Multi-Hop Offloading of Multiple DAG Tasks in Collaborative Edge Computing

Yuvraj Sahni, Jiannong Cao, *Fellow, IEEE,* Lei Yang, and Yusheng Ji, *Senior Member, IEEE*

Abstract—Collaborative edge computing (CEC) is a recently popular paradigm enabling sharing of data and computation resources among different edge devices. Task offloading is an important problem to address in CEC as we need to decide when and where each task is executed. However, it is challenging to solve task offloading in CEC as tasks can be offloaded to a multi-hop neighbouring device leading to bandwidth contention among network flows. Most existing works do not jointly consider network flow scheduling which can lead to network congestion and inefficient performance in terms of completion time. Another challenge is to formulate and solve the problem considering the dependencies among dependent tasks and conflicting network flows. Few recent works have considered multi-hop computation offloading; however, these works focus on independent tasks and do not jointly consider the dependencies with network flows. In this work, we mathematically formulate the problem of jointly offloading multiple tasks consisting of dependent subtasks and network flow scheduling in CEC to minimize the average completion time of tasks. We have proposed a joint dependent task offloading and flow scheduling heuristic (JDOFH) that considers both dependencies in task DAG and start time of network flows. Performance comparison done using simulation for both real application task graph and simulated task graphs shows that JDOFH leads to up to 85% improvement in average completion time compared to benchmark solutions which do not make a joint decision.

Index Terms—Offloading, DAG tasks, Network flow scheduling, Collaborative Edge Computing, Internet of Things.

1 Introduction

Collaborative edge computing (CEC) has become a popular computing paradigm recently in which multiple stakeholders (IoT devices, edge devices, cloud, or end-users) collaborate with each other by sharing data and computation resources to satisfy individual and/or global goals. There are many challenging issues to be addressed in CEC, including collaboration space formation, social trust-based incentive policies, cooperation policies, inter-domain cooperation, smart collaborative networking, and mobility management [1]. Many existing works such as [2], [3], [4], [5], [6], etc. have studied different problems related to collaborative edge computing. The work in [4] proposed a framework called CVEC to support large-scale vehicular services by using both horizontal and vertical collaboration. Another work [6] proposed CEC among small-cell base stations (SBSs) by forming SBS coalitions to share computation resources. In our previous work, we proposed Edge Mesh [7] that pushes the computation within the network by sharing data and computation tasks among mesh network of edge devices instead of sending all the data to a centralized server.

One fundamental problem in CEC is to offload and schedule computation tasks among edge devices. However, unlike many existing works that offload computation to a single-hop neighbour, tasks in CEC can be offloaded to a device at a multi-hop distance depending on resource availability. Multi-hop offloading has an advantage over single-hop by enabling the use of underutilized resources in a mesh network of devices. Furthermore, application scenarios such as unmanned aerial vehicle (UAV) robot swarms, autonomous vehicles, etc. have few ground stations or road-side units that can be connected through a multi-hop network with the other devices. Multi-hop offloading is essential in such scenarios as we can utilize computation-intensive edge devices which can be at a multi-hop distance from many resource-constraint devices. There are some recent works in literature such as [8], [9], [10], [11], etc. that have studied multi-hop computation offloading problem. These works, however, focus on independent tasks and usually do not jointly consider network flow scheduling. Network flow scheduling includes making a decision on the start time of the flows. Task offloading without jointly considering network flow scheduling leads to network congestion and inefficient performance as network links have limited bandwidth capacity.

This paper studies the problem of multi-hop dependent task offloading in CEC with the objective of minimizing the average completive time of all tasks. The problem considers jointly offloading tasks, where each task consists of multiple dependent subtasks, and scheduling network flows that are generated to transfer data between dependent subtasks. The task offloading problem includes making a decision on both offloading the subtask to a remote device and scheduling start time of each subtask within the task. The task offloading decision is dependent on the decision of start time of

- *This work was supported in part by the RGC General Research Fund under Grant PolyU 152133/18, and in part by the RGC General Research Fund under Grant PolyU 15217919.*
- *Yuvraj Sahni and Jiannong Cao are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. (E-mail: {csysahni,csjcao}@comp.polyu.edu.hk)*
- *Lei Yang is with the School of Software Engineering, South China University of Technology, Guangzhou, China. (E-mail: sely@scut.edu.cn).*
- *Yusheng Ji is with the Information Systems Architecture Research Division, National Institute of Informatics, Tokyo, Japan and also with the Department of Informatics, SOKENDAI (The Graduate University for Advanced Studies), Tokyo, Japan. (E-mail: kei@nii.ac.jp).*

flows made in network flow scheduling problem. One of the main challenging issues with the problem is that communication cost associated with transfer of data between dependent subtasks is not constant and difficult to estimate as there can be multiple simultaneous network flows due to different subtasks within multiple tasks. The decision on the start time of network flows changes the communication cost of transferring data and hence, the start time of different subtasks. The problem also considers different release time of different tasks. This overall dependence among different decision variables makes it difficult to formulate and solve the problem.

Many existing works have studied scheduling and offloading of dependent tasks such as [12], [13], [14], etc. These works, however, make task scheduling decision without jointly considering network flow scheduling. Some works such as [15] consider network resources but do not leverage dependency among tasks to make task offloading decision. Other works such as [16] assume the underlying network scheduler to make task scheduling decisions. Compared to these works, our work considers jointly making a decision on offloading of multiple DAG tasks and scheduling network flows. Our proposed solution leverages the knowledge of both parallelism and dependency among subtasks in a DAG task to make offloading decisions. Our work does not just consider network bandwidth to decide task offloading but also jointly makes decisions on the start time of network flows to avoid network congestion. We have shown in evaluation that joint decision leads significantly better performance, in terms of average completion time, compared to making task offloading and network flow scheduling decision separately.

This problem is useful for applications such as large-scale multi-camera video analytics where video and image data from multiple cameras is used for generating situational awareness. The idea of collaboration among different edge devices for video processing has been discussed in some recent works such as [17], [18], etc. The work in [17] discusses the conceptual idea of collaborative edge computing for video processing. The work in [18] developed a real-time Edge video analytics system named REVAMP^{2}T for multi-camera privacy-aware pedestrian tracking. The use of collaborative edge computing has also been studied for other application domains such as vehicular networks [4], small-cell base stations [6], etc. The work in [19] developed a lightweight virtualization model to support collaboration among edge devices in smart city and other related applications.

