BUDA: Budget and Deadline Aware Scheduling Algorithm for Task Graphs in Heterogeneous Systems

Hamza Djigal, Linfeng Liu, Jian Luo, Jia Xu
Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu 210023 China
djigalshamza@gmail.com, {liulf, luoj, xujia}@njupt.edu.cn,
Corresponding author: Jia Xu

***Abstract*—Task graphs are widely used to represent data-intensive applications. To efficiently execute these applications on heterogeneous systems, each task must be properly scheduled on the processors of the system. The NP-completeness of the task scheduling problem has motivated researchers to propose various heuristic methods. Recently, Quality of Service (QoS) aware scheduling is becoming an active research area in heterogeneous systems because the end-user has different QoS requirements. Generally, time and cost are the most relevant user concerns. However, it is challenging to find a feasible scheduling plan which minimizes the total execution time of the user's application (makespan) while satisfying both budget and deadline constraints. In this paper, we present a novel heuristic algorithm called Budget-Deadline-Aware-Scheduling (BUDA) that addresses task graphs scheduling under budget and deadline constraints in heterogeneous systems. The novelty of the BUDA algorithm is based on a Heterogeneous Time-Cost Matrix (HTCM) that is used to prioritize tasks and for processor selection. In addition, we introduce a new Heterogeneous Time-Cost Trade-off factor (HTCT) that tries to adjust the time and cost for the current task among all processors. The experiments based on randomly generated graphs and real-world applications graphs show that the BUDA algorithm outperforms the state-of-the-art algorithms in terms of makespan, time efficiency, and success rate.**

***Index Terms*—Task scheduling, quality of service, task graphs, heterogeneous system**

I. Introduction

A heterogeneous computing system (HCS) can be defined as a set of computing resources (e.g., central processing units (CPUs) or graphics processing units (GPUs)) interconnected with a high-speed network for executing data-intensive applications [1]. In HCS, the computing resources have different performances with different prices and Quality of Service (QoS) levels [2]. To efficiently execute data-intensive applications on a heterogeneous system, each task must be properly scheduled on the processors of the system. Task scheduling consists of assigning the tasks of the application to a set of processors and ordering their execution so that the dependencies between them are maintained while optimizing one or more QoS parameters. The usual QoS parameters are budget and deadline. In general, the task scheduling problem is an NP-complete problem even in the case of scheduling a set of jobs that requires one or two time-unit onto two processors [3]. Hence, QoS-aware scheduling has become more challenging, and many heuristic methods have been proposed [4], [5], [6], [7], [8].

Recently, budget-deadline-aware scheduling is becoming an active research area in heterogeneous systems because time and cost are the most relevant user concerns. Many algorithms have been proposed, but most of them do not consider the time and monetary cost together during the scheduling decision, which mainly comprises the task prioritization phase and computing resource selection phase. Therefore, these algorithms either minimize the time while having a high cost or reduce the cost while having a high execution time. Moreover, these algorithms fail to satisfy the defined budget and deadline simultaneously.

This paper presents a novel heuristic scheduling algorithm called Budget-Deadline-Aware (BUDA) for allocating budget and deadline constrained task graphs to fully connected heterogeneous processors. The proposed algorithm aims to find a feasible schedule map that minimizes the total execution time (TET) also called makespan while satisfying the budget and deadline constraints simultaneously. The novelty of BUDA is the Heterogeneous Time-Cost Matrix (HTCM) on which the task prioritization and processor selection are based.

The rest of the paper is organized as follows. Section II discusses the related work. Section III describes the heterogeneous system, application model, and the research problem. Section IV presents the proposed BUDA algorithm. Section V presents the experimental results. Finally, Section VI concludes the paper.

II. Related Work

The workflow or task graph scheduling problem has been widely studied and various heuristic algorithms are proposed. These algorithms can been classified into variety of categories according to the scheduling objectives. The minimization of the total execution time (makespan) is the major objective in most of the scheduling algorithms [9]. In [1], a wide

comprehensible heuristic list scheduling algorithms is presented with the objective to minimize the makespan. The energy consumption minimization is another important goal addressed by several scheduling algorithms, such as [10] and [11]. Some algorithms consider one or more objectives, e.g., makespan and energy consumption minimization [12], energy consumption and latency [13], and makespan and monetary cost minimization [14]. Other scheduling objectives, such as security-aware [15], [16], and reliability-aware [17], [18] are also proposed. Moreover, the authors of [19] present a classification of scheduling objectives.

However, recently, more works are focusing on the development of QoS-aware scheduling algorithms which is more complex since it addresses the optimization problem (i.e., the minimization or maximization of one or more objective functions) while satisfying one or more QoS constraints. Generally, time and cost are the most relevant user concerns since the user wants to minimize the execution time while meeting both the deadline and budget of its application on the target computing resources. Hence, recent works propose budget-aware and deadline-aware scheduling algorithms. A deadline-aware scheduling algorithm aims to minimize the total execution time while satisfying the defined deadline constraint [4], [5]. Concerning a budget-aware scheduling algorithm, it aims to minimize the total execution time while satisfying the budget constraint [20] [21]. Finally, the objective of a budget-deadline-aware scheduling algorithm is to minimize the makespan while satisfying the deadline and budget simultaneously. For instance, a budget-deadline-aware scheduling algorithm called BHEFT is presented in [22]. The BHEFT algorithm is an extension on HEFT algorithm that used $rank_u$ as defined in [23] to prioritize tasks. In BHEFT, the processor selection is based on three variables, namely spare application budget, task actual budget, and adjustment factor. In [24], the authors propose a budget constrained scheduling algorithm called HBCS with the objective to minimize the makespan while satisfying the defined budget. Similar to BHEFT, the HBCS algorithm also uses $rank_u$ to calculate the task priority. To select a processor for a task, the HBCS algorithm computes a worthiness value for each task on each processor. The worthiness value guarantees that the schedule result does not exceed the given budget. The drawback of HBCS algorithm is that it yields to longer makespan since it assigns task with low priority and small budget to the processor with the minimum execution cost. For this reason, the authors of [21] present a new budget constrained scheduling algorithm called MSLBL. MSLBL also used $rank_u$ to prioritize task. The main idea of MSLBL is to transfer the given budget constraint to the task of the application. In MSLBL, a task is assigned to the processor that achieves the smallest EFT value while satisfying the current task budget. A more recent budget constrained application scheduling algorithm called FBCWS is proposed in [25] with the aim to minimize the makespan while achieving a fair task assignment strategy that satisfies the budget constraint. The FBCWS algorithm first categorizes tasks of the application into two types: CPU intensive tasks, which require more processing power and time, and less intensive tasks. Then, tasks are ranking in a decreasing order of their B-level value. In the processor selection phase, the less intensive tasks are assigned to slow processors while the CPU intensive tasks are allocated to the faster processors.

