 has the least finishing time on the other hand p_1 has the least computation cost. The earliest finishing time of task n_2 on the processor p_3 is minimum when compared to the processor p_1, but the computation cost of the processor p_1 to execute the task n_2 is13 whereas the computation cost of the processor p_3 is 18. Even though the processor p_3 has the least EFT, its computation cost to execute task n_2 is high. Now the decision has to be made on which processor the task n_2 has to be executed, whether task n_2 is to be executed on the processor $p_1$or p_3. To make the decision Cross_Threshold value is calculated as shown in the Eq. (12)

If the resulted value is between 0.0 and 0.3 the task is assigned to the processor which has the least earlier finishing time. If the value is above 0.3, then the task is executed on the processor which has the least computational cost. The Cross_Threshold value of task $n_2 = weight(n_i)/weight_{abstract} = \frac{\frac{18-13}{(18/13)}}{\left(\frac{40-27}{\frac{40}{27}}\right)} = 0.41$. The calculated value is greater than 0.3 so. task n_2 gets its execution on the processor p_1 which has the least computation cost. The next task to be executed is task $n_{3.}$the EST $(t_3, p_1) = 40$, EST $(n_3, p_2) = 26$, EST $(n_3, p_3) = 9$, EFT $(n_3, p_1) = 51$ and EFT $(n_3, p_2) = 39$ and EFT $(n_3, p_3) = 28$. Now the decision has to be made on which processor task n_3 has to be executed to minimize the makespan. as the processor p_1 has the least computation cost and processor p_3 has the earlier finishing time. The cross_threshold value of task $n_3 = \frac{\frac{19-11}{(19/11)}}{\frac{51-28}{(51/28)}} = 0.36$. The resulted value is greater than 0.3 so the task n_3 can be assigned to processor p_1 which has the least computation cost. But the processor p_2 has the minimum earliest finishing time when compared to p_1 so the cross_threshold value between p_1 and p_2 is to be calculated to finalize the decision. The cross_threshold value of the processor p_1 and p_2 is 0.18 (which is between 0.0 and 0.3) so the task n_3 is assigned to processor $p_{2.}$ Following the same procedure rest of the tasks are scheduled to their suitable processor. As shown in Fig. 6 the tasks n_2 and n_8 are executed on the processor p_3, tasks n_4, n_3, n_7, $n_{9,}$ and nt_{10} are executed on the processor $p_{2,}$ and task $n_{1,}$ $n_{5,}$ and n_6 are scheduled on processor $p_{3.}$ Due to this balanced decision of assigning the task between the processors, the proposed algorithm gains the lowest makespan, whereas the makespan of HEFT-80, makespan of PEFT-85, makespan of MOPT-85, and SDBATS-88. The processor selection is shown in Tables 4 and 5 shows the total computation cost spent to execute all the tasks, the proposed algorithm has the least Overall computation cost on processors, Overall computation cost of the proposed algorithm-92, HEFT-110, PEFT-128, SDBATS-154, and MOPT-176, respectively.

4.4 Flowchart of the proposed algorithm

See Fig. 7.

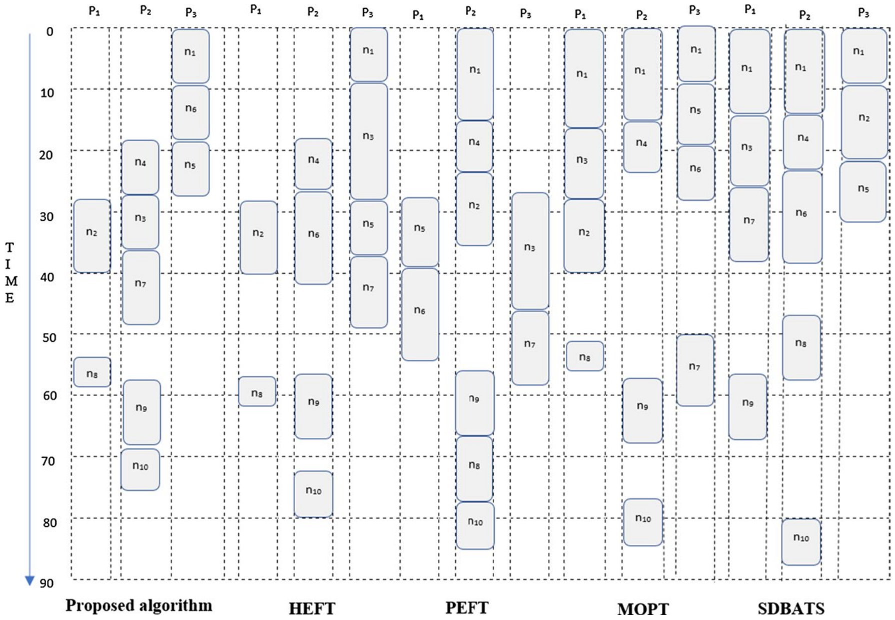

Fig. 6 Scheduling diagram for Fig. 4

Table 4 Processor selected by the proposed algorithm

Task	*rank*(n_i)	Processor selected	Earliest start time (EST)	Earliest finish time (EFT)
n_1	103	p_1	0	9
n_4	191	p_2	18	26
n_2	188	p_3	27	40
n_3	169	p_2	26	39
n_6	156	p_1	9	18
n_5	151	p_1	18	28
n_9	254	p_2	56	68
n_8	232	p_1	53	58
n_7	228	p_2	39	54
n_{10}	305	p_2	69	76

5 Simulation experiment and result discussion

The main purpose of this experiment is to verify the performance of the proposed algorithm with other existing algorithms like PEFT, HEFT, MOPT, and SDBATS.

5.1 Simulation environment

The proposed algorithm was implemented using iFogSim. iFogSim is a simulation framework that enables us to simulate, model, and experiment with the fog system. The simulation environment consists of Nodes which are the processing nodes capable of executing the incoming tasks. The fog nodes are heterogeneous. Each fog node can perform the execution of tasks at the same can communicate with other fog nodes. The fog nodes are connected with high-speed networks and all the tasks are non -preemptive on the nodes. The random graph generator is used to produce a huge variety of graphs that serve input to the fog nodes. The proposed algorithm was written in the Java programing language. It has been simulated on Intel Core i5 Processor, 2.3 GHz machine having 3 MB of L3 cache and 4 GB of RAM running Mac OS, Eclipse IDE, and iFogSim Toolkit.

5.2 Comparison metrics

Some of the performance metrics used in this experiment for comparing the algorithms are given below.

Table 5 Results of scheduling DAG

	Proposed algorithm	HEFT	PEFT	MOPT	SDBATS
Task prioritizing	n_1, n_4, n_2, n_3, n_6, n_5, n_9, n_8, n_7, n_{10}	n_1, n_3, n_4, n_2, n_5, n_6, n_7, n_9, n_8, n_{10}	n_1, n_4, n_2, n_3, n_5, n_6, n_7, n_9, n_8, n_{10}	n_1, n_5, n_4, n_6, n_3, n_2, n_8, n_7, n_9, n_{10}	n_1, n_2, n_3, n_4, n_7, n_5, n_6, n_8, n_9, n_{10}
Makespan	76	80	85	85	88
Overall computation cost	92	110	128	167	154