The main contribution made in this paper are:

1) We have mathematically formulated the problem of multi-hop offloading of multiple DAG (directed acyclic graph) tasks in CEC with the objective of minimizing the average completion time of tasks. We consider the tasks to be heterogeneous in terms of the computation load of subtasks and input data. The tasks can set to be generated at any device at different release times. The formulated problem is shown to be NP-hard.
2) We propose a joint dependent task offloading and flow scheduling heuristic (JDOFH) which solves the problem by considering the start time of associated network flows to determine the offloading device for each subtask. JDOFH also leverages the global knowledge of all task graphs by considering each task graph as a set of cosubtask stages. The execution schedule is determined based on priority of each cosubtask stage.
3) We have conducted simulation experiments to evaluate the performance of JDOFH and compare it against other benchmark solutions. The performance comparison is done for both real application task graph of FFT with 4 points and randomly generated task graphs by varying different input parameters including the number of tasks, number of subtasks, number of devices, and communication-to-computation ratio of task graphs. The performance comparison JDOFH leads to up to 25% and 85% improvement in average completion time compared to joint scheduling solution based on list scheduling algorithms and other benchmark solutions respectively.

The rest of the paper is as follows. In Section 2, we discuss some related works. In Section 3, we give the system model and problem formulation. In Section 4, we describe the proposed solution, JPOFH, for the multi-hop dependent task offloading problem. In Section 5, we explain the results obtained during performance evaluation. Finally, we give the conclusion in Section 6.

2 Related Works

Existing works in literature have addressed different types of computation offloading problem. Table 1 gives a comparison of existing works on computation offloading. Most of existing works consider the single-hop computation offloading problem. Some recent works such as [9], [8], [10], [11], etc. have addressed multi-hop computation offloading problem. These works usually consider offloading of independent tasks to a single remote site and routing path selection in a multi-hop network. Compared to these existing works, this work considers offloading of multiple tasks, each consisting of dependent subtasks. Our work also considers scheduling of network flows arising due to transfer of data between dependent subtasks which has not been much explored in these works.

Scheduling and offloading problem of tasks with a directed acyclic graph (DAG) model has been studied extensively in the literature. The work in [23] proposed heterogeneous earliest finish time (HEFT) algorithm for placing a DAG task on heterogeneous processors. Many other works proposed similar list scheduling algorithms based on the work in [23]. Recently some have studied the problem of scheduling dependent tasks for various scenarios including single DAG task [12], multiple DAG tasks [13], within a cluster [14], across geo-distributed clusters [15], etc. The work in [14] proposes a scheduler, Graphene, to place multiple tasks with dependencies within a cluster by identifying the troublesome subtasks within each task. The work in [15] proposes Tetrium for scheduling tasks with dependencies in geo-distributed clusters by considering both computation and network resources. The work in [12] gave a lower bound solution for scheduling a single DAG task. Another recent work [21] has proposed a solution to schedule multiple DAG tasks by proposing a new abstraction branch and considering the urgency of different branches within a DAG

TABLE 1: Comparison of existing works

Existing works	Multiple tasks	Dependency-aware	Bandwidth-aware	Joint network flow decision
[14] (2016)	No	Yes	No	No
[16] (2016)	Yes	Parallel stages	Yes	No
[12] (2018)	No	Yes	No	No
[15] (2018)	Yes	Parallel stages	Yes	No
[20] (2018)	No	Yes	Yes	Yes
[13] (2019),	Yes	Yes	No	No
[21] (2019)	Yes	Yes	No	No
[9] (2019)	Yes	No	Yes	No
[10] (2019)	Yes	No	Yes	No
[22] (2020)	Yes	Yes	No	No
This paper	Yes	Yes	Yes	Yes

task. These works, however, either do not jointly consider network flow scheduling such as in [14], [12], [21], or do not take dependencies among subtasks to make decisions [15]. Another recent work [24], similar to our work, also considers different stages within a DAG task to make scheduling decisions. However, this work [24] does not consider multiple DAG tasks and network flow scheduling.

Some works in literature such as [16], [25], [26], [27], etc. have considered network bandwidth while making task scheduling decisions. The work in [16] assumes the underlying network scheduler to make task scheduling decisions. Another work [25] considers joint reducer placement and coflow scheduling problem. However, compared to our work, these works do not consider multiple DAG tasks and jointly consider network flow scheduling. There are few other works such as [28] and [29] which also consider cotask stages similar to our work. Here, the cotask is usually defined as a set of independent tasks related to each other by a common application. The work in [28] considers each DAG job as stages of cotasks and makes task scheduling decisions. However, it does not consider network flow scheduling. The work in [29] considered cotask offloading problem for independent tasks and did not consider network flow scheduling.

Our previous work in [20] studied data-aware task allocation problem while jointly considered network flow scheduling. However, the work in [20] made scheduling decision for a single task DAG and did not consider heterogeneous release time of tasks. Besides, the work in this paper also proposes a new heuristic solution where the task offloading and network flow scheduling decision is made in a single step for each subtask as opposed to different steps in [20].

3 System Model and Problem Formulation

This section first describes the system model including network and application model and then the problem formulation.

3.1 System Model

Fig 1 shows the system architecture of Edge Mesh based on collaborative edge computing paradigm where the intelligence is distributed and pushed within the network by sharing computation resources and data between mesh network of edge devices [7]. Edge devices is such an architecture can be heterogeneous in computation capacity and can also serve as routers, as shown in Fig 1. Due to the heterogeneity of devices, computation tasks can be offloaded to an edge device at a multi-hop distance.

The system architecture includes an SDN controller which is assumed to have global knowledge by collecting network and task-related information from all the edge devices and routers. The role of SDN controller is to act as a centralized controller with global knowledge, which is responsible for making the decision for offloading tasks and scheduling flows in the network. It includes different functional components responsible for collecting information and making scheduling decisions, as shown in Fig 1. There are some previous works such as [30], [31], etc. which have used the SDN controller to make scheduling decisions in wireless networks. The work in [32] proposed meSDN to extend the control of SDN to mobile devices. The work in [33] implemented a prototype for the proposed algorithm in [31].

Although the system model assumes that edge devices are connected using a wireless network, we have not fully considered all the issues due to dynamics in a wireless network such as spatial and temporal variation of wireless channel conditions, interference of wireless transmission among neighbouring devices [20]. Nevertheless, these issues in a wireless network should be considered as part of future work. The problem formulation in this work has been done assuming a static network condition. The problem is solved for an offline setting where we assume the information for all the tasks and network are already known. In practice, the cost and other information of executing the tasks on the different devices can be obtained using an application profiler [34] [35]. Furthermore, the offloading problem in this paper is solved for DAG tasks; however, we do not focus on how the application is modelled as a dependency graph. The work in [36] surveys different works on application profiling and partitioning. Different algorithms have been proposed in the literature for graph-based modelling based on the type of graph. The works such as [37], [38], etc. describe the profiling and partitioning method for graph-based modelling.