Limitation of existing works. Most scheduling algorithms for budget and deadline constrained applications used the upward rank value ($rank_u$, as defined in [23]) to prioritize tasks. However, $rank_u$ uses the average execution time $\overline{ET}(T_i)$ of task T_i, and does not consider the execution cost of the task on the heterogeneous processors. Also, in heterogeneous computing systems, each task has variable execution times on different processors, which leads to variable execution costs. For this reason, the authors of [26] introduce a heterogeneous upward rank ($hrank_u$) value to determine the priority of tasks. However, the $hrank_u$ value also does not consider the execution cost (monetary cost). Therefore, for accuracy, we introduce a novel task prioritization approach, which considers not only the heterogeneity of the processors but also the execution times and execution costs of the tasks on the processors.

III. Problem definition

A. Target System

The heterogeneity model of a target computing system can be categorized into two models: (i) processor-based heterogeneity model (PHM), where a processor executes the application's tasks at the same speed, regardless of their type; (ii) task-based heterogeneity model (THM), where the speed at which a processor executes a task depends on how well the heterogeneous processor satisfies the task requirements and features [27].

In this study, we assume a THM model as in [27], because different tasks may have different processing requirements. The heterogeneous computing system consists of a set P of m heterogeneous processors, $P_1, P_2, ..., P_m$, which are fully interconnected with a high-speed network. The processors are priced in a way that the most powerful processor will have the highest execution cost (i.e, monetary cost). To normalize different price units for the heterogeneous processors, we define the price $Price(T_i, P_j)$ of a processor P_j to execute a task T_i as follows:

$$Price(T_i, P_j) = \alpha_{i,j}(1 + \alpha_{i,j})/2, \tag{1}$$

where $\alpha_{i,j}$ defined by (2), is the ratio of the speed $SP(T_i, P_j)$ at which processor P_j executes task T_i to the speed at which the faster processor executes T_i. Since we assume a THM, the heterogeneous processors will have different price units for different tasks. Hence, the price will be in the range of]0,1], where the fastest processor with the highest power has a price value equal to 1. This approach that we used to normalize different price units for the heterogeneous processors is also adopted in [2].

$$\alpha_{i,j} = \frac{SP(T_i, P_j)}{\max\limits_{P_k \in P}[SP(T_i, P_k)]}. \tag{2}$$

B. Application Model

In this paper, an application is represented by a direct acyclic graph (DAG) $G(T, E)$, where T is the set of n tasks, and E is the set of e edges. Each task $T_i \in T$ is associated with a non-negative weight $W(T_i)$ representing the amount of data to be processed in task T_i. Each edge $e(i, j) \in E$ is also associated with a non-negative weight $C(T_i, T_j)$ representing the communication time between task T_i and task T_j. The edges also represent the task-dependence constraints, i.e., task T_i should complete its execution before task T_j can be started. The time to execute task T_i on processor P_j denoted as $ET(T_i, P_j)$ is calculated by (3). The average execution time $\overline{ET(T_i)}$ of task T_i is calculated by (4).

$$ET(T_i, P_j) = \frac{W(T_i)}{SP(T_i, P_j)} \tag{3}$$

$$\overline{ET(T_i)} = \frac{1}{k} \sum_{j=1}^{k} ET(T_i, P_j) \tag{4}$$

The average communication cost $\overline{C(T_i, T_j)}$ of an edge $e(i, j)$ is calculated as follows:

$$\overline{C(T_i, T_j)} = \overline{L} + \frac{data(T_i, T_j)}{\overline{B}}, \tag{5}$$

where $\overline{L}$ is the average latency time of all processors and $\overline{B}$ is the average transfer rate among the processors. $data(T_i, T_j)$ is the amount of data required to be sent from task T_i to task T_j. It should be noted that when T_i and T_j are scheduled on the same processor, the communication cost is considered to be zero since it is negligible compared with the interprocessor communication cost [28].

C. Definitions

a) Earliest Start Time (EST): The EST of a task T_i on a processor P_j is defined as follows:

$$EST(T_i, P_j) = \max\{avail(P_j), \max_{T_k \in pred(T_i)} \{AFT(T_k) + C(T_k, T_i)\}\}, \tag{6}$$

where $avail(P_j)$ is the earliest time at which processor P_j is ready. The inner max block in (6) is the time at which all data required by T_i has arrived at P_j, and $pred(T_i)$ is the set of immediate predecessor tasks of T_i. $AFT(T_k)$ is the actual finished time of task T_k, i.e., the time when task T_k scheduled on a processor P_j was actually finished. For the entry task T_{entry} , $EST(T_{entry}, P_j) = 0$.

b) Earliest Finished Time (EFT): The EFT of a task T_i on a processor P_j is defined as follows:

$$EFT(T_i, P_j) = EST(T_i, P_j) + ET(T_i, P_j) \tag{7}$$

c) Schedule: We define a Schedule Sch of an application as a tuple $\{T_i, P_j, EST(T_i, P_j), EFT(T_i, P_j), EC(T_i, P_j)\}$. It is interpreted as task T_i is assigned to processor P_j, and P_j is expected to start executing T_i at time $EST(T_i, P_j)$ and completed by time $EFT(T_i, P_j)$. $EC(T_i, P_j)$ is cost (monetary) of executing T_i on P_j.

TABLE I: Acronyms Used in the Paper

Notation	Definition
$ET(T_i, P_j)$	Execution time of task T_i on processor P_j
$EC(T_i, P_j)$	Execution cost of task T_i on processor P_j
$SP(T_i, P_j)$	Speed at which a processor P_j executes task T_i
$Price(T_i, P_j)$	unit price of a processor P_j to executes task T_i
P_{select}	Processor selected to execute the current task
β	Budget factor
δ	Deadline factor
$Makespan(G)$	Total execution time of the application G
$TEC(G)$	Total execution cost of the application G

d) Makespan: The overall completion time of a DAG G, or makespan, also called scheduling length is defined as the actual finished time (AFT) of the last task in G, i.e., the exit task (T_{exit}). If there are multiple exit tasks and no redundant task is added, the makespan is the maximum AFT of all exit tasks. It can be defined as follows:

$$Makespan = \max_{T_{exit} \in S_{exit}} [AFT(T_{exit})], \tag{8}$$

where S_{exit} is the set of the exit tasks.

e) Total Execution Cost (TEC): The execution cost of running a task T_i on processor P_j denoted as $EC(T_i, P_j)$ is calculated by the following equation:

$$EC(T_i, P_j) = ET(T_i, P_j) \times Price(T_i, P_j). \tag{9}$$

The total execution cost TEC of an application represented by a DAG G can be calculated as:

$$TEC(G) = \sum_{i=1}^{n} EC(T_i, P_{select}), \tag{10}$$

where P_{select} is the processor on which task T_i has been assigned, and n is the number of tasks in the application. Table I summarizes the acronyms used in this paper.