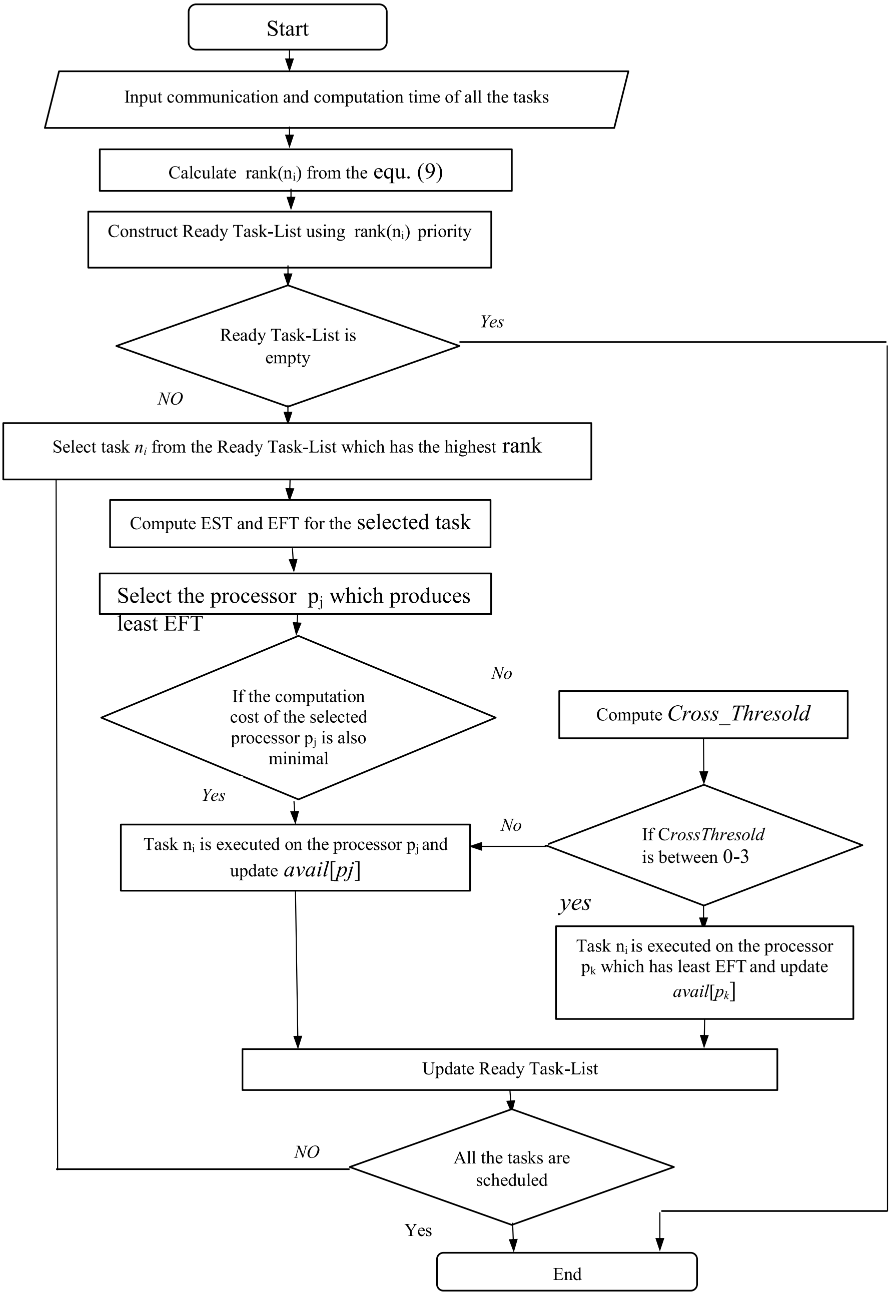
Start
Input communication and computation time of all the tasks
Calculate rank(n_i) from the equ. (9)
Construct Ready Task-List using rank(n_i) priority
Ready Task-List is empty
Yes
NO
Select task n_i from the Ready Task-List which has the highest rank
Compute EST and EFT for the selected task
Select the processor p_j which produces least EFT
If the computation cost of the selected processor p_j is also minimal
No
Yes
Compute Cross_Thresold
If CrossThresold is between 0-3
No
yes
Task n_i is executed on the processor p_j and update avail[pj]
Task n_i is executed on the processor p_k which has least EFT and update avail[p_k]
Update Ready Task-List
All the tasks are scheduled
NO
Yes
End

Fig. 7 Flow chart for the proposed algorithm

5.2.1 Makespan

Makespan is the most likely expected metric in scheduling problems because using this value other metrics like SLR, Speeup ratio also calculated. Makespan in the execution finishing time of the exit task.

5.2.2 Scheduling length ratio (SLR)

It is the most preferable performance measure of a scheduling algorithm. Normalizing the scheduling length to a lower bound is called scheduling length ratio (SLR). It is the ratio between the makespan and critical path including communication (CPIC). The critical path is the longest path from the entry node to the exit node of the given DAG on the fastest processor. The makespan is the actual finish time of the exit node in the given DAG. Since the denominator is the lowest bound, the value of the SLR cannot be lower than 1. The task scheduling algorithm, which has the least SLR value, is considered the best performing algorithm.

$$SLR = \frac{makespan}{\sum_{n_i \epsilon CP_{MIN}} \min_{p_j \in \mathrm{Q}} \{w_{i,j}\}} \tag{13}$$

The denominator is the summation of the minimal computation cost of the tasks on the critical path(CP).

5.2.3 Speedup

It is the ratio between the sequential execution time to the parallel execution time. The sequential execution time is assigning all the tasks to a single processor which minimizes the overall cumulative computation costs. The parallel execution time is the makespan

$$Speedup = \frac{\min_{p_j \in \mathrm{Q}} \{\sum_{n_i \in N} \{w_{i,j}\}\}}{makespan} \tag{14}$$

5.2.4 Number of occurrences of better quality schedules

This is the count of the number of times an algorithm schedules better, equal, or worst when compared with other algorithms.

5.2.5 Average running time of the algorithms

It refers to the execution time for obtaining the output schedule of a given DAG. The lower the values better the algorithm is.

5.3 Random graph generator (RGG)

This Random Graph Generator helps in generating a huge variety of graphs that help this experiment to test the algorithm in a more complex environment. Some of the inputs to the graphs are given below.

- *n*: total number of tasks in the graph(*n*);
- *Shape* (α): the shape of the graph depends upon the value of α, if the chosen α value is greater than 1.0 then a denser graph is generated or a longer graph by choosing the value α lesser than 1.0. The denser graph has a high degree of parallelism and a longer graph has a low degree of parallelism.
- *Density*: it indicates the number of out-degree from the given task. With varying the value of density the number of edges can be varied from few to large.
- *Jump*: It references to the connection of edge can go from level *i* to level $i+jump$
- *CCR* (communication to computation ratio): It is the ratio of the sum of the weights of the edges to the sum of the weights of the nodes. If CCR = 0.1 then it is a computationally intensive application. If CCR = 10.0 then is a data-intensive application.
- *β*: This value indicates the heterogeneity factor of the processors. It is the range percentage of the computation costs on the processors. By varying the β value, the heterogeneity factor of the tasks can be varied. For a high *β* value, graph with the huge difference in computing cost among the tasks are generated and for a lower *β* value, the tasks with the same computing cost are generated

In this experiment around 5000 graphs are generated and the algorithms are implemented and results are observed. Different varieties of graphs were generated by setting the parameters in different combinations given below.

$v = \{10, 30, 50, 100, 200, 300, 500, 1000\}$
$\alpha = \{0.1, 0.5, 1.0, 2.0\}$
$density = \{ 1, 2, 3, 4, 5\}$
jump = {1, 2, 4}
CCR = {0.1, 0.5,1.0, 5.0, 10.0}
$\beta = \{0.1, 0.5, 1, 2\}$
P = {2, 4, 12, 24}

By varying the parameters mentioned above about 9000 graphs can be generated. To conduct this experiment about 10 DAG for each set with different tasks and edge weights are generated so a total of 90,000 different graphs set are generated and scheduling algorithms are executed. The performance metrics are calculated for all the stated algorithms and finally, the results are compared among them.

5.4 Performance results

This section of this paper deals with the performance comparison of the proposed algorithm with other algorithms like HEFT, PEFT, MOPT, and SDBATS concerning various graph characteristics like SLR, Speedup, Makespan, and algorithm running time.

5.4.1 Average scheduling length ratio

The denominator of the SLR is the lowest computation cost of the tasks in the critical path. The makespan is always greater than the denominator of the SLR equation so the algorithm with a lower SLR value is the best. As the SLR is the key factor in deciding the performance of an algorithm, this experiment is carried out against different factors like different task sets, CCR, and heterogeneity factor. To assess the performance of the proposed algorithm a huge variety of graphs with different task sets (10, 30, 50, 100, 200, 300, 500, 1000) are generated and the obtained average SLR values for all the algorithms are plotted and are shown in Fig. 8.