The objective of the problem is to minimize the average completion time of all the tasks. We have included both communication and computation cost to make task offload-

ing and network flow scheduling decisions. The computation cost includes both waiting time at the devices and time to execute the task. The communication cost includes waiting time to start the data transmission, and data transmission cost. We do not include propagation time in communication cost as it is usually very small. Further, we also ignore the switching cost for routing between subsequent links in the multi-hop path. In practice, these costs would influence the total cost; however, we ignore these costs to simplify the system model. Other works such as [12], [10], [22] etc. have used similar assumptions to calculate the total cost.

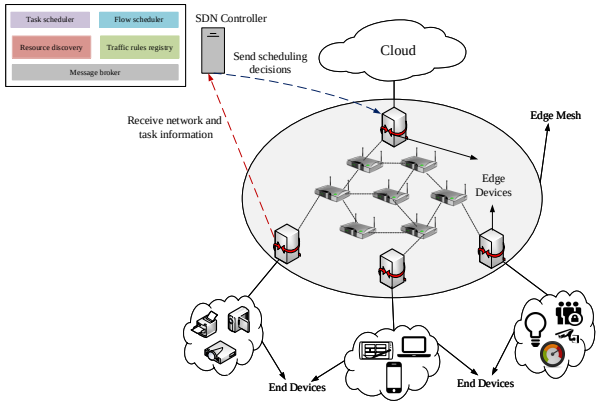


Fig. 1: System Architecture

The network and application model used in formulating the problem are:

Network model: The communication network is a mesh network of edge devices, shown to Edge Mesh circle in Fig 1, connected to each other using a multi-hop path. The communication network is modelled as a connected graph G = (V, E), where V is the set of devices, V = $\{k|1 \leq k \leq M\}$, and E is the set of links connecting different devices, E = $\{e_{kw}|k, w \in V\}$. Here, M is the total number of devices. In the problem description, we sometimes neglect the subscript and denote the link as e. The weight of each device is PS_k which represents the processing speed of each device k and weight of link e_{kw} represents the bandwidth between devices k and w. The devices can be heterogeneous in processing speed.

The network is assumed to be connected, i.e. there is at least one routing path between any two devices in the network. We assume that the shortest routing path is used to route the data between the two devices. The shortest path can be found using Djikstra's algorithm or another shortest path algorithm. A binary parameter Y_{kwe} (1 for yes, 0 for no) is used to represent if an edge e lies on routing path between device k to device w. The number of hops and bandwidth of the routing path between devices k and w is represented by H_{kw} and R_{kw}, respectively.

Application model: The application model consists of a set of tasks, A = $\{i|1 \leq i \leq O\}$, where O is the total number of tasks. Each task i is modelled as a DAG $B_i = (T_i, P_i)$, where T_i is the set of dependent subtasks in task i, $T_i = \{j|1 \leq j \leq N_i\}$, and P_i is set of dependencies between the subtasks in task i. Here, N_i is the number of subtasks in task i. We do not make an assumption on the dependency among subtasks in the DAG. Each DAG can have general dependency as shown by an example DAG task in Fig 2. Each task i is assumed to be generated at an edge device $z_i|z_i \in V$ at release time $Trel_i$. The amount of input data required for task i is ID_i. Each subtask j in the task i is associated with a computation load CL_{ij}. The weight of link connecting subtasks j and v of task i is D_{ijv}, which represents the amount of data to be transmitted if the dependent subtasks j and v are executed on different devices. The set of predecessors and successor subtasks for subtask j in task i is represented by Pd_{ij} and Sc_{ij} respectively.

We have assumed each task DAG includes an additional dummy subtask corresponding to the input data of the task. This dummy subtask can be inserted at the start without violating the original task DAG. The total number of subtask can be changed to $N_i^{'}$, where $N_i^{'} = N_i + 1$. The dummy subtask is numbered $N_i^{'}$ and is connected to subtasks without predecessors in original task DAG. The computation load of dummy subtask, i.e. $CL_{iN_i^{'}}$, is set equal to zero and the weight of link connecting dummy subtask with a successor subtasks in the task DAG, i.e. $D_{iN_i^{'}Sc_{iN_i^{'}}}$, is set as the amount of input data of the task.

Some of the assumptions made in the problem formulation are:

1) The routing path is assumed to be known.
2) The bandwidth for each flow is assumed to be given and equal to the minimum bandwidth of a link in the routing path.
3) Each device can execute one task at a time, while other tasks wait in the queue at the device.
4) Preemptive scheduling of tasks is not allowed.
5) No two flows are allowed to pass through a link at the same time to consider the interference among simultaneous wireless transmissions.

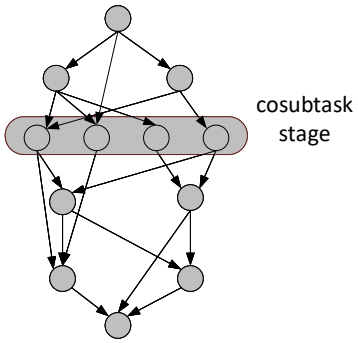


Fig. 2: Example DAG Task

3.2 Problem Formulation

This section describes the constraints and formulates the problem as an optimization problem. Table 2 summarizes the notations used in the paper.

TABLE 2: Notations used in the Paper

G = (V, E)	network model, where V is the set of devices and E is the set of edges
e_{kw}	link connecting device k and w
PS_k	processing speed of device k
Y_{kwe}	binary parameter to represent if an edge e lies on the routing path between device k to device w
H_{kw}	number of hops in the routing path between devices k and w
R_{kw}	bandwidth of the routing path between devices k and w
A	set of tasks
$B_i = (T_i, P_i)$	DAG of task i, where T_i is the set of dependent subtasks and P_i is the set of edges
CL_{ij}	computation load of subtask j in task i
D_{ijv}	amount of dependent data between subtask j and v in task i
Pd_{ij}	predecessor subtasks of subtask j in task i
Sc_{ij}	successor subtasks of subtask j in task i
z_i	device where task i is generated
$Trel_i$	release time of task i
M	number of devices
O	number of tasks
N_i	number of subtasks in task i
$\overline{N}$	maximum number of subtasks in all task graphs
$\overline{P}$	maximum number of edges in all task graphs
X_{ijk}	binary variable to represent if subtask j of task i is executed on device k
St_{ivj}	variable to represent start time of data flow between subtasks v and j of task i
Flt_{ivj}	variable to represent finish time of data flow between subtasks v and j of task i
Ts_{ij}	variable to represent start time of subtask j in task i
Tf_{ij}	variable to represent finish time of subtask j in task i
F_i	variable to represent the completion time of task i
Tft_i	variable to represent the time instance the task i is finished
Tas_{ijk}	start time of the subtask j in task i at device k
Taf_{ijk}	finish time of the subtask j in task i at device k
$Tcomm_{ivjk}$	start time of flow from subtask v to subtask j in task i with destination at device k
$rval_{ij}$	rank of subtask j in task i
$rank_{is}$	rank of cosubtask stage s in task i
U_{is}	set of subtasks in cosubtask stage s in task i
S	set of all cosubtask stages
i, p	index used to represent task i and p
k, w, l, m	index used to represent device k, w, l, and m
j, v, r, q	index used to represent subtasks j, v, r and q