D. Problem Formulation

Based on the previous definitions, the problem can be formulated as follows: assign the tasks of a given DAG to a set of processors such that the $Makespan$ is minimized while satisfying both budget (β) and deadline (δ) constraints specified by the user. This is expressed by (11).

$$\begin{aligned} &Minimize : Makespan, \\ &subject \quad to : Makespan \leq \delta, \\ &TEC \leq \beta. \end{aligned} \tag{11}$$

IV. Proposed BUDA Algorithm

In this section, we present a new Budget and Deadline aware scheduling algorithm, called BUDA, that minimizes the makespan while satisfying both budget and deadline constraints. The BUDA algorithm has two main phases: a task prioritizing phase for giving priority to tasks, and a processor selection phase for selecting the suitable processor

to execute the current task. Before introducing the details of our algorithm, we define the novelty concept, namely the Heterogeneous Time-Cost Matrix (HTCM), which determines the two phases of the algorithm.

A. Heterogeneous Time-Cost Matrix

The novelty of our algorithm is based on the Heterogeneous Time-Cost Matrix (HTCM) on which the task prioritizing and processor selection phases are based. HTCM is a $(n \times m)$ matrix, where each element $HTCM(T_i, P_j)$ represents the length of the critical path from task T_i to the exit task, including both execution time and execution cost of T_i on the processor P_j. The elements $HTCM(T_i, P_j)$ of the matrix $HTCM$ are recursively defined by (12) by traversing the DAG from the exit task to the entry task.

$$\begin{aligned} HTCM(T_i, P_j) = \max_{T_k \in succ(T_i)} \big\{ & HTCM(T_i, P_j) \\ & + ET(T_i, P_j) + EC(T_i, P_j) + \overline{C(T_i, T_k)} \big\}, \end{aligned} \tag{12}$$

where $HTCM(T_{exit}, P_j) = ET(T_{exit}, P_j) + EC(T_{exit}, P_j)$, and $succ(T_i)$ is the set of immediate successor tasks of task T_i.

By considering both execution time (ET) and execution cost (EC) in HTCM, we are giving priority to the task with higher ET and EC values, which can lead to a shorter makespan as shown in the illustration example IV-E.

B. Task Prioritizing Phase

To determine task priority, we first compute the average $HTCM$ value of each task denoted by $rank_{htcm}$, and defined as follows:

$$rank_{htcm}(T_i) = \frac{\sum_{j=1}^{m} HTCM(T_i, P_j)}{m}, \tag{13}$$

where m is the number of processors. Then, the task priority list is obtained by sorting the tasks in decreasing order of their $rank_{htcm}$ values.

C. Processor Selection Phase

To select a processor for a task, we compute the Heterogeneous Time-Cost Trade-off (HTCT) value defined by (14), which is the summation of EFT and HTCM. In this way, we are trying to adjust the time and cost for the current task among all processors. The goal is to guarantee that a processor that achieves the Earliest Finish Time for the current task, but with a high monetary cost may not be selected, and this is the purpose of the HTCM matrix. This goal may be achieved since both execution time and execution cost are included in HTCM.

$$HTCT(T_i, P_j) = EFT((T_i, P_j) + HTCM(T_i, P_j) \tag{14}$$

D. Description of BUDA Algorithm

Before the description of the BUDA algorithm, we define the attributes used in the algorithm as follows:

- max_{eft} and max_{htct} are the initial values of EFT and HTCT, respectively.
- $\mathbf{EC_{min}(T_i)}$ denotes the minimum execution cost of task T_i among all available processors of the target system;
- $\mathbf{P_{best}}$ is defined as the processor that yields the smallest EFT value for the actual task.
- $\mathbf{EC_{best}(T_i)}$ is the execution cost of task T_i on P_{best}.
- **Cheapest Cost** ($Cost_{cheapest}$): denotes the cost of the cheapest assignment and represents the lowest possible cost required for executing the given DAG irrespective of finishing time. It is obtained by assigning each task to its cheapest processor. $Cost_{cheapest}$ is defined by (15).
- **Remaining Average Budget** (RAB): RAB is defined as the remaining cheapest cost for unscheduled tasks, excluding the average execution cost of the current task. The initial value of RAB is equal to $Cost_{cheapest}$. It is updated at each step before selecting a processor for the current task using (16).
- **Available Budget** ($AvailB$): $AvailB$ is defined as the actual remaining budget after execution a task. Its initial value is equal to the user budget β, and it is updated after a processor is selected for the current task using (17).

$$Cost_{cheapest} = \sum_{T_i \in T} EC_{min}(T_i) \tag{15}$$

$$RAB = RAB - \overline{EC}, \tag{16}$$

where $\overline{EC}$ is the average cost of the current task.

$$AvailB = AvailB - EC(T_i, P_{select}), \tag{17}$$

The BUDA algorithm is given by Algorithm 1. The main idea of BUDA is to balance the execution time and the execution cost for the current task among all processors. In this way, the priority of a task will depend not only on the heterogeneity of the processors but also on the task's features. This is more accurate for the task-based heterogeneity model as explained in Section III-A. Also, by adjusting the time and cost, a processor with a high monetary cost may not be selected even if it achieves the minimum earliest finish time for the current task.

The core steps of the BUDA algorithm are described as follows:

(1) The BUDA algorithm starts by computing $HTCM$ matrix and $rank_{htcm}$ at line 1.
(2) At line 2, an empty list denoted by $L\text{-}rank_{htcm}$ is created, and the entry task (T_{entry}) is placed on top of the list.
(3) In lines 3-4, the cheapest cost of the given DAG G is computed, and the algorithm initializes the available budget ($AvailB$) and the remaining average budget (RAB).
(4) In lines 5-35, the BUDA algorithm starts to map the tasks to processors. At each step, the task with the highest $rank_{htcm}$ value among the unscheduled tasks is selected as the current task T_i to be scheduled. After computing the earliest finished time and the execution cost of T_i on each processor, the $EC_{best}(T_i)$ is set (Line 15). Next, the actual task budget ATB is computed

Algorithm 1 BUDA Algorithm

Input: DAG G with Budget B and Deadline D; m heterogeneous processors, $SP(T_i, P_k)$ and $Price(T_i, P_k)$;
Output: Output Sch, $Makespan(G)$, and $TEC(G)$;