The next experiment is concerning various CCR values. The graphs with 0.1, 0.5, 1.0, 5.0, 10.0 values of CCR are generated and the outcome results for all the algorithms are noted. It is observed that the proposed algorithm consistently outperforms all other algorithms. Figure 9 shows the average SLR value against different CCR values for all the algorithms. The results show that the proposed algorithm has the lowest average SLR value for all the CCR values. For the CCR value of 0.1 the proposed algorithm is 6% better than PEFT, 13% better than HEFT, 26% better than MOPT, and 39% better than SDBATS algorithms. For a CCR Value of 10, the

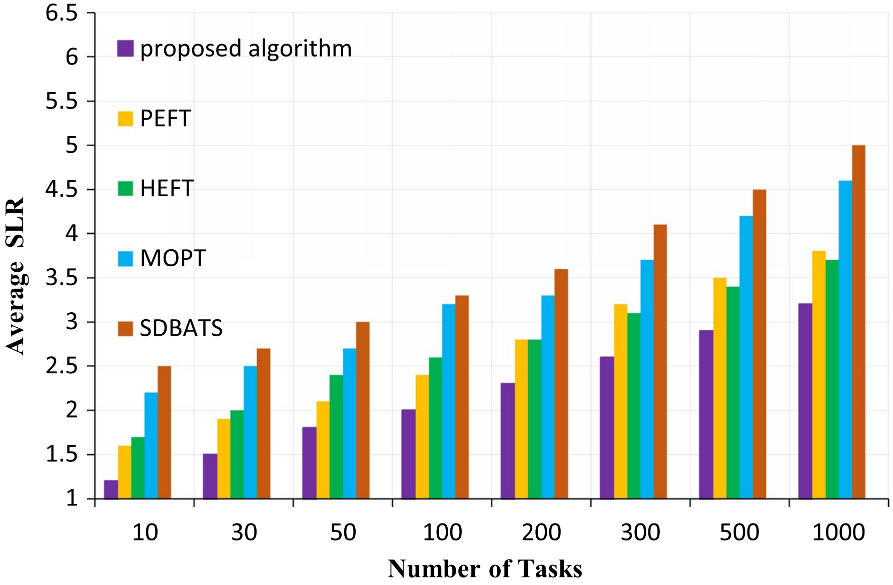

Fig. 8 Average SLR for the different task set

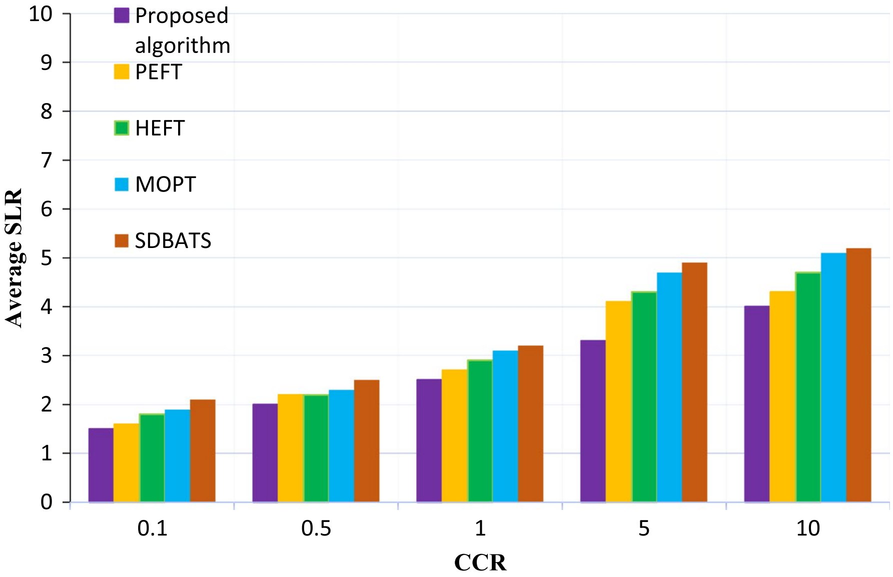

Fig. 9 Average SLR for various CCR

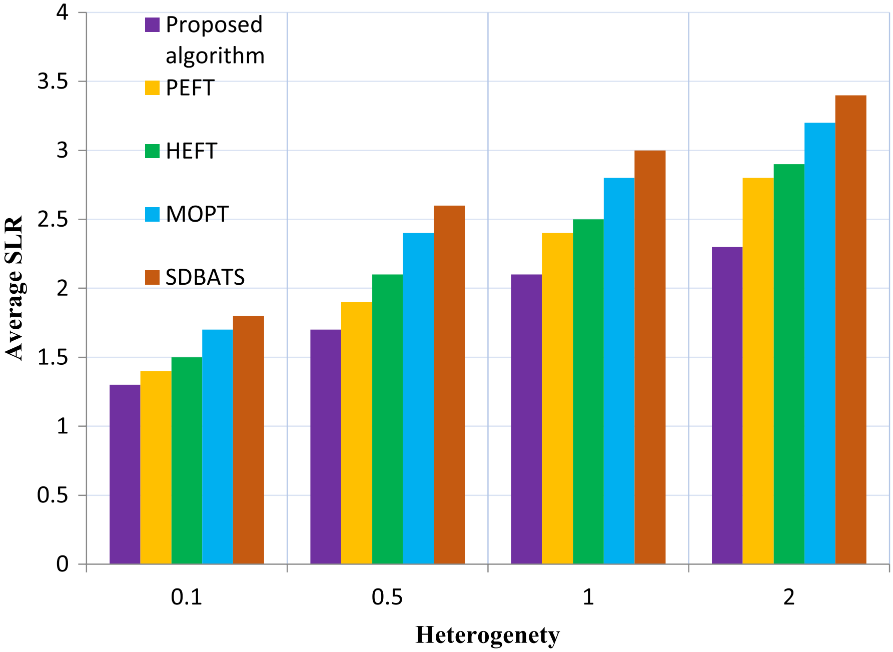

Fig. 10 Average SLR for various heterogeneous factors

proposed algorithm performs better than PEFT by 7%, HEFT by 17%, MOPT by 27%, and SDBATS by 30%. It is noticed that the increase in the CCR value didn't affect the performance of the proposed.

Another experiment is conducted to measure the quality of scheduling of the proposed algorithm against the heterogeneity factor of the graph. The varying value of β will affect the heterogeneity of a graph. A greater value of β will generate a high heterogeneous graph. In this experiment, four different values of β are taken (0.1, 0.5, 1, and 2). It is observed that for $\beta=0.1$, the proposed algorithm is better than the PEFT algorithm by 7%, the HEFT algorithm by 15%, the MOPT algorithm by 30%, and the SDBATS algorithm by 39%. For $\beta,=0.2$, the proposed algorithm is better than the PEFT algorithm by 21%, the HEFT algorithm by 25%, the MOPT algorithm by 38%, and the SDBATS algorithm by 47%. For all the four values of β, the proposed algorithm outperforms all other algorithms as shown in Fig. 10. The proposed algorithm produced the lowest value of SLR when compared to other stated algorithm is due to the balanced decision of cross over technique followed by it. Even though the SDBATS algorithm follows an entry-level task duplication policy, comparatively it has the highest value of SLR.

The Speedup is also one of the performance deciding factors of an algorithm. The speedup is the ratio of sequential executing time to the parallel execution time. The sequential execution time is the cumulative computation time taken by the tasks on the fastest processor. The speedup value of a good algorithm should be always high. To evaluate the proposed algorithm in terms of speedup this experiment is being conducted. For this, a huge variety of graphs with task sets varied from 10 to 1000 are generated and all the stated algorithms are executed. The speedup value for all the stated algorithms is computed and it is shown in Fig. 11. The results show that