3.2.1 Constraints

The task offloading decision includes making a decision on when and where each subtask within a task is offloaded. The constraints associated with task offloading are described in equation (1)-(5). Each subtask is offloaded to exactly one device as represented by equation (1).

$$\sum_{k \in V} X_{ijk} = 1, \qquad \forall i \in A, \quad j \in T_i \tag{1}$$

The problem assumes that processor sharing is not allowed therefore, only one subtask can be executed at one device at one time as represented by equation (2).

$$|X_{ijk} - X_{pqk}| * L + (Ts_{ij} - Tf_{pq})) * (Tf_{ij} - Tspq) >= 0 \quad \forall i, p \in A, \quad j \in T_i, \quad q \in T_p, \quad k \in V \tag{2}$$

Since each task contains dependent subtasks, a subtask can start only after preceding subtasks have finished as represented by equation (3).

$$Ts_{ij} \geq Tf_{iv} \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \tag{3}$$

In case the dependent subtasks are executed on different devices, a subtask can start after receiving dependent data from the preceding task as represented by equation (4).

$$Ts_{ij} \geq Flt_{ivj} \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \tag{4}$$

The relationship between finish time and start time of a subtask is represented by equation (5). The finish time of a subtask is equal to sum of start time and time to execute the subtask at the device.

$$Tf_{ij} = Ts_{ij} + \frac{CL_{ij}}{PS_k} * X_{ijk} \quad \forall i \in A, \quad j \in T_i, \quad k \in V \tag{5}$$

There are some additional constraints, equation (6)-(8), added to satisfy the schedule of the inserted dummy subtask. The dummy subtask is executed at the device the task is generated and started when the task is released.

$$X_{iN'_i z_i} = 1, \quad \forall i \in A \tag{6}$$

$$Ts_{iN'_i} = Trel_i, \quad \forall i \in A \tag{7}$$

$$Tf_{iN'_i} = Trel_i, \quad \forall i \in A \tag{8}$$

Network flow scheduling also involves separate constraints that are dependent on the task offloading decision. In this work, we consider network flow scheduling as deciding the start time of flows to transfer data between dependent subtasks executed on different devices. The lower bound of the start time of a flow is the release time of task as represented in equation (9).

$$St_{ivj} = Trel_i \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \tag{9}$$

The network flow start only after the preceding subtask has finished as represented by equation (10).

$$St_{ivj} = Tf_{iv} \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \tag{10}$$

This work assumes that two flows cannot pass through a link at the same time as represented by equation (11). If any two flows pass through the same link then, one of the flows can start only after other flow is finished. This constraint also helps to preserve the bandwidth constraint.

$$|Y_{wke} * X_{ijk} * X_{ivw} - Y_{lme} * X_{prl} * X_{pqm}| * L + (St_{ivj} - Flt_{prq}) * (Flt_{ivj} - St_{prq}) >= 0 \quad \forall i, p \in A, \quad j, v \in T_i, \quad r, q \in T_p, \quad k, w, l, m \in V \tag{11}$$

The finish time of a flow can be calculated directly once the start time is known. Equation (12) represents the relationship between start time and finish time of a data flow.

$$Flt_{ivj} = St_{ivj} + \frac{D_{ivj} * H_{wk}}{R_{wk}} * X_{ijk} * X_{ivw} \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij}, \quad k, w \in V \tag{12}$$

The task is considered to be finished when the last subtask within the task has finished execution. We do not consider the time to send any output data from the last subtask. This can be included, if required, by adding a dummy subtask as done in [12]. The completion time of a task, represented by equation (13), is the difference between the time instance when the last subtask is finished and the time instance when the task is released.

$$F_i = \max_{j=1,\ldots N_i'} Tf_{ij} - Trel_i, \quad \forall i \in A \tag{13}$$

The constraint of the range of binary decision variable X_{ijk} is represented by equation (14).

$$X_{ijk} = \{0, 1\}, \quad \forall i \in A, \quad j \in T_i, \quad k \in V \tag{14}$$

3.2.2 Optimization Problem

The objective function of the problem is to minimize the average completion time of all tasks. The objective function considers the time instant each task is finished (Tft_i) rather than the time period (F_i) to complete the task. As $Trel_i$ is a constant value, it is equivalent to minimizing the time period to complete the task. The multi-hop dependent task offloading problem, **P1**, can be formatted as:

$$\underset{X_{ijk}, Tf_{ij}, Ts_{ij}, St_{ivj}, Flt_{ivj}, F_i}{\text{minimize}} \frac{\sum_{i=1}^{J} Tft_i}{J}, \tag{15}$$

subject to (1) - (14) $\quad \forall i \in A, \quad j, v \in T_i, \quad k \in V$

This problem can be reduced to shop-scheduling problem [39] for a fully connected network and not considering the offloading of subtasks. In the reduced problem, each DAG task refers to a job, and the devices are referred to as machines. The subtasks have a set of precedence order similar to one for operations in the job. The shop-scheduling problem has been proven to be NP-hard for 3 three jobs in [39]. Hence, the problem in this paper is also NP-hard. Since we cannot find an optimal solution in polynomial time. Therefore, we have proposed a heuristic solution.

4 Joint Dependent Task Offloading and Flow Scheduling Heuristic (JDOFH)

We have proposed a joint dependent task offloading and flow scheduling heuristic (JDOFH) that determines the execution schedule which includes the decision on the start time of execution, offloading device, and the start time of input data flow for each subtask. JDOFH is developed considering two main principles:

1) Leverage information of both parallelism and dependency with each DAG task: The parallelism among subtasks is leveraged by considering each task as a set of cosubtask stages, as shown in Fig 2. The dependency information is utilized by assigning each cosubtask stage a priority based on the release time of tasks and maximum computation load of a subtask within a cosubtask stage. Scheduling decisions are made in the increasing order of priority metric. Existing methods usually make scheduling decisions based on the order of task release time or earliest task finish time which requires other tasks to wait even if a subtask from another task can be scheduled in the meantime. The use of both parallelism and dependency information using cosubtask stages helps in creating a better execution schedule. It allows subtasks from multiple tasks to be scheduled while considering both the release time and finish time of tasks.
2) Make joint task offloading and network flow scheduling decision: The offloading decision for each subtask within a task is made considering the start time of input data flows. Flows are scheduled based on the priority of corresponding subtasks. A joint task offloading and flow scheduling decision leads to better performance, in terms of completion time.