1: Compute $HTCM$ and $rank_{htcm}$ for each task using (12) and (13), respectively;
2: Create an empty list $L\text{-}rank_{htcm}$ and put t_{entry} as initial task;
3: . Compute $Cost_{cheapest}$ using (15);
4: $AvailB = \beta$; $RAB = Cost_{cheapest}$; TEC = 0;
5: **while** $L\text{-}rank_{htcm}$ is not empty **do**
6: $T_i \leftarrow$ the task with the highest $rank_{htcm}$ value from $L\text{-}rank_{htcm}$;
7: $max_{eft} = +\infty$; $max_{htct} = +\infty$; update the RAB using (16);
8: **for** $(j = 1; j <= m; j++)$ **do**
9: Compute $EFT(T_i, P_j)$ and $EC(T_i, P_j)$;
10: **if** $max_{eft} > EFT(T_i, P_j)$ **then**
11: $max_{eft} = EFT(T_i, P_j)$;
12: $P_{best} = j$; ▷ Processor with the minimum EFT value for task T_i
13: **end if**
14: **end for**
15: $EC_{best}(T_i) = EC(T_i, P_{best})$;
16: ATB = $AvailB - RAB$;
17: **for** $(j = 1; j <= m; j++)$ **do**
18: **if** $EC(T_i, P_j) > EC_{best}(T_i)$ or $EC(T_i, P_j) > ATB$ **then**
19: $HTCT(T_i, P_j) = +\infty$; ▷ P_j cannot be selected
20: **else**
21: $HTCT(T_i, P_j) = EFT((T_i, P_j) + HTCM(T_i, P_j)$ ▷ P_j is a candidate processor
22: **end if**
23: **if** $max_{htct} > HTCT(T_i, P_j)$ **then**
24: $max_{htct} = HTCT(T_i, P_j)$;
25: $P_{select} = j$;
26: **end if**
27: **end for**
28: Assign task T_i to P_{select};
29: Compute $EST(T_i, P_{select}) = EFT(T_i, P_{select}) - ET(T_i, P_{select})$;
30: $Sch = \{T_i, P_j, EST(T_i, P_j), EFT(T_i, P_j),$
31: $EC(T_i, P_j)\}$;
32: $AFT(T_i) = EFT(T_i, P_{select})$;
33: $TEC(G) = TEC(G) + EC(T_i, P_{select})$;
34: Update $AvailB$ as defined in (17);
35: **end while**
36: $Makespan = AFT(T_{exit})$;
37: **return** Sch, $Makespan(G)$, and $TEC(G)$;

(Line 16). In lines 17-27, the algorithm starts to check the candidate processors. Line 18 guarantees that if the execution cost of task T_i on processor P_j is higher than the execution cost on the processor that achieves the smallest EFT or if that cost is higher than the actual task budget, then processor P_j cannot be selected. Hence, the current schedule does not exceed the budget. In lines 23-26, the processor with the minimum $HTCT$ value is selected to execute the current task T_i. After assigning task T_i to the processor P_{select}, the schedule parameters and the available budget for the remaining unscheduled tasks are updated (lines 29-34).

(5) Finally, the algorithm computes and returns the schedule, the makespan, and the total monetary cost.

In terms of time complexity, BUDA requires the computation of $HTCM$ matrix and the cheapest cost of the given DAG that have complexity $O(n \times m)$, where n and m are the number of tasks and number of processors, respectively. In the processor selection phase, the complexity is $O(n \times m)$ for calculating the EFT and EC for the current task among all processors, and $O(m)$ for selecting the suitable processor for the current task. The total time is $O((n \times m) + n(n \times m) + m)$, where the total BUDA algorithm complexity is of the order $O(n^2 \times m)$. That is, the time complexity of BUDA is of the same order as the BHEFT [22] and HBCS [24] algorithms.

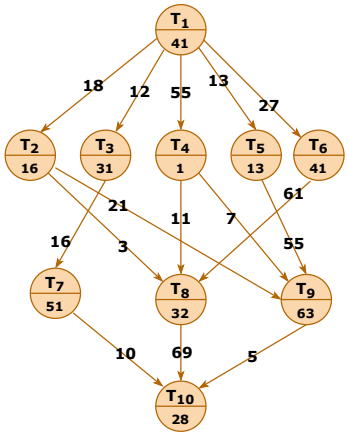


Fig. 1: Sample DAG and execution times of 10 tasks on 3 processors.

E. An Illustrative Example

We use the sample DAG of Fig. 1 to illustrate the performance of the BUDA algorithm. We assume a budget of 230 and a deadline of 240 for the sample DAG. Table II shows the speed of the processors and the DAG attributes. The values of the tasks weight and edges weight are given in Fig. 1. The value of $Price(T_i, P_k)$, $W(T_i, P_k)$, and $EC(T_i, P_k)$ are calculated using (1), (3), and (9), respectively. Table III shows the elements of the HTCM matrix. The priority of each task is calculated using (13). For instance, $rank_{htcm}(T_1) = (484.04 + 750.07 + 943.8)/3 = 726$. After computing the priority of each task, we obtain the task list $(T_1, T_5, T_2, T_4, T_6, T_9, T_3, T_7, T_8, T_{10})$ by ordering them in decreasing value of $rank_{htcm}$. We observe that our algorithm schedules T_7 before T_8, while the BHEFT (Table V) and

TABLE II: Processors Speed (SP), Execution Time (ET), Price, and Execution Cost (EC) for the DAG of Fig. 1

		$SP(T_i, P_k)$			$ET(T_i, P_k)$			$Price(T_i, P_k)$			$EC(T_i, P_k)$		
T_i	$W(T_i)$	P_1	P_2	P_3	P_1	P_2	P_3	P_1	P_2	P_3	P_1	P_2	P_3
1	41	1.07	0.8	0.72	38.32	51.25	56.95	1.0	0.65	0.56	38.32	33.31	31.89
2	16	0.08	0.36	0.66	200	44.44	24.24	0.07	0.42	1.0	14.0	18.66	24.24
3	31	0.27	1.9	1.75	114.82	16.32	17.72	0.08	1.0	0.88	9.19	16.32	15.59
4	1	1.34	0.12	0.35	0.75	8.33	2.86	1.0	0.05	0.16	0.75	0.42	0.46
5	13	0.19	0.79	0.02	68.42	16.46	650	0.15	1.0	0.01	10.26	16.46	6.5
6	41	0.72	1.9	1.52	56.95	21.58	26.98	0.26	1.0	0.72	14.81	21.58	19.43
7	51	1.21	0.33	0.64	42.15	154.55	79.69	1.0	0.17	0.4	42.15	26.27	31.88
8	32	1.22	0.55	1.22	26.23	58.18	26.23	1.0	0.33	1.0	26.23	19.2	26.23
9	63	1.07	0.13	1.67	58.88	484.62	37.73	0.53	0.04	1.0	31.21	19.38	37.73
10	28	1.05	0.87	1.12	26.67	32.19	25	0.91	0.69	1.0	24.27	22.21	25.0

TABLE III: Heterogeneous Time-Cost Matrix (HTCM) for the DAG of Fig. 1

Task	P_1	P_2	P_3
T_1	484.04	750.07	943.8
T_2	389.34	647.51	222.94
T_3	285.25	293.86	220.88
T_4	184.9	579.15	185.78
T_5	279.71	651.33	841.96
T_6	305.16	304.94	278.87
T_7	145.24	245.22	171.57
T_8	172.4	200.78	171.46
T_9	146.03	563.4	130.46
T_{10}	50.92	54.40	50.0

HBCS (Table VI) algorithms that used $rank_u$ schedule T_8 before T_7. This is due to the fact that our algorithm considers both execution time (ET) and execution cost (EC) while the BHEFT and HBCS algorithms do not consider the EC during the task prioritization phase. Therefore, our algorithm tries to give priority to the task with the highest ET and EC values, which are included in the HTCM matrix. Since task T_7 has higher ET and EC values than task T_8 (Table II), it is scheduled first by our algorithm.