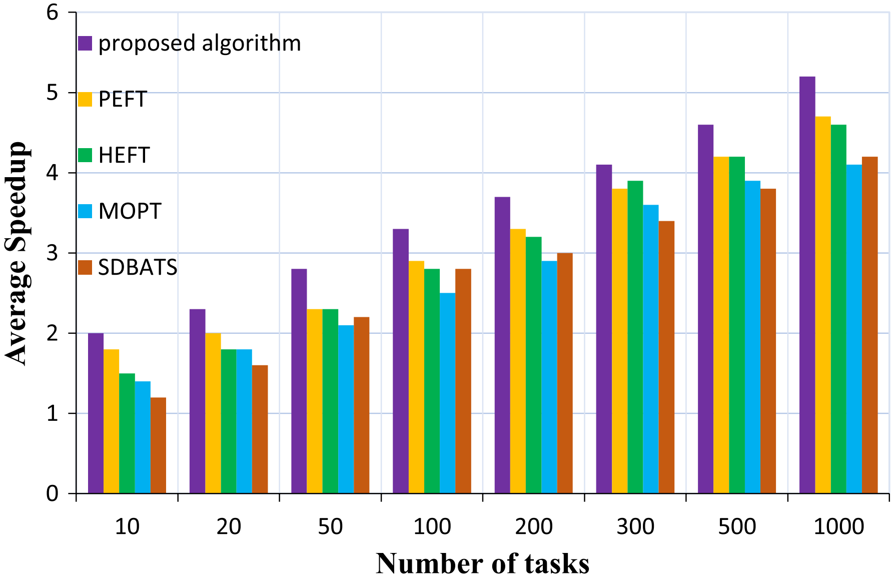

Fig. 11 Average speedup value for the different task set

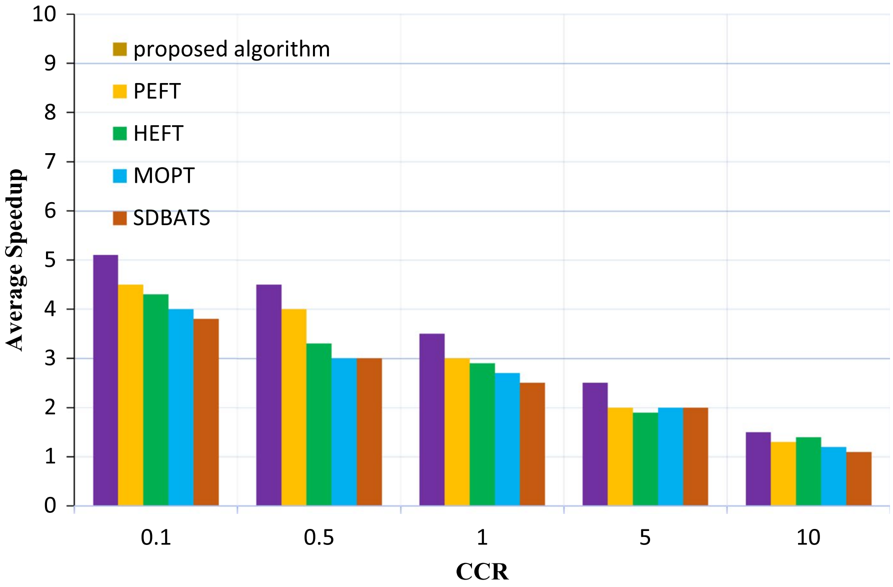

Fig. 12 Average Speedup value for various CCR

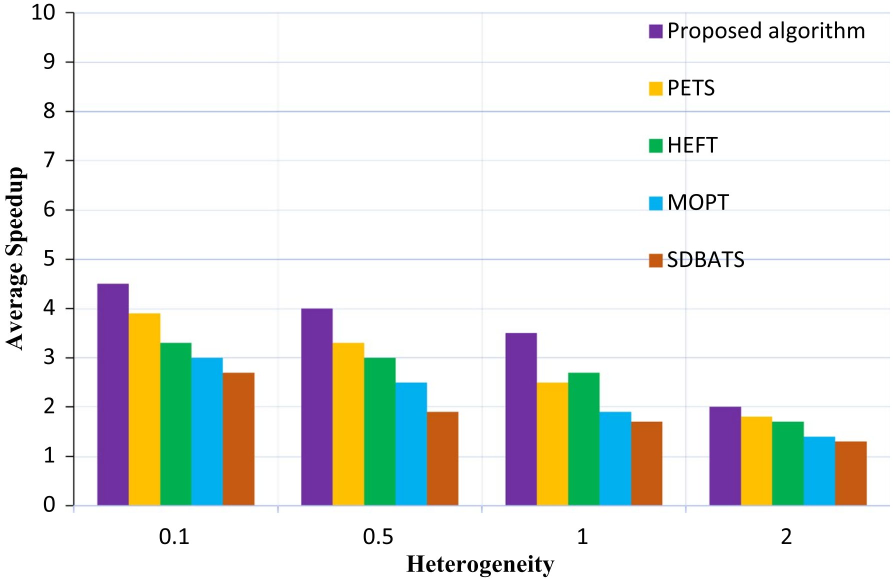

Fig. 13 Average speedup for various heterogeneity factors

the proposed algorithm is better than the PEFT algorithm by 11%, HEFT algorithm by 13%, SDBATS algorithm by 17% MOPT algorithm by 20%.

Figure 12 shows the speedup value concerning various CCR values. For the value, CCR = 0.1 the proposed algorithm shows improvement in speedup value by

11% for PEFT, 15% for HEFT, 20% for MOPT, and 24% for SDBATS. The results show that the Proposed algorithm has the highest speedup value when compared to all other mention algorithms. This improvement in the performance in speedup value against CCR is due to the restriction in the number of the crossover which limits the data transferred between the processors.

The next experiment is being conducted concerning the heterogeneity of the graphs. The graphs with different heterogeneity (β) are generated and the speedup value of all the stated scheduling algorithms is calculated and is shown in Fig. 13. The speedup value of the proposed algorithm is noticeably high when compared to all other algorithms. It is 13 better than PEFT, 26% better than HEFT, 33% better than MOPT, and 40% better than SDBATS.

The outcome makespan of a scheduling algorithm should be always expected to be low because all other performance metrics depend on the makespan. The makespan is the overall scheduling length of the tasks. Figure 14 shows the average makespan concerning different task sets. The proposed algorithm has the lowest makespan when compared to all other algorithms. For the task set of 1000, the proposed algorithm performs better than PEFT by 9%, HEFT by 15, MOPT by 17%, and SDBATS by 33%. The gain in makespan is due to the optimized prioritizing phase of the proposed algorithm.

Figure 15 shows the average makespan concerning different CCR values. The proposed algorithm has the lowest makespan when compared to all other algorithms. For the highest CCR value = 10 the proposed algorithm is better than PEFT by 12%, HEFT by 10%, MOPT by 13%, and SDBATS by 21%. The proposed algorithm yields minimal makespan when compared to all other stated algorithms.