The algorithm first creates a top-to-bottom rank of each subtask, val_{ij}, within a task using the equation (17). The rank metric for each cosubtask stage within a task is calculated as defined in equation (18).

$$val_{ij} = \max_{v \in Pd_{ij}} (val_{iv} + \frac{D_{ivj}}{\overline{BW}}) + \frac{CL_{ij}}{\overline{PS}}, \quad \forall i \in A, \quad j \in T_i \tag{16}$$

$$rval_{ij} = val_{ij} + Trel_i \quad \forall i \in A, \quad j \in T_i \tag{17}$$

$$rank_{is} = \max_{j \in U_{is}} (rval_{ij}) \quad \forall i \in A \tag{18}$$

Cosubtask stages are given priority in the increasing order of rank metric calculated in Equation (18). Subtasks within the cosubtask stage with the highest priority are selected to be scheduled. However, since there can be multiple subtasks within a cosubtask stage, subtasks are selected in the decreasing order of $rval$. Executing a subtask with higher $rval$ first can help in minimizing the difference between the completion time of different subtasks within a cosubtask stage [28].

The schedule for the selected subtask is determined by selecting the device, which leads to minimum finish time, as shown in equation (22). However, in order to do that, we need to calculate the start time of the subtask at each device, as shown in equation (19). Start time is calculated by considering both the available time at the device and time to receive the input data from preceding subtasks. Here, the available time implies that the subtask can be scheduled to execute before other scheduled subtasks at the device if there is enough available time to execute the current subtask. In case, there is no such available time at the device; the available time is equal to the waiting time until other scheduled tasks are executed. The time to receive data from the preceding task needs to consider other network flows. We calculate the start time of sending the data from preceding subtask, as shown in Equation (20), by considering the assumption that no two flows can pass through a

link at the same time. The network flow corresponding to the current subtask is given higher priority; therefore, we only need to consider the flows corresponding to previous selected subtasks.

$$Tas_{ijk} = \max\{Tavail_{ijk}, \max_{v \in Pd_{ij}} (Tcomm_{ijvk} + \frac{D_{ivj} * H_{wk}}{R_{wk}})\} \quad (19)$$
$$\forall i \in A, \quad j, v \in T_i, \quad k \in V$$

$$Tcomm_{ivjk} = \max\{Tf_{iv}, \max_{e \in P_{wk}} (Y_{wke} * Y_{lme} * Flt_{prq})\} \quad (20)$$
$$\forall i, p \in A, \quad j, v \in T_i, \quad r, q \in T_p, \quad k, w, l, m \in V$$

The finish time of the subtask for different devices is calculated by adding the cost to execute the subtask at the device to the start time as shown in Equation (21). The device corresponding to the minimum finish time calculated in equation (22) is selected for executing the subtask as shown in Equation (23).

$$Taf_{ijk} = Tas_{ijk} + CT_{ijk}, \quad \forall i \in A, \quad j, v \in T_i, \quad k \in V \quad (21)$$

$$Tf_{ij} = \min_{k \in V} Taf_{ijk}, \quad \forall i \in A, \quad j \in T_i \quad (22)$$

$$X_{ijk^*} = 1, \quad \forall i \in A, \quad j \in T_i \quad (23)$$

The start time of executing the subtask, start time and finish time of input data flows corresponding to the subtask can be determined accordingly as shown in equations (24) to (26).

$$Ts_{ij} = Tas_{ijk^*}, \quad \forall i \in A, \quad j \in T_i \quad (24)$$

$$St_{ivj} = Tcomm_{ijvk^*}, \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \quad (25)$$

$$Flt_{ivj} = St_{ivj} + \frac{D_{ivj} * H_{wk^*}}{R_{wk^*}} \quad (26)$$
$$\forall i = 1, ...J, \quad j = 1, ...N_i, \quad v \in Pd_{ij}$$

The details of the proposed heuristic solution are given in Algorithm 1. The algorithm uses the three principles and returns the execution schedule of all subtasks within the tasks, including the start time and finish time of corresponding network flows. The finish time instance for each task can be determined by taking the maximum of the finish time of all subtasks within the task (Line 26-28).

The computation complexity of proposed heuristic is $\mathcal{O}(|A|^2 * \overline{N}^2 * |V|^2 * \overline{P})$. The computation complexity is calculated by considering the most complex operation in the proposed heuristic, which is the calculation of $Tcomm_{ijvk}$ in line 13. There are four for loops for calculating $Tcomm_{ijvk}$ one loop each corresponding to the number of stages (S), number of subtasks in a stage ($\overline{N}$ in worst case), number of devices ($|V|$), and number of preceding tasks ($\overline{N} - 1$ in the worst case). The first two for loops correspond to the total number of subtasks in all tasks which is equivalent to $|A| * \overline{N}$ in the worst case. Besides, the calculation of $Tcomm_{ijvk}$ requires maximum operation over different values of the finish time of flows corresponding to previous subtasks, i.e. $|A| * \overline{P}$ which is calculated by considering all the edges in all previous tasks in the worst case, and all the edges in the routing path, i.e. $|V| - 1$ in the worst case. Hence the complexity of proposed heuristic is$\mathcal{O}(|A|^2 * \overline{N}^2 * |V|^2 * \overline{P})$.

Algorithm 1: Joint Dependent Task Offloading and Flow Scheduling Heuristic (JDOFH)

Input: The set of J tasks with task graph model consisting of dependent subtasks, the network of M edge devices

Output: The execution schedule specifying the selected device, start time and finish time of executing the subtask, and the start time and finish time of each input data flow

1 $S_{ivj} \leftarrow Trel_i, \quad \forall i = 1, ...J, j, v = 1, ...N_i$;
2 $X_{iN'_i z_i} \leftarrow 1, \quad \forall i = 1, ...J$;
3 $Ts_{iN'_i}, Tf_{iN'_i} \leftarrow Trel_i, \quad \forall i = 1, ...J$;
4 Create an index I of cosubtask stages in increasing order of rank metric;
5 **for** $t \leftarrow 1$ *to* $|S|$ **do**
6 $s \leftarrow I(t)$;
7 Create an index L of subtasks in cosubtask stage s in increasing order of rval metric;
8 **for** $q \leftarrow 1$ *to* $|U_{is}|$ **do**
9 $j \leftarrow L(q)$;
10 **for** $k \leftarrow 1$ *to* $|V|$ **do**
11 **for** $v \leftarrow 1$ *to* $|Pd_{ij}|$ **do**
12 Calculate $Tcomm_{ivjk}$ using equation (20);
13 **end**
14 Calculate Tas_{ijk} using equation (19);
15 Calculate Taf_{ijk} using equations (21);
16 **end**
17 Calculate k^*, Tf_{ij} based on $\min_{k=1,...M} Taf_{ijk}$;
18 $X_{ijk^*} \leftarrow 1$;
19 $Ts_{ij} \leftarrow Tas_{ijk^*}$;
20 **for** $v \leftarrow 1$ *to* $|Pd_{ij}|$ **do**
21 Calculate St_{ivj}, Flt_{ivj} according to equation (25) and (26);
22 **end**
23 **end**
24 **end**
25 **for** $i \leftarrow 1$ *to* J **do**
26 $Tft_i \leftarrow \max_{j \in V} Tf_{ij}$
27 **end**
28 **return** $X_{ijk}, Ts_{ij}, Tf_{ij}, St_{ivj}, Flt_{ivj}, Tft_i$;

5 Performance Evaluation

We have done simulation to evaluate and compare the performance of JDOFH with other benchmark solutions. The performance evaluation has been done for two performance metrics: average completion time and running time of the algorithm. The parameters used for simulation are in a similar range to the one used previously in [12] and [40].