Table IV shows the schedule produced by the BUDA algorithm. To see how the HTCT values computed for each task guide the processor selection, let us consider the scheduling of task T_5. It can be seen that the processor P_2 (best processor) gives the smallest EFT value for T_5. However, T_5 is scheduled on P_1 instead of P_2. The reason is that the execution cost of task T_5 on P_2, i.e, $EC(T_5, P_2) = 16.46$ (Table II) is higher than the execution cost of T_5 on P_1, which is equal to 10.26. This information is included in the HTCM matrix on which the processor selection is based. In the same manner task T_3 is scheduled on P_3 instead of P_2. We can also observe that the HTCT values guide the BUDA algorithm to determine the candidate processors, i.e., the processors that can be selected for the current task. For instance, P_1 is not a candidate processor for task T_4 because $HTCT(T_4, P_1) = +\infty$ (Table IV). The makespan and the total execution cost (TEC) obtained by the BUDA algorithm are 235.78 and 214.95, respectively, which satisfy both budget and deadline constraints. Also, the makespan and TEC of the BUDA algorithm are less than those of BHEFT (Table V) and HBCS (Table VI) algorithms.

This is a simple numerical example used to illustrate our algorithm. The next section shows the performance of BUDA over the related algorithms.

V. EXPERIMENTAL RESULTS AND DISCUSSION

This section compares the performance of the BUDA algorithm with BHEFT [22], HBCS [24], and FBCWS [25] algorithms. For this purpose, we consider two sets of graphs as the workload: randomly generated graphs and real-world applications graphs.

A. Comparison Metrics

- **Average Makespan**. The makespan is the usual metric used to evaluate the performance of scheduling algorithms. We use the average makespan as a metric because a large set of DAGs with different properties is used. The algorithm with the smallest average makespan is the best algorithm according to this metric.
- **Cost Ratio (CR)**: To compare the achieved money cost between each algorithm, we use a cost ratio which is calculated by dividing the total execution cost defined in (10) by the given budget β.

$$CR = \frac{TEC(G)}{\beta}. \tag{18}$$

A CR value greater than one denotes a monetary cost larger than the defined budget which counts as a failure to meet the budget. The algorithm with the smallest CR value is the best algorithm in terms of monetary cost according to this metric.

TABLE IV: Schedule Produced by the BUDA Algorithm

Task	ATB	EFT			Best	HTCT			Processor	EST	Execution
T_i		P_1	P_2	P_3	Processor	P_1	P_2	P_3	selected		Cost
T_1	101	**38.32**	51.25	56.95	P_1	**522.36**	801.325	1000.75	P_1	0.0	38.32
T_5	74	106.74	**67.78**	701.32	P_2	**386.45**	719.11	1543.28	P_1	38.32	10.26
T_2	83	306.74	100.76	**80.56**	P_3	696.14	748.27	**303.5**	P_3	56.32	24.24
T_4	60	107.49	101.65	**96.18**	P_3	$+\infty$	680.8	**281.96**	P_3	93.32	0.46
T_6	79	163.69	**86.9**	123.16	P_2	468.85	**391.84**	402.03	P_2	65.32	21.58
T_9	86	**165.62**	646.36	199.47	P_1	**311.65**	1209.77	$+\infty$	P_1	106.74	31.21
T_3	69	280.44	**103.22**	113.9	P_2	565.69	397.08	**334.78**	P_3	96.18	15.59
T_7	86	207.77	284.45	**193.59**	P_3	$+\infty$	529.67	**365.16**	P_3	113.9	31.88
T_8	78	191.8	**165.36**	219.82	P_2	$+\infty$	**366.14**	$+\infty$	P_2	107.18	19.2
T_{10}	83	261.03	**235.78**	259.36	P_2	$+\infty$	**290.18**	$+\infty$	P_2	203.59	22.21

The $+\infty$ value in the table means that the Processor cannot be selected.
Makespan(G) = 235.78 and TEC(G) = 214.95

TABLE V: Schedule Produced by BHEFT Algorithm

T_i	$rank_u(T_i)$	EST	EFT	P_{select}	EC
T_1	588.5	0	38.32	P_1	38.32
T_5	526.66	38.32	106.74	P_1	10.26
T_2	337.26	56.32	100.76	P_2	18.66
T_4	237.68	93.32	96.18	P_3	0.46
T_6	230	96.18	123.16	P_3	19.43
T_9	226.7	121.76	180.64	P_1	31.21
T_3	195.7	123.16	140.88	P_3	15.59
T_8	133.83	140.88	167.11	P_3	26.23
T_7	130.08	167.11	246.8	P_3	31.88
T_{10}	27.95	246.8	271.8	P_3	25.0

Makespan(G) = 271.8 and TEC(G) = 217.04

TABLE VI: Schedule Produced by HBCS Algorithm

T_i	$rank_u(T_i)$	EST	EFT	P_{select}	EC
T_1	588.5	0	38.32	P_1	38.32
T_5	526.66	38.32	106.74	P_1	10.26
T_2	337.26	56.32	100.76	P_2	18.66
T_4	237.68	93.32	96.18	P_3	0.46
T_6	230	96.18	123.16	P_3	19.43
T_9	226.7	121.76	180.64	P_1	31.21
T_3	195.7	100.76	117.08	P_2	16.32
T_8	133.83	123.16	149.39	P_3	26.23
T_7	130.08	149.39	229.08	P_3	31.88
T_{10}	27.95	229.08	254.08	P_3	25.0

Makespan(G) = 254.08 and TEC(G) = 217.77

- **Time Ratio (TR)**: To compare the achieved time between each algorithm, we use a time ratio which is calculated by dividing the makespan by the defined deadline δ.

$$TR = \frac{Makespan(G)}{\delta}. \tag{19}$$

With the same reasoning with the cost ratio, a TR value greater than one indicates an execution time larger than the defined deadline which counts as a failure to meet the deadline. Based on this metric, the algorithm with the smallest TR value is the best algorithm in terms of time efficiency.