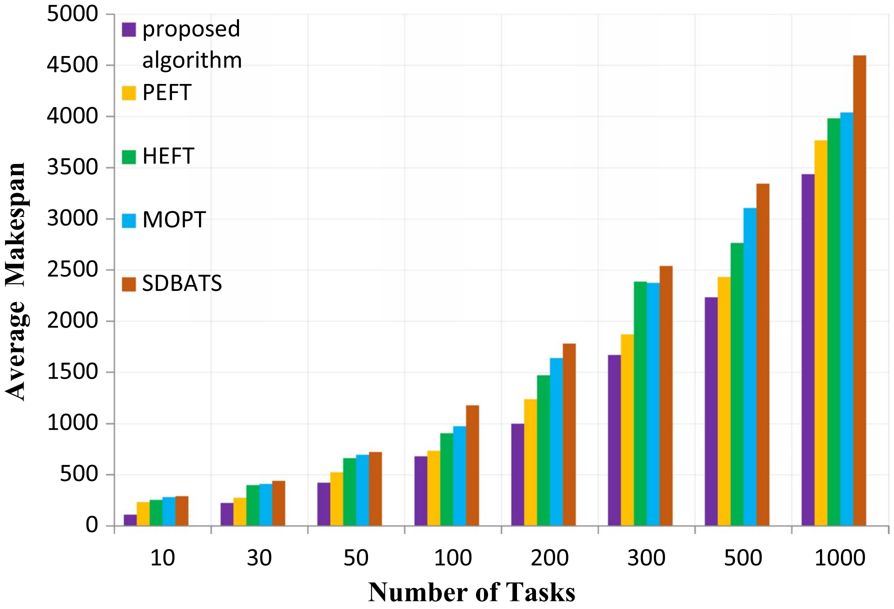

Fig. 14 Average makespans for the different task set

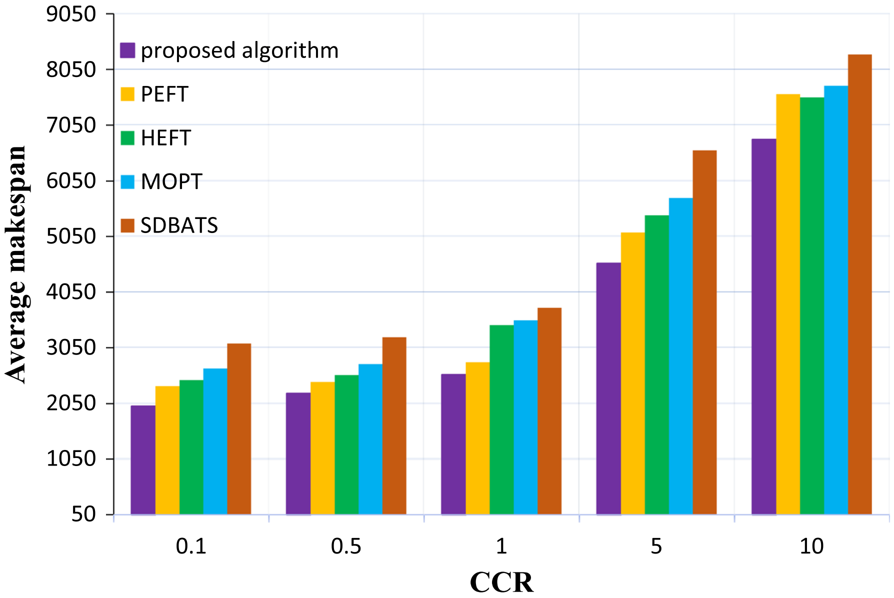

Fig. 15 Average makespan for different value of CCR

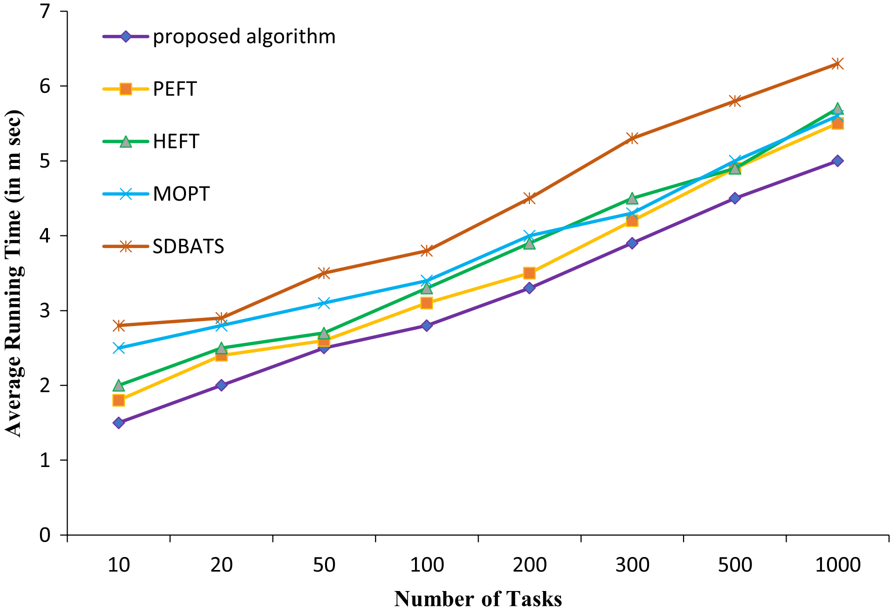

Fig. 16 Average running time for the different task set

The average running time of the algorithms is observed and plotted in the graph shown in Fig. 16. It is observed that the proposed algorithm is the fastest one when compared to other algorithms. The set of 1000 tasks are executed and the result shows that the proposed algorithm performs better than PEFT by 10%, HEFT by 14%, MOPT by 12%, and SDBATS by 24%.

5.4.2 Performance comparison of real-world scientific workflow

To further evaluate the performance of the proposed algorithm, the application of real-world problems like Gaussian elimination (GE), montage, epigenomic and are considered. In real-world applications, the structure of the application graph is known, other parameters like Shape parameters, out-degree, and density are not required. The same value of CCR and heterogeneity specified in Sect. 5.3 are considered for this experiment. On the other hand, a new parameter, matrix size(m) is used to calculate the number of tasks. The total number of tasks in the GE graphs is equal to $\frac{m^2+m-2}{2}$. In this experiment, the matrix size of 5 to 15 is considered. So that the graph with 14–119 task sets is considered to conduct the experiments. The average SLR as the function of matrix size is shown in Fig. 17. The proposed algorithm has the lowest SLR when compared to all other reported algorithms. For matrix size of 15 the proposed algorithm performs 5% better than the PEFT algorithm, 15% better than the HEFT algorithm, 22% better than the MOPT algorithm, and 35% better than the SDBATS algorithm.

Figure 18 shows the average SLR value against various CCR values is calculated. It is observed that the proposed algorithm outruns all other stated algorithms for all the values of CCR.

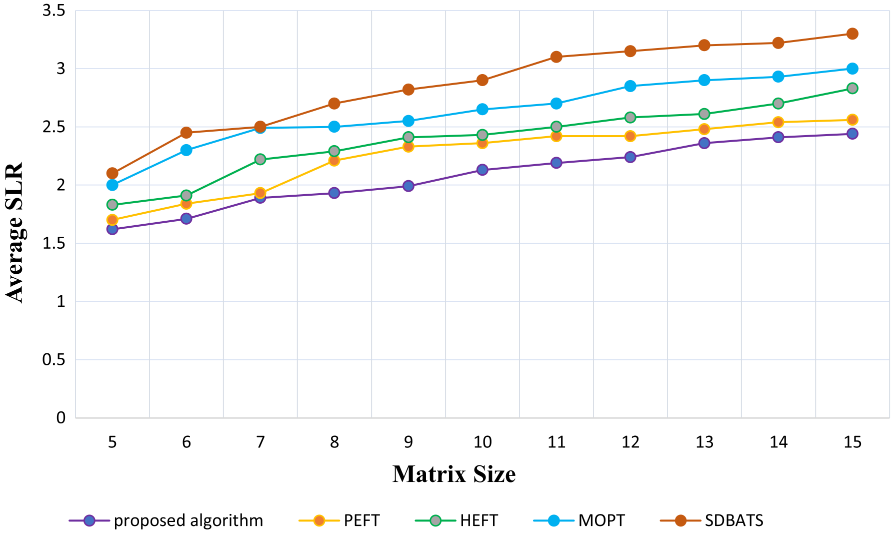

Fig. 17 Gaussian elimination application workflow: average SLR value for different matrix size

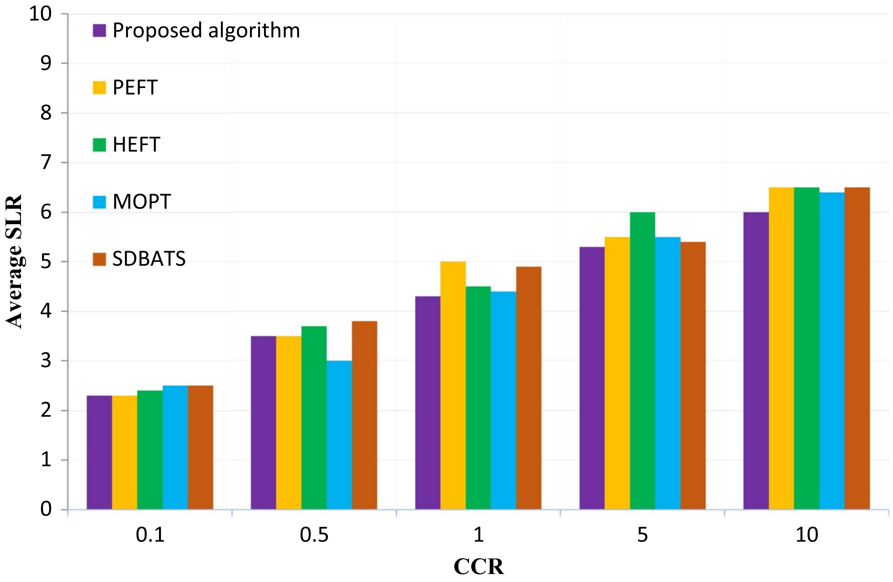

Fig. 18 Gaussian elimination application workflow: average SLR value for different CCR

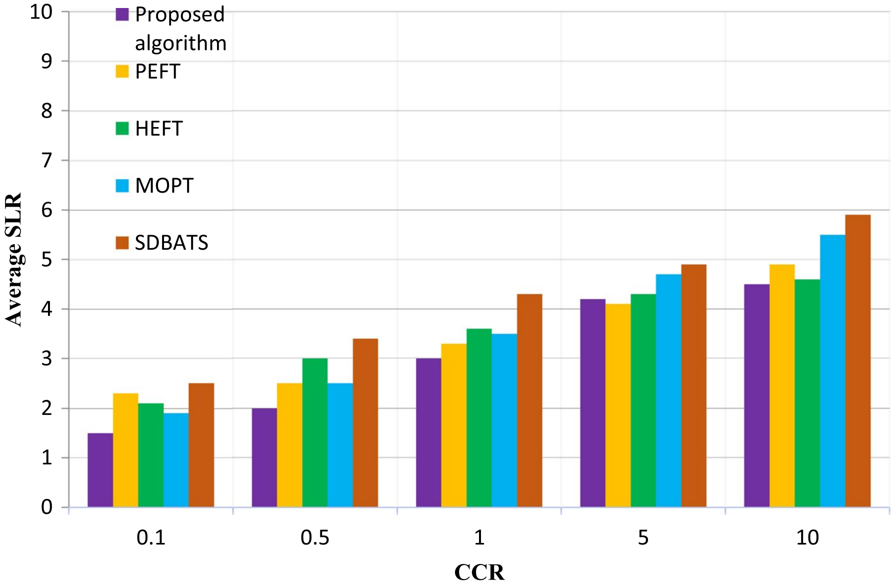

Fig. 19 Montage workflow: average SLR value against various CCR

Montage is another real-world application that is used to custom an astronomical image mosaic of the sky. The Montage of 23,50,100 tasks is used in this experiment. Figure 19 shows the average SLR value as the function for different