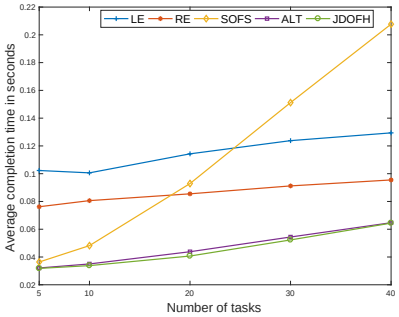


Fig. 3: Effect of changing number of tasks for FFT DAG

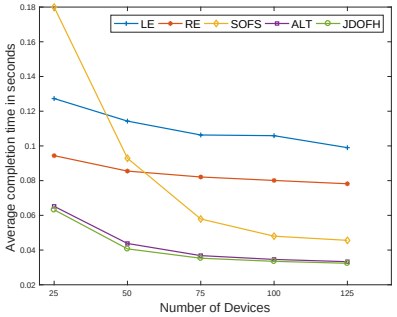


Fig. 4: Effect of changing number of devices for FFT DAG

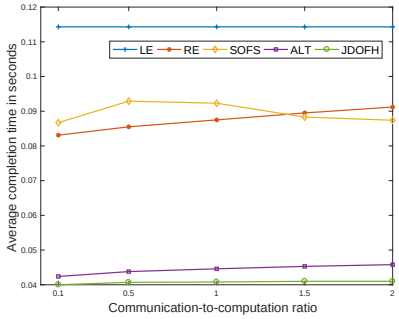


Fig. 5: Effect of changing CCR of task graph for FFT DAG

5.1 Simulation Parameters

Parameters for Network Model: We generate a network of edge devices where devices are deployed randomly using uniform distribution. The size of the area is selected to be $M \times M$ square units, and any two devices less than $2 * M/5$ units apart are connected to each other. The distance between devices is set to be in a similar range as done in previous works such as [40] and [20]. However, compared to the fixed-size area used in these works, a variable area size makes it easier to create connected mesh network topology even with a low number of devices. Besides, maintaining a similar network density using variable area size helps in avoiding network topology with too little or too much network links. All the devices are connected to each other using a multi-hop path to form a connected graph. Each vertex in the graph represents a device, and its weight represents the processing power of the device. The weight of the link (edge) connecting two devices (vertices) represents the bandwidth capacity of the link (edge). The devices are heterogeneous in terms of processing power which is selected from a normal distribution with mean 50MCPS (Million Cycles Per Second) and variance 20%. The bandwidth of each link is selected from a normal distribution with mean 20Mbps and variance 20%.

Parameters for Application Model: The J tasks in the application model are generated at a device selected randomly. We implemented a random DAG generator for the task graph using the layer-by-layer method mentioned in [41]. The parameters used for generating the task graph are number of tasks (nodes), the height of task graph (number of layers), number of tasks in each layer, and the edges between the tasks in different layers. The number of nodes in i^{th} task graph is selected to be a normal distribution in the range [1, N_i]. The number of layers in the i^{th} task graph is selected to be a normal distribution in the range [1, $\frac{N_i}{2}$]. The value of N_i is selected to be 50 for the default case. Each layer in the task graph is constrained to have at least one node, and the number of nodes in each layer is selected randomly. The number of edges between two consecutive layers is determined using a uniform distribution. The weight of the node in the task graph represents the computation load of subtask, which is selected from a normal distribution with mean 300KCC (Kilo Clock Cycles) and variance 20%. The amount of input data for each task and data transferred between two dependent subtasks within a task graph is selected from a normal distribution with mean 120 kilobits and variance 20%. The amount of data to be transferred is calculated based on the communication-to-computation ratio (CCR) of 0.5.

5.2 Benchmark Solutions

We have proposed some benchmark solutions based on the ideas of solutions in existing works to compare the performance of JDOFH.

1) Local Execution (LE): LE solution is obtained by executing the task at the local device where the task is generated. Compared to JDOFH, LE is easy to obtain as it does not require consideration of network flow scheduling. The tasks are scheduled in the order of release time.
2) Remote Execution (RE): RE solution is obtained by considering each task with multiple subtasks as a single unit. Each task is scheduled, in the order of release time, by greedy offloading to a remote device such that the completion time of the task is minimized. Similar to LE, RE also does not require consideration of network flow scheduling for data transfer between subtasks. However, since each task is associated with input data, there are network flows while offloading the task to a remote device. RE uses the first-come-first-serve (FCFS) approach to schedule the network flows by pausing other contending flows. The scheduling order for each flow is determined based on the scheduling order of corresponding DAG tasks.
3) Separate task offloading and network flow scheduling (SOFS): SOFS solution is obtained by first solving the task offloading problem while ignoring the bandwidth constraint and then solving the network flow scheduling problem. Task offloading problem is solved based on the priority order of the earliest release time. The offloading solution for each subtask is obtained assuming there is no other network flow at that time. We use a list scheduling algorithm similar to HEFT [23] for task offloading. Network flow scheduling is done based on the priority order of earliest deadline first (EDF) approach, used in flow scheduling algorithm PDQ [42], by pausing other contending network flows. The deadline for each flow is determined based on the scheduling order of corresponding subtasks determined in the previous step.

4) Joint Scheduling based on task release time (ALT): ALT is an alternative solution that jointly solves the dependent task offloading and network flow scheduling problem. ALT determines the execution schedule for each task based on increasing order of task release time. ALT solution is similar to an online solution where the information of future incoming tasks is not known, and hence the tasks are scheduled in the order of release time. Compared to JDOFH where each task is considered as a set of cosubtask stages, ALT determines the top-down rank for each subtask similar to a list scheduling algorithm. ALT schedules the tasks sequentially unlike JDOFH where cosubtask stages from different tasks can interleave each other. Network flow scheduling is done based on the scheduling order of corresponding subtasks, similar to JDOFH. The network flow scheduling is similar to scheduling in order of earliest deadline first (EDF) [42] as subtask with earliest start time will have the earliest deadline for the corresponding network flow.