- **Success Rate (SR)**: SR is defined as the ratio between the number of simulation runs that successfully met the scheduling constraints, i.e., deadline and budget denoted as $Success_{run}$, and the total number of simulation runs denoted by $Total_{run}$. To obtain the SR in percentage we multiply its value by 100. Hence, SR is defined as follows:

$$SR = \frac{Success_{run}}{Total_{run}} \times 100. \tag{20}$$

B. Random Graph Generator

We used a synthetic DAG generation program available at [29] to generate weighted DAGs. We consider the following parameters to generate the DAGs:

- Number of task $n = \{20, 40, 60, 80, 100\}$.
- Communication to Computation Ratio (CCR). It is defined as the ratio of the sum of the edge weights to the sum of the task weights in a DAG.
- $CCR = \{0.1, 5, 10\}$.
- Number of processors $m = \{32, 64\}$.
- The parallelism factor is randomly taken from $\{1, 2\}$.
- The speeds $SP(T_i, P_j)$ were randomly taken from $]0, 2[$ as in [27].

Budget factor β and deadline factor δ. We have considered different combinations of β and δ as follows: $(\beta, \delta) = \{(4,4), (5,5), (6,6), (7,7), (8,8), (12,12), (16,16), (20,20), (4,8), (8,12), (12,16), (16,20), (8,4), (12,8), (16,12), (20,16)\}$. These combinations result in a sufficient budget and very relaxed deadline as shown in [30]. These parameters give a total of 4000 different random DAGs since 25 DAGs were generated for each DAG type.

C. Real-World Application Graphs

In addition to the randomly generated DAGs, we also evaluated the performance of the algorithms with two well-known scientific applications: molecular dynamics code [1] (Fig. 2), and Epigenomics with 100 tasks [31]. Since the structure of these applications is well-known, we simply consider the same DAG parameters and (β, δ) combinations given in Section V-B. For these applications, we have considered 10 different

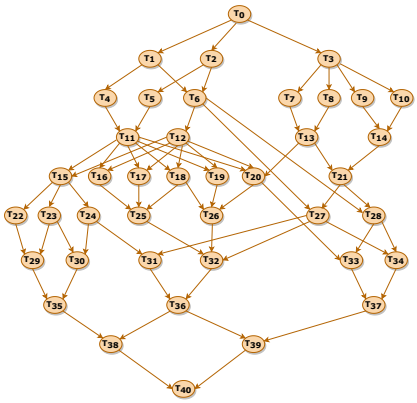
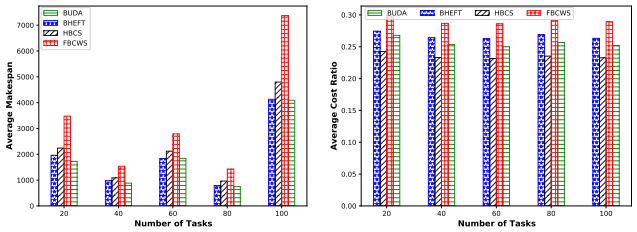
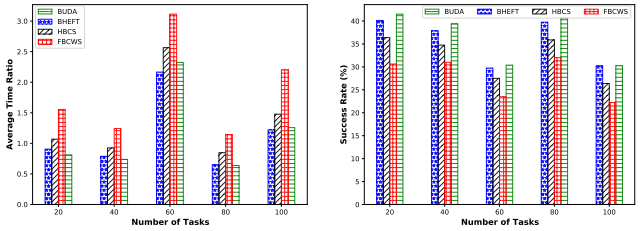


Fig. 2: Task graph of molecular dynamic code



(a) Makespan using 32 processors (b) Cost ratio using 32 processors



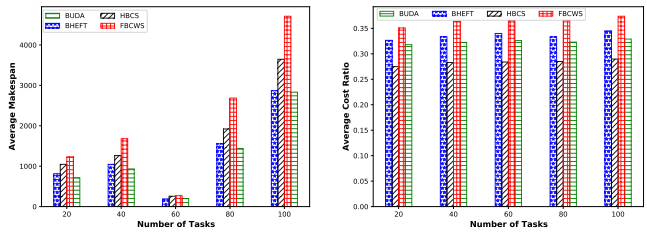
(c) Time ratio using 32 processors (d) Success rate using 32 processors

Fig. 3: (a) Average makespan, (b) average cost ratio, (c) average time ratio, and (d) success rate as function of DAGs size.

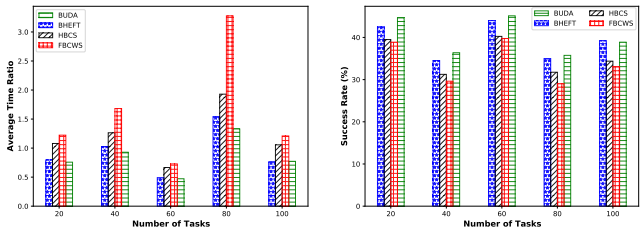
graphs that lead to a total of 320 different graphs for each real-world application type.

D. Performance Results

1) Results of Randomly Generated Graphs with 32 processors for varying number of tasks : Fig. 3 shows the average makespan for all algorithms with respect to the number of tasks. The BUDA algorithm obtained the best results. For instance, when the number of tasks n is equal to 20, the BUDA algorithm outperforms the HBEFT, HBCS, and FBCWS algorithms by 12.01%, 22.92%, and 50.35%, respectively. In terms of the average cost ratio (Fig. 3b), the HBCS algorithm obtained the best. The second best algorithm is BUDA. For instance, when $n = 100$, HBCS outperformed BUDA by 7.62% and BHEFT by 11.59%. This performance of HBCS over the other algorithms is due to the fact that HBCS always assigns a task to a processor that achieves minimum cost for



(a) Makespan using 64 Processors (b) Cost ratio using 64 processors

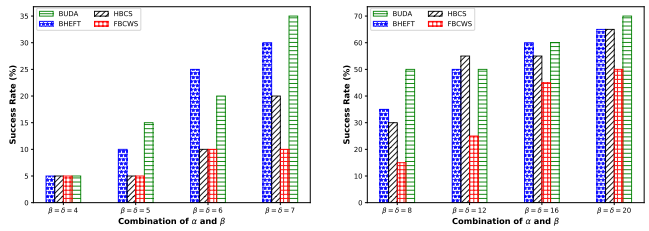


(c) Time ratio using 64 Processors (d) Success rate using 64 Processors

Fig. 4: (a) Average makespan, (b) average cost ratio, (c) average time ratio, and (d) success rate as function of DAGs size.