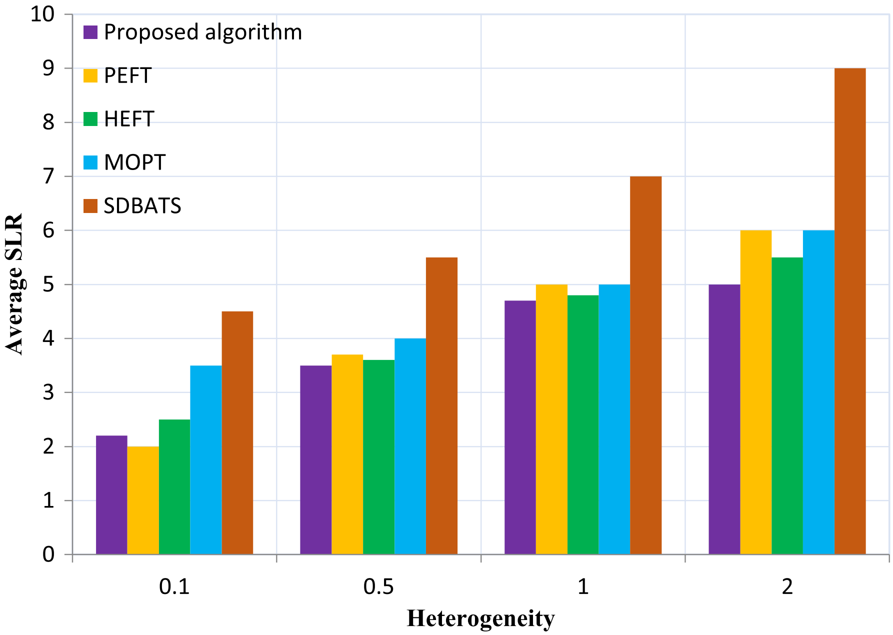

Fig. 20 Montage workflow: average SLR value against different heterogeneity factors

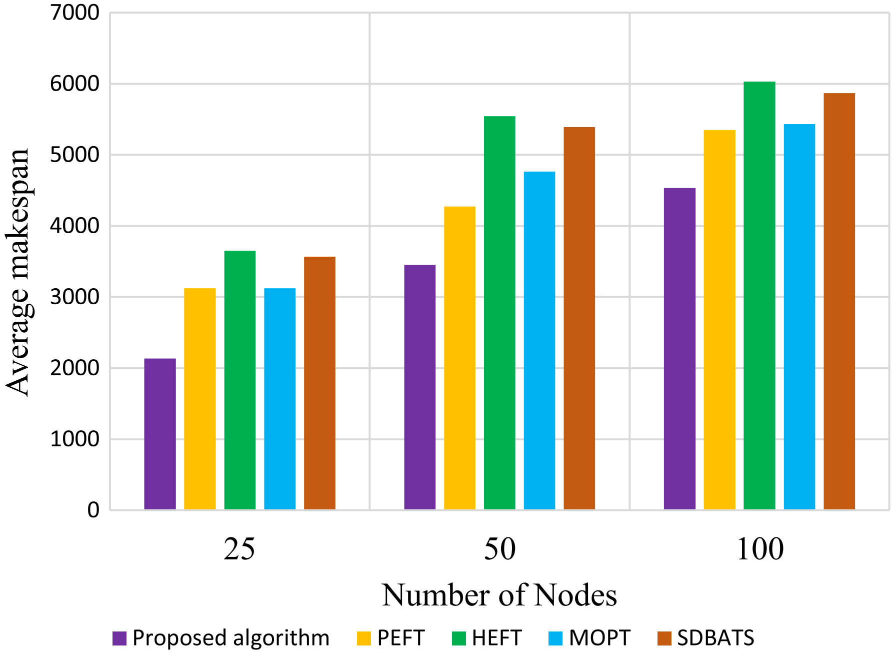

Fig. 21 Montage workflow: average makespan

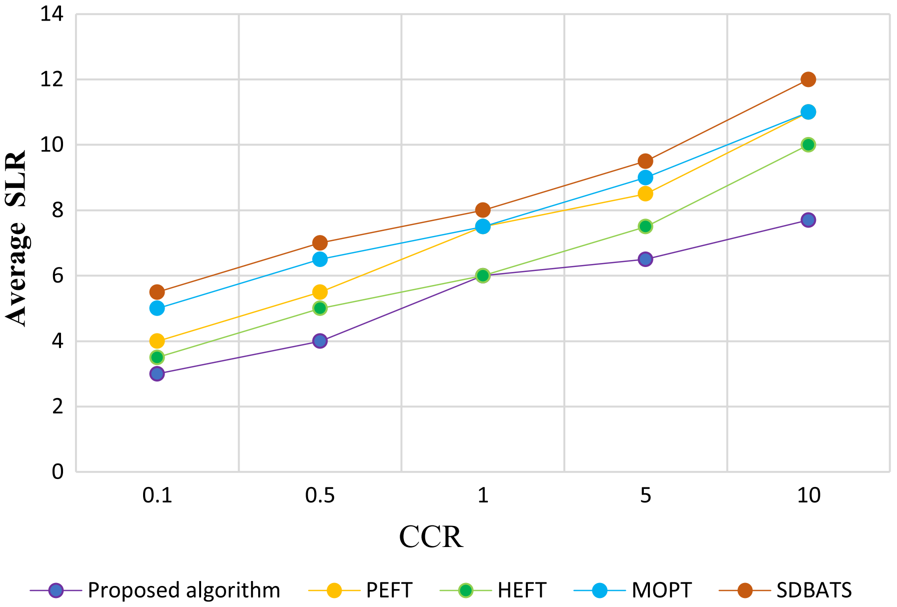

Fig. 22 Epigenomic workflow: average SLR for different CCR values

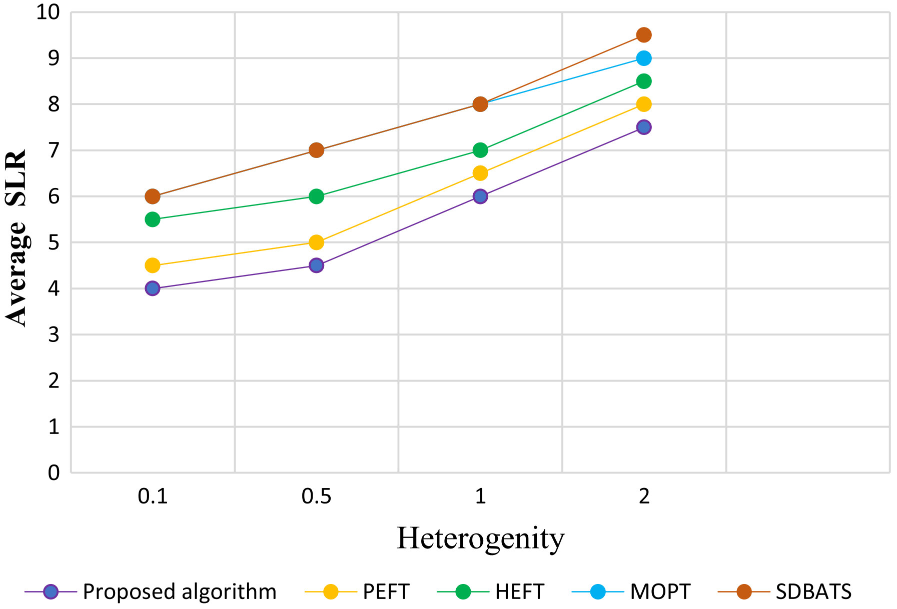

Fig. 23 Epigenomic workflow: SLR for various heterogeneity factor

CCR values. For CCR = 10 the average SLR value of the proposed algorithm is better than PEFT by 20%, HEFT by 2%, MOPT by 22%, and SDBATS by 31%.

The results in Fig. 20 show the obtained average SLR value against different heterogeneity parameters. When $\beta = 2$ the proposed algorithm performs better than PEFT by 20%, HEFT by 10%, MOPT by 20%, and SDBATS by 80%.

In the next experiment average makespan is calculated for different montage task sets. The proposed algorithm has reduced schedules when all other algorithms are shown in Fig. 21. This reduced makespan is due to the balanced crossover decision of the proposed algorithm.

Epigenomic Workflow is another real application workflow. It is used to compare the genetic performance of human cells on genome range. Like other real-time applications, the structure is known and different CCR, heterogeneity, and number of nodes are considered. Figure 22 shows for the CCR value of 10 the proposed algorithm is 42% better than PEFT, 29% better than HEFT, 42% better than MOPT, and 55% are better than SDBATS.