5.3 Simulation Results for Real Application Task Graph

The real application task graph used for performance comparison is FFT DAG with 4 points used in many other works such as [23], [12], etc. The number of nodes, i.e. N, in FFT DAG is 15. The computation load of each subtask and weight of edges is the same setting mentioned earlier for the application model. The characteristics of FFT DAG is that both the weight of all nodes at the same level and the weight of all edges from nodes at the same level is equal. Table 3 gives the performance comparison of JDOFH against benchmark solutions for default parameters. In the default case, the number of FFT DAG, i.e. J, is 20; CCR is each FFT DAG is 0.5; and the number of devices in the network, i.e. M, is 50. The values have been averaged for 30 iterations, and the error margin is calculated for 95% confidence interval. The input values for the task graph and network model in each iteration are different and generated randomly.

Table 3 shows JDOFH performs better than the four benchmark solutions in terms of average completion time. There is a significant difference in performance between JDOFH and three benchmark solution (LE, RE, SOFS) that do not make joint task offloading and scheduling decision. JDOFH is around 64.39% better than LE, 52.39% better than RE, and 56.18% better than SOFS in terms of completion time for the default parameter setting. Both JDOFH and ALT, which make joint decision perform better than other benchmark solutions. JDOFH is around 7% better than ALT in terms of completion time as ALT makes the scheduling decision sequentially after completion of each task, however, JDOFH can interleave cosubtask stages for different tasks which reduces the average completion time. Table 3 also shows a comparison between JDOFH and other benchmark solutions in terms of running time of the algorithm. We can see that JDOFH running time is close to ALT. As expected, the running time of JDOFH is higher than the other three benchmark solutions, but it is still within range as other benchmark solutions. There is a trade-off between the two performance metrics; however, this paper focuses on proposing a better solution in terms of completion time.

We have also done performance comparison, in terms of completion time, by varying different simulation parameters. Fig 3, 4, and 5 show the performance comparison by changing the number of tasks, the number of devices, and CCR of FFT DAG, respectively. We can observe that JDOFH performs better than other benchmark solutions for all different range of parameters. Another main observation is that there is a significant change in performance difference between SOFS and JDOFH for both a large number of tasks and a low number of devices. The reason is that SOFS makes a separate decision, so its performance deteriorates significantly when network flows have to compete for bandwidth resources such as during both a large number of tasks and a low number of devices. JDOFH, on the other hand, performs better, and there is a similar performance difference as observed for default parameter setting.

5.4 Simulation Results for Randomly Generated Task Graphs

Besides using FFT DAG, the performance of JDOFH has also been evaluated by using randomly generated task graphs. The input parameters and details related to the task graph are mentioned in Section 5.1. Compared to FFT DAG, the number of subtasks in each randomly generated DAG can be higher and different. Table 4 shows the performance comparison with default parameters for randomly generated DAG. The default parameters used are: number of task DAG, i.e. J, is 20; the maximum number of subtasks, i.e. N_i, is 50, CCR of each task DAG is 0.5; and the number of devices in the network, i.e. M, is 50. The values have been averaged for 30 iterations with an error margin for 95% confidence interval. Similar to results observed for FFT DAG, JDOFH performs better than other benchmark solutions. There is a significant performance difference between JDOFH and three benchmark solutions (LE, RE, SOFS) that make task offloading and network flow scheduling decisions separately. JDOFH is around 58.39% better than LE, 43.33% better than RE, and 74.58% better than SOFS in terms of average completion time of tasks. Although both JDOFH and ALT jointly solve the problem, JDOFH is around 17% than ALT in terms of average completion time. These results show the benefit of making a joint decision and better performance of the proposed solution JDOFH compared to ALT. We can observe there is significant error margin as input values for each iteration are generated randomly. Besides, the number of subtasks in each task graph is selected from a uniform distribution which leads to a significant difference in results over different iterations.

The rest of this section gives detailed performance comparison, in terms of average completion time, by changing difference input parameters including the number of tasks, number of subtasks, number of devices, and CCR of task graph. The different sub-sections describe the reasons behind change in performance on varying different parameters.

5.4.1 *Effect of change in the number of tasks*

Fig 6 shows the effect of changing the number of tasks from 5 to 40 on average completion time. There is an increase in average completion time for all algorithms due to both

TABLE 3: Performance Comparison with default parameters for FFT DAG

Metric	LE	RE	SOFS	Alt	JDOFH
Completion time (sec)	0.1143 ± 0.0047	0.0855± 0.0009	00929 ± 0.0061	0.0438± 0.0006	0.0407 ± 0.0005
Running time (sec)	0.0023 ± 0.0029	1.0151 ± 0.0183	29.155 ± 0.2085	576.18 ± 6.7478	711.80 ± 6.6176

TABLE 4: Performance Comparison with default parameters for randomly generated DAG

Metric	LE	RE	SOFS	Alt	JDOFH
Completion time (sec)	0.1872 ± 0.0137	0.1372± 0.0072	0.3065 ± 0.0525	0.0939± 0.0067	0.0779 ± 0.005
Running time (sec)	0.0019 ± 0.0027	0.9996 ± 0.017	92.410 ± 10.785	3022.4 ± 487.72	3921.6 ± 641.47

increase in waiting time at the devices to execute the subtasks and increase in the total number of network flows.

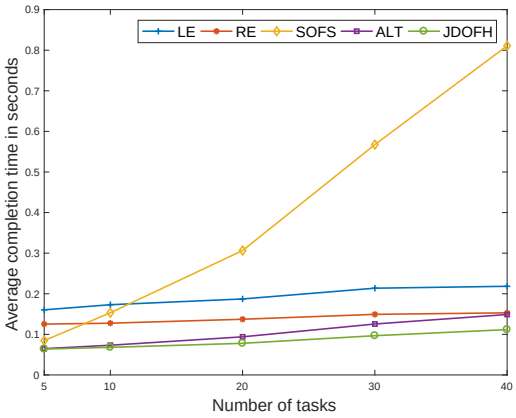


Fig. 6: Effect of changing number of tasks for randomly generated DAG

The performance difference between LE and JDOFH decreases from 60.22% at 5 tasks to 48.85% at 40 tasks. This decrease in the gap is observed because LE is not affected by an increase in the number of network flows when the number of tasks are increased. RE also shows a similar performance trend as LE where the performance difference decreases from 49.08% at 5 tasks to 27% at 40 tasks. Compared to LE, RE can offload the complete task which can help in comparatively reducing the waiting time at the devices. However, since both LE and RE are not affected by an increase in network flows, there is a decrease in the performance gap. JDOFH shows increase in performance difference with SOFS from 24.59% at 5 tasks to 86.22% at 40 tasks. This significant increase in performance difference is similar to the one observed for FFT DAG. Compared to JDOFH, SOFS performance deteriorates rapidly with an increase in the number of tasks as SOFS separates the network flow scheduling decision. The difference in average completion time between JDOFH and ALT also increases from 1.54% at 5 tasks to 24.98% at 40 tasks. In case of a low number of tasks, there is less contention among resources, so both JDOFH and ALT perform almost equivalently. However, as the number of tasks is increased, JDOFH performs better as it leverages cosubtasks stages to determine the execution schedule. ALT, on the other hand, requires considerable waiting time at the devices to finish each task sequentially based on the order of release time.