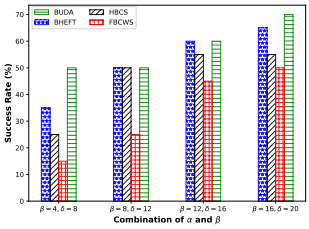
the current task within the available budget. Consequently, HBCS can achieve a shorter cost ratio, but with a higher makespan as shown in Fig. 3a. Also, all algorithms meet the budget constraint for all DAG sizes since their cost ratio values are less than one. For the time ratio (Fig. 3c), the BUDA algorithm achieved a shorter average time ratio than all other algorithms for DAG sizes of 20, 40, and 80. For instance, BUDA surpassed BHEFT by 10.56% and 6.4% for DAGs with 20 and 40 tasks, respectively. For DAGs with 100 tasks, the BHEFT only obtains an average improvement of 3.01% over the BUDA algorithm. Also, the BUDA and BHEFT algorithms meet the deadline constraint for DAG sizes of 20, 40, and 80, while the HBCS algorithm meets the deadline constraint only for DAG sizes of 40 and 80. Moreover, the ranking of the algorithms with respect to success rate (Fig. 3d) is {BUDA, BHEFT, HBCS, FBCWS}. The BUDA algorithm achieved the best results because it can adjust the time and cost during the processor selection phase as shown in Section IV-E (Table IV). The FBCWS algorithm yielded the worst results because it schedules the majority of the less intensive tasks (denoted as LTCTL in [25]) to the slow processors, while the algorithm always scheduled the data-intensive tasks (denoted as MTCTL in [25]) to the faster processors. Consequently, FBCWS produced longer makespan and monetary costs than the other algorithms.

2) Results of Randomly Generated Graphs with 64 processors for varying number of tasks: In terms of makespan, BUDA achieves the best results when the number of processors is equal to 64. For instance, when the number of tasks n is equal to 80, BUDA outperforms the BHEFT by 7.31%, HBCS by 25.02%, and FBCWS by 46.38%. We observe that when the DAGs size is equal to 60, all algorithms achieve a small

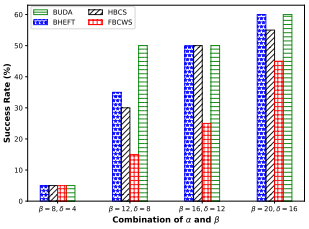


(a) Smaller values of budget (β) and deadline (δ), $\beta = \delta$

(b) Higher values of budget (β) and deadline (δ), $\beta = \delta$

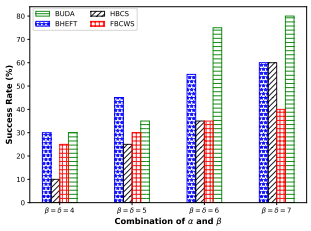


(c) $\beta < \delta$

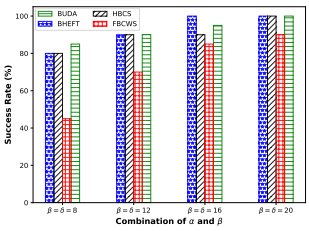


(d) $\beta > \delta$

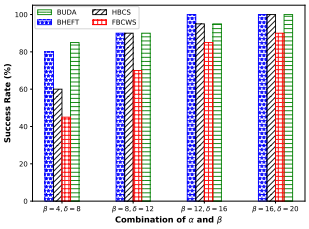
Fig. 5: Success rate of the molecular dynamics code application for varying budget and deadline constraints.



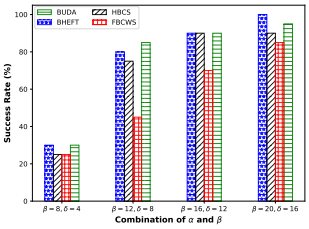
(a) Smaller values of budget (β) and deadline (δ), $\beta = \delta$



(b) Higher values of budget (β) and deadline (δ), $\beta = \delta$



(c) $\beta < \delta$



(d) $\beta > \delta$

Fig. 6: Success rate of the Epigenomic workflow for varying budget and deadline constraints.

average makespan because the CCR values generated for these DAGs are small compared to other random generated DAGs. In terms of average cost ratio (Fig.4b), the HBCS algorithm again gives the best results for the same reasons explained in Section V-D1. For time efficiency, the BUDA algorithm achieved a shorter average time ratio than all other algorithms for DAG sizes of 20, 40, 60, and 80. For instance, BUDA surpassed the BHEFT algorithm by 9.08% and 13.83% for DAGs with 40 and 80 tasks, respectively. For DAGs with 100 tasks, the BUDA and BHEFT obtain similar results. Also, the BUDA algorithm meets the deadline constraint for DAG sizes of 20, 40, 60, and 100, while the BHEFT algorithm meets the deadline constraint for DAG sizes of 20, 60, and 100 only. Moreover, the ranking of the algorithm in terms of success rate (Fig. 4d) is {BUDA, BHEFT, FBCWS, HBCS}.

3) Success rate of the Molecular dynamics code application for varying budget and deadline constraints: Fig. 5a shows the success rate for smaller values of budget (β) and deadline (δ) constraints. On average the BUDA algorithm obtained the highest success rate. For instance, when $\beta = \delta = 5$, the success rate of BUDA, BHEFT, and HBCS algorithms are 15%, 10%, and 5%, respectively. For higher values of (β) and (δ) (Fig. 5b), BUDA again achieved the best results. In particular, when $\beta = \delta = 8$ and $\beta = \delta = 20$, the ranking of the algorithms with respect to the success rate is {BUDA, BHEFT, HBCS, FBCWS} and {BUDA, BHEFT=HBCS, FBCWS}, respectively. Finally, when $\beta < \delta$ (Fig. 5c) and $\beta > \delta$ (Fig. 5d), the BUDA algorithm again produced the best results on average. We observe that the success rate of all algorithms increased when both budget and deadline values increased. This is because the number of tasks in the graph of the molecular dynamics code is fixed.

4) Success rate of the Epigenomic workflow with for varying budget and deadline constraints: Fig. 6 shows the results achieved for Epigenomic in terms of success rate. In most cases, the BUDA algorithm obtained the best performance in terms of meeting both budget and deadline, i.e, the success rate. For instance, when $\beta = \delta \in \{4, 5, 6, 7\}$ (Fig. 6a), the average success rate values of BUDA, BHEFT, HBCS, and FBCWS algorithms are $(30.0 + 35.0 + 75.0 + 80.0)/4 = 55$, $(30.0 + 45.0 + 55.0 + 60.0)/4 = 47.5$, $(10.0 + 25.0 + 35.0 + 60.0)/4 = 32.5$, and $(25.0 + 30. + 35.0 + 40.0)/4 = 32.5$, respectively. Hence, the performance ranking of the algorithms with respect to the success rate is {BUDA, BHEFT, HBCS=FBCWS}. For $\beta = \delta \in \{8, 12, 16, 20\}$, BUDA and BHEFT show similar performance on average and outperform the other algorithms. Finally, when $\beta < \delta$ (6c) and $\beta > \delta$ (6d) BUDA and BHEFT again obtained the best results with similar performance on average.