The average SLR value concerning various heterogeneity is shown in Fig. 23 respectively. For heterogenety = 2, the proposed algorithm is 6% better than PEFT, 13% better than HEFT, 20% better than MOPT, and 26% than SDBATS.

The final experiment is based on a cyber shake task graph. This workflow is used to characterize the earthquake hazard. Figure 24 shows the average makespan for various task sets. The proposed algorithm outruns all other algorithms with reduced scheduling time.

Figure 25 shows the average SSR results for various CCR values, for CCR value = 10 the proposed algorithm is 42% better than PEFT, 29% better than HEFT, 68% better than MOPT, and 55% better than SDBATS.

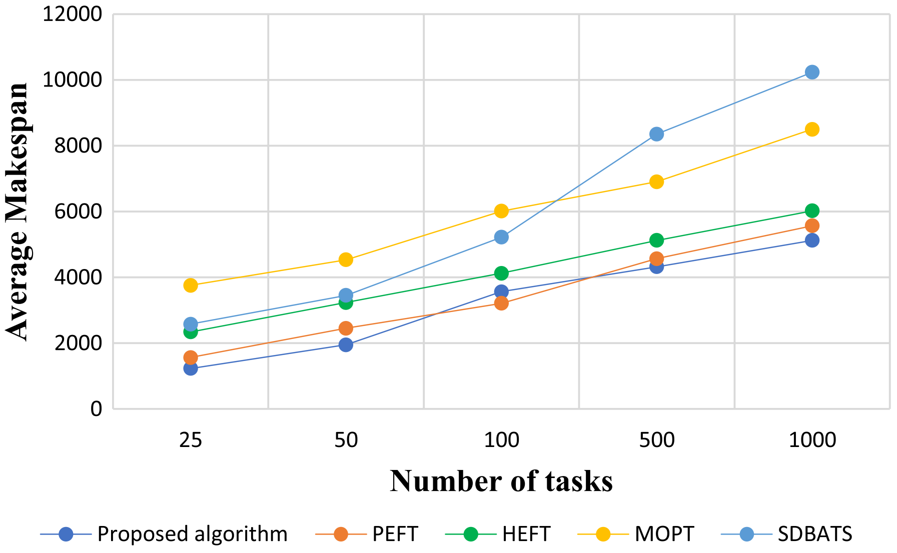

Fig. 24 Cyber shake workflow: SLR for the different task set

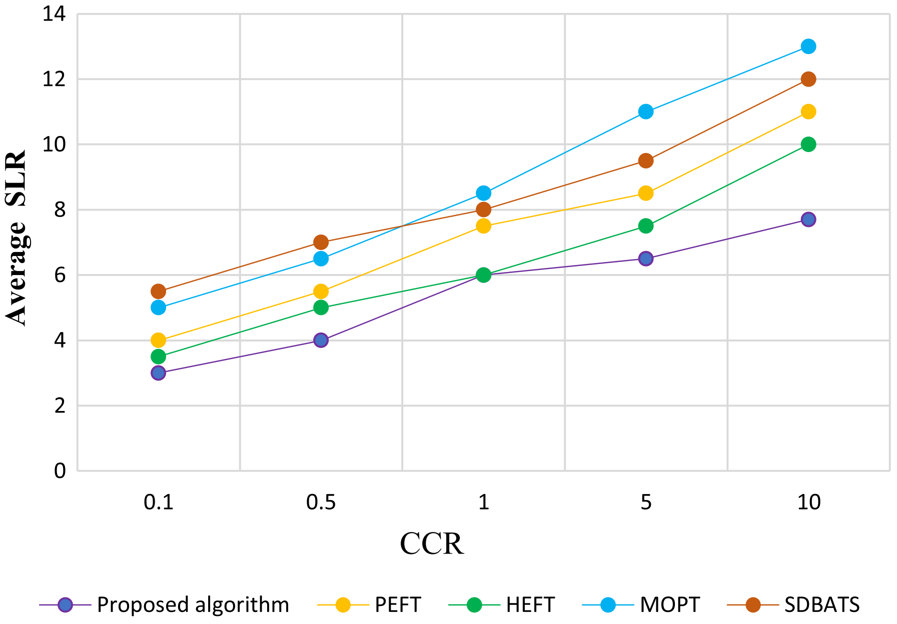

Fig. 25 Cyber shake workflow: SLR for various CCR factor

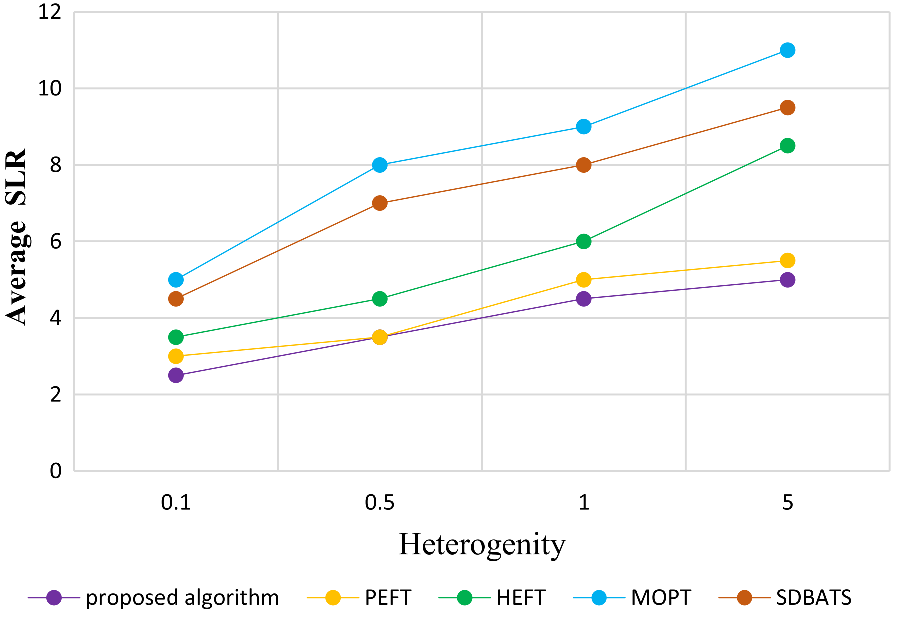

Fig. 26 Cybershake workflow: SLR for various Heterogeneity factor

The final experiment finding average SLR for various heterogeneity factors. Figure 26 shows the performance results of all the algorithms. The proposed algorithm outperforms all other stated algorithms.

From all the above experiments it is observed that the proposed algorithm exhibits sustainable performance improvement. Some significant feature of this algorithm is

- Tasks are prioritized accurately based on their significant.
- The balanced decision between local optimal and globally optimal.
- Restricted cross-over between processors.
- The computation cost of the processor is accessed while assigning a task.

Following these strategies, the proposed algorithm can effectively schedule the tasks in the workflow and can produce good performance matrices when compared to all other mention algorithms.

6 Conclusion and future work

In this paper, a new list-based task scheduling algorithm is proposed to schedule the DAG structured task in fog environment. This algorithm has three phases. The first phase facilitates the execution of tasks that are independent of each other and in the second phase, the task with more successor tasks is given higher priority so that more tasks in the next level need not have to wait. In the final phase decision between local optima and global optimal is balanced to further minimize the makespan and overall computation cost of the processors. To evaluate the performance of the algorithms random graph generator is used to produce more complex graph structures is also used. The experiment results show that the proposed algorithm is better than all other stated algorithms in terms of performance matrices. The complexity of this algorithm is O (n + e) (p + log n), where n is the number of tasks and p is the number of processors. Further, as future work, this approach can be extended for a dynamic network.

References

1. Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on mobile cloud computing, MCC '12, New York, NY, USA, 2012. ACM, pp 13–16
2. Agarwal S, Yadav S, Yadav A (2016) An efficient architecture and algorithm for resource provisioning in fog computing. MCEP. https://doi.org/10.5815/ijieeb.2016.01.06
3. Kwok Y, Ahmed I (1996) Dynamic critical-path scheduling: an effective technique for allocation task graphs to multi-processors. IEEE Trans Parallel Distrib Syst 7(5):506–521
4. Topcuoglu H, Hariri S, Wu M (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
5. Ilavarasan E, Thambidurai P, Mahilmannan R (2005) Performance effective task scheduling algorithm for heterogeneous computing system. In: The 4th internationalsymposium on parallel and distributed computing. IEEE, pp 28–38

6. Luiz F, Bittencourt RS, Edmundo RMM (2010) Dag cheduling using a lookahead variant of the heterogeneous earliestfinish time algorithm. In: 18th Euromicro international conferenceon parallel, distributed and network-based processing (PDP). IEEE, pp 27–34
7. Shetti KR, Fahmy SA, Bretschneider T (2013) Optimization of the HEFT algorithm for a CPU-GPU environment. In: IEEE parallel and distributed computing. applications and technologies (PDCAT). International conference on, pp 212–218
8. Arabnejad H, Barbosa JG (2014) List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans Parallel Distrib Syst 25(3):682–694
9. Hong H, Tsai P, Hsu C (2016) Dynamic module deployment in a fog computing platform. In: 2016 18th Asia–Pacific network operations and management symposium (APNOMS), Kanazawa, 2016, pp 1–6
10. Pham X, Huh E (2016) Towards task scheduling in a cloud-fog computing system. In: Proceedings of the 2016 18th Asia–Pacific network operations and management symposium (APNOMS), Kanazawa, Japan, 5–7 October 2016, pp 1–4
11. Taneja M, Davy A (2017) Resource aware placement of IoTapplication modules in fog-cloud computing paradigm. In: Integrated network and service management (IM), 2017 IFIP/IEEE symposium on.IEEE, pp 1222–1228
12. Yang Y, Zhao S, Zhang W, Chen Y, Luo X, Wang J (2018) DEBTS: delay energy balanced task scheduling in homogeneous fog networks. IEEE Internet Things J 5:2094–2106
13. Tejaswini C, Melody M, Teng-Sheng M (2018) Prioritized task scheduling in fog computing. ACM SE '18 March 29–31, 2018, Richmond, KY, USA
14. Amir K, Abdelhakim H, El Mostapha A (2019) On the fog-cloud cooperation: how fog computing can address latency concerns of IoT application. In: 2019 fourth international conference on fog and mobile edge computing (FMEC), IEEE, pp 166–172
15. Zahra R, Mahboobe R, Mohsen N (2019) LAMP: a hybrid fog-cloud latency-aware module placement algorithm for IoT applications. In: 5th conference on knowledge-based engineering and innovation (KBEI), Iran University of Science and Technology, IEEE, Tehran, Iran, pp 845–850
16. Shahzad Arif M, Iqbal Z, Tariq R, Aadil F, Awais M (2019) Parental prioritization-based task scheduling in heterogeneous systems. Arab J Sci Eng 44:3943–3952
17. Tang X, Li K, Liao G, Li R (2010) List scheduling with duplication for heterogeneous computing systems. J Parallel Distrib Comput 70(4):323–329
18. Ijaz S, Ullah Munir E (2009) MOPT: list-based heuristic for scheduling workfows in cloud environment. J Supercomput 75:3740–3768
19. Munir EU, Mohsin S, Hussain A, Nisar MW, Ali S (2013) SDBATS: a novel algorithm for task scheduling in heterogeneous computing systems. In: Proceedings of IEEE IPDPS workshops (IPDPSW), 2013
20. AlEbrahim S, Ahmad I (2017) Task scheduling for heterogeneous computing systems. J Supercomput 73:2313–2338
21. Ilavarasan E, Thambidura P (2007) Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. J Comput Sci 3(2):94–103
22. Ahmad I, Kwok YK (1998) On exploiting task duplication in parallel program scheduling. IEEE Trans Parallel Distrib Syst 9(9):872–892
23. Baskiyar S, Dickinson C (2005) Scheduling directed a-cyclic graph on a bounded set of heterogeneous processors using task duplication. J Parallel Distrib Comput 65:911–921
24. Agarwal A, Kumar P (2009) Economical duplication based task scheduling for heterogeneous and homogeneous computing systems. In: IEEE international advance computing conference, 2009, pp 6–7
25. Boeres C, Filho JV, Rebello VEF (2004) A cluster based strategy for scheduling task on heterogeneous processors. In: Proceedings of 16th symposium on computer architecture and high performance computing (SBAC-PAD), 2004, pp 214–221
26. Cirou B, Jeannot E (2001) Triplet: a clustering scheduling algorithm for heterogeneous systems. In: International conference on parallel processing workshops, pp 231–236
27. Kanemitsu H, Hanada M, Nakazato H (2016) Clustering-based task scheduling in a large number of heterogeneous processors. IEEE Trans Parallel Distrib Syst 27(11):3144–3157 ((**2**))
28. Bonomi F, Milito R, Natarajan P, Zhu J (2014) Fog computing: A platform for internet of things and analytics. In: Big data and internet of things: a roadmap for smart environments. Springer, pp 169–186

29. Masood A, Ullah Munir E, Mustafa Rafique M, Khan SU (2015) HETS: heterogeneous edge and task scheduling algorithm for heterogeneous computing systems. In: 2015 IEEE 17th international conference on high performance computing and communications (HPCC)
30. Singh S, Chiu Y, Tsai Y, Yang J (2016) Mobile edge fog computing in 5G era: architecture and implementation. In: IEEE international computer symposium (ICS), pp 731–735
31. Cisco Systems (2016) Fog computing and the internet of things: extend the cloud to where the things are, p 6. http://www.cisco.com. Accessed 10 Jan 2019
32. Sakellariou R, Zhao H (2004) A hybrid heuristic for dag scheduling on heterogeneous systems. In: 18th international symposium on parallel and distributed processing. IEEE, p 111
33. Guoqi X, Renfa L, Keqin L (2015) Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems. JPDC 83(C):1–12
34. Shirahata K, Sato H, Matsuoka S (2010) Hybrid map task scheduling for gpu-based heterogeneous clusters. In: Cloud computing technology and science (CloudCom), pp 733–740
35. Zhao H, Sakellariou R (2003) An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In: Euro-Par 2003. Parallel processing. Springer, pp 189–194
36. Ahmed A, Ahmed E (2016) A survey on mobile edge computing. In: Intelligent systems and control (ISCO). 10th international conference on. IEEE, pp 1–8
37. Satyanarayanan M (2017) The emergence of edge computing. Computer 50(1):30–39
38. Datta SK, Bonnet C, Haerri J (2015) Fog computing architecture to enable consumer centric internet of things services. In: International symposium on consumer electronics (ISCE), pp 1–2
39. Pahl C, Lee B, (2015) Containers and clusters for edge cloud architectures—a technology review. In: Future internet of things and cloud (FiCloud). 3rd international conference on. IEEE, pp 379–386
40. Tao Y, Gerasoulis A (1994) ADSC: scheduling parallel tasks on an unbounded number of processors. IEEE TPDS 5(9):951–967
41. Gulzar Ahmad S, Ullah Munir E, Nisar W (2011) A segmented approach for dag scheduling in heterogeneous environment. In: 12th international conference on parallel and distributed computing. Applications and technologies (PDCAT). IEEE, pp 362–367
42. Grewe D, O'Boyle MFP (2011) A static task partitioning approach for heterogeneous systems using opencl. In: Proceedings of the 20th international conference on compiler construction, vol 201. Springer, pp 286–305. https://doi.org/10.1007/978-3-642-19861-8_16
43. Canon L-C, Jeannot E, Sakellariou J, Zhang W (2008) Comparative evaluation of the robustness of dag scheduling heuristics. Grid Comput. https://doi.org/10.1007/978-0-387-09457-1_7
44. Chen H, Liu XZG, Pedrycz W (2017) Ushncertainty-aware online scheduling for real-time workflows in cloud service environment. IEEE Trans Serv Comput 10(6):929–941
45. Prasad Rima B, Maier M (2017) Workflow scheduling in multi-tenant cloud computing environments. IEEE Trans Parallel Distrib Syst 28(1):290–304
46. Rodriguez MA, Buyya R (2018) Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. Future Gener Comput Syst 79:739–750
47. Li X, Qian L, Ruiz R (2016) Cloud workflow scheduling with deadlines and time slot availability. IEEE Trans Serv Comput 11(2):329–340
48. Liu Z, Yang X, Yang Y, Wang K, Mao G (2019) DATS: dispersive stable task scheduling in heterogeneous fog networks. IEEE Internet Things J 6(2):3423–3436
49. Li H, Louis-Claude C, Henri C, Yves R, Frederic V (2018) Checkpointing workflows for fail-stop errors. IEEE Trans Comput 67(8):1105–1120
50. Vaquero LM, Rodero-Merino L (2014) Finding your way in the fog: towards a comprehensive definition of fog computing. SIGCOMM Comput Commun Rev 44(5):27–32

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

R. Madhura[1] · **B. Lydia Elizabeth**[2] · **V. Rhymend Uthariaraj**[1]

B. Lydia Elizabeth
lydiajohn@annauniv.edu

V. Rhymend Uthariaraj
rhymend@annauniv.edu

[1] Ramanujan Computing Center, Anna University, Chennai, India

[2] Department of Information Technology, Anna University, Chennai, India