5) Total Success Rate (TSR): Table VII shows the total success rate of each algorithm considering the combinations of budget factor β and deadline factor δ defined in Section V-B. The best algorithm is BUDA for all application types. For instance, BUDA obtained the highest performance (78.75%) for the Epigenomic application while FBCWS yielded the worst performance with a total success rate value of 58.44%. In general, our BUDA algorithm is more consistent and it is much more likely to give acceptable schedules under budget and deadline constraints.

VI. Conclusion

This paper presents a novel algorithm called BUDA, for scheduling budget and deadline constrained applications on heterogeneous systems. The experiments performed for a large

TABLE VII: Total Success Rate of the Algorithms

Application	BUDA	BHEFT	HBCS	FBCWS
Random DAG	76.58%	74.6%	67.6%	61.93%
Epigenomic workflow	78.75%	76.87%	69.69%	58.44%
Molecular Dynamics Code	43.75%	40.0%	35.63%	24.38%

set of randomly generated graphs proved that the BUDA algorithm outperformed the BHEFT, HBCS, and FBCWS algorithms in terms of average makespan, time ratio, and success rate. The BUDA algorithm also surpassed the benchmark algorithms for the molecular dynamics code and the Epigenomic workflows. However, for the Epigenomic workflow, BUDA and BHEFT obtained similar results when the budget factor and the deadline factor are different. This is due to the feature of the Epigenomic graphs, which is characterized by having more tasks belong to the critical path.

One key future research direction is to define a mechanism to control the execution time and execution cost so that in any step, the user can define the QoS that should be satisfied. We also intend to extend the algorithm by considering the energy consumption during both the task prioritization phase and the processor selection phase. This consideration will allow the user to minimize energy consumption while satisfying the budget and deadline constraints.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grants 61872193 and 61872191.

REFERENCES

[1] H. Djigal, F. Jun, J. Lu, and J. Ge, "Ippts: An efficient algorithm for scientific workflow scheduling in heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, pp. 1–1, 2020.

[2] H. Arabnejad, J. G. Barbosa, and R. Prodan, "Low-time complexity budget–deadline constrained workflow scheduling on heterogeneous resources," *Future Gener Comp Sy*, vol. 55, pp. 29–40, 2016.

[3] J. Ullman, "Np-complete scheduling problems," *J Comput Syst Sci*, vol. 10, no. 3, pp. 384–393, 1975.

[4] R. A. Haidri, C. P. Katti, and P. C. Saxena, "Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 6, pp. 666–683, 2020.

[5] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1270–1286, 2019.

[6] F. Wu, Q. Wu, Y. Tan, R. Li, and W. Wang, "Pcp-b2: Partial critical path budget balanced scheduling algorithms for scientific workflow applications," *Future Gener Comp Sy*, vol. 60, pp. 22–34, 2016.

[7] C. Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li, "End-to-end delay minimization for scientific workflows in clouds under budget constraint," *IEEE Trans. on Cloud Comput.*, vol. 3, no. 2, pp. 169–181, 2015.

[8] H. R. Faragardi, M. R. Saleh Sedghpour, S. Fazliahmadi, T. Fahringer, and N. Rasouli, "Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1239–1254, 2020.

[9] H. Djigal, J. Feng, and J. Lu, "Task scheduling for heterogeneous computing using a predict cost matrix," in *Proc. of the 48th International Conference on Parallel Processing: Workshops*, ser. ICPP 2019. New York, NY, USA: ACM, 2019, pp. 25:1–25:10.

[10] M. Safari and R. Khorsand, "Energy-aware scheduling algorithm for time-constrained workflow tasks in dvfs-enabled cloud environment," *Simul Model Pract Th*, vol. 87, pp. 311–326, 2018.

[11] U. U. Tariq, H. Ali, L. Liu, J. Hardy, M. Kazim, and W. Ahmed, "Energy-aware scheduling of streaming applications on edge-devices in iot-based healthcare," *IEEE trans. green commun. netw.*, vol. 5, no. 2, pp. 803–815, 2021.

[12] M. Mezmaz, N. Melab, Y. Kessaci, Y. C. Lee, E.-G. Talbi, A. Y. Zomaya, and D. Tuyttens, "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems," *J. Parallel Distrib. Comput.*, vol. 71, no. 11, pp. 1497–1508, 2011.

[13] X. Huang, S. Leng, S. Maharjan, and Y. Zhang, "Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 9, pp. 9282–9293, 2021.

[14] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1344–1357, May 2016.

[15] H. Djigal, J. Feng, and J. Lu, "Performance evaluation of security-aware list scheduling algorithms in iaas cloud," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 330–339.

[16] L. Zeng, B. Veeravalli, and X. Li, "Saba: A security-aware and budget-aware workflow scheduling strategy in clouds," *J. Parallel Distrib. Comput.*, vol. 75, pp. 141 – 151, 2015.

[17] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Reliability-aware scheduling and routing for messages in time-sensitive networking," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5, pp. 1–24, 2021.

[18] S. Saha, X. Zhai, S. Ehsan, S. Majeed, and K. McDonald-Maier, "Rasa: Reliability-aware scheduling approach for fpga-based resilient embedded systems in extreme environments," *IEEE Trans. Syst., Man, Cybern., Syst.*, 2021.

[19] M. Rodriguez and R. Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments," *Concurr. Comput. Pract. E.*, vol. 29, no. 8, 2017.

[20] Y. Caniou, E. Caron, A. K. W. Chang, and Y. Robert, "Budget-aware scheduling algorithms for scientific workflows with stochastic task weights on heterogeneous iaas cloud platforms," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 15–26.

[21] W. Chen, G. Xie, R. Li, Y. Bai, C. Fan, and K. Li, "Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems," *Future Gener Comp Sy*, vol. 74, pp. 1–11, 2017.

[22] W. Zheng and R. Sakellariou, "Budget-deadline constrained workflow planning for admission control," *Journal of grid computing*, vol. 11, no. 4, pp. 633–651, 2013.

[23] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002.

[24] H. Arabnejad and J. G. Barbosa, "A budget constrained scheduling algorithm for workflow applications," *Journal of grid computing*, vol. 12, no. 4, pp. 665–679, 2014.

[25] N. Rizvi and D. Ramesh, "Fair budget constrained workflow scheduling approach for heterogeneous clouds," *Cluster Computing*, vol. 23, no. 4, pp. 3185–3201, 2020.

[26] G. Xie, R. Li, and K. Li, "Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems," *J. Parallel Distrib. Comput.*, vol. 83, pp. 1–12, 2015.

[27] Y. Xu, K. Li, L. He, L. Zhang, and K. Li, "A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3208–3222, 2015.

[28] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, 2014.

[29] Wzwtime, "Randomgraphgenerator," 2017. [Online]. Available: https://github.com/wzwtime/RandomGraphGenerator_new

[30] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget and deadline aware e-science workflow scheduling in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 29–44, 2019.

[31] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," 2008.