5.4.2 *Effect of change in the number of subtasks*

We have evaluated the effect of changing the number of subtasks within each randomly generated task DAG as shown in Fig 7. As mentioned earlier, the number of subtasks is selected from a uniform distribution in the range [1, N_i]. We observe the performance trend by changing the value of N_I, i.e. maximum subtasks in a task DAG, from 10 to 100. All algorithms show increase in average completion time as there is increase of subtasks in each task DAG which leads to both increase in number of network flows and waiting time at the devices.

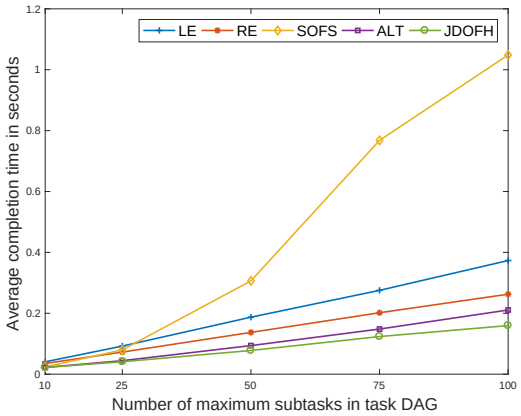


Fig. 7: Effect of changing number of subtasks for randomly generated DAG

The difference in average completion time between LE and JDOFH increases from 46.06% at 10 maximum subtasks to 57.25% at 100 maximum subtasks. Compared to JDOFH, LE executes the task locally at the devices it is generated, which leads to larger waiting time as the number of subtasks are increased. The performance difference between JDOFH and RE increases slightly from 37.6% at 10 maximum subtasks to 39.24% at 100 maximum subtasks. Compared to LE, RE can offload the complete task to a remote device to reduce the waiting time. The average completion time for both JDOFH and RE increases at an approximately similar rate with an increase in the number of subtasks. As observed earlier, there is a significant increase in performance difference between JDOFH and SOFS from 7.98% at 10 maximum subtasks to 84.77% at 100 maximum subtasks. Compared

to JDOFH, SOFS does not perform well when the number of network flows increases and have to compete for the bandwidth resources. The performance difference between JDOFH and ALT also increases from 1.35% at 10 maximum subtasks to 24.28% at 100 maximum subtasks. As the number of subtasks increases, the completion time of each task is increases as well which leads to worse performance for ALT as tasks are scheduled sequentially. On the other hand, JDOFH is able to utilize the knowledge of all tasks and interleave cosubtasks stages to decrease average completion time.

5.4.3 Effect of change in the number of devices

Fig 8 shows the effect of changing the number of devices from 25 to 125 on the average completion time of tasks. It is expected that an increase in the number of devices decreases the average completion time due to availability of more resources. However, there is a marginal decrease in completion time after a certain number of devices.

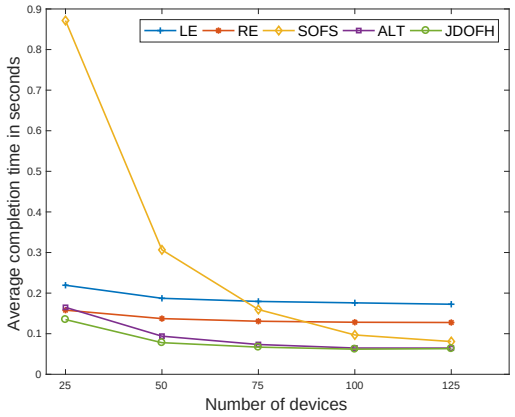


Fig. 8: Effect of changing number of devices for randomly generated DAG

There is an increase in performance difference between JDOFH and LE from 38.56% at 25 devices to 63.38% at 125 devices. LE is not able to leverage the increase in the availability of resources as each task is executed locally on the devices. RE shows a similar performance trend as LE where the difference in the average completion time of JPOFH increases from 14.68% at 25 devices to 50.47% at 125 devices. The benefit of an increase in the number of devices is more for JDOFH as each subtask within each task can be offloaded compared to offloading of a complete task in RE. The difference in average completion time between JDOFH and SOFS decreases from 84.53% at 25 devices to 21.68% at 125 devices. The availability of more resources with an increase in the number of devices leads to less contention among network flows for bandwidth. However, the benefit of joint task offloading and network flow scheduling decision is still apparent as JDOFH outperforms SOFS even with an increase in the number of devices. ALT shows a decrease in average completion time between JDOFH from 18.25% at 25 devices to 2.06% at 125 devices. An increase in the number of devices reduces the waiting time to execute the subtasks at the devices. Therefore, both ALT and JDOFH perform similarly when abundant resources are available.

5.4.4 Effect of change in communication to computation ratio (CCR)

Fig 9 shows the performance comparison done for different types of tasks ranging from computation-intensive tasks (CCR value equal to 0.1) to communication-intensive tasks (CCR value equal to 2). It is expected that average completion time of tasks increases as communication cost will increase on increasing the CCR of each task DAG. However, solutions such as LE and RE do not show much increase as they do not consider offloading of subtasks.

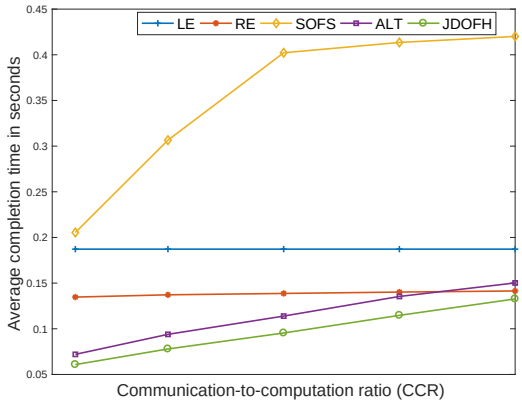


Fig. 9: Effect of changing CCR of task graph for randomly generated DAG

The difference in completion time between JDOFH and LE decreases from 67.52% at 0.1 CCR to 29.11% at 2 CCR. Since LE executed the tasks at the devices locally, it is not affected by increasing the CCR leading to a decrease in the performance difference. RE also shows a similar performance trend where the difference in average completion time between JDOFH decreases from 54.86% at 0.1 CCR to 6.15% at 2 CCR. Similar to LE, RE executes the task as a whole and offloads them to a remote device leading to a further decrease in the average completion time of tasks. This decrease in performance trend for both LE and RE is to be expected as when communication cost becomes significantly higher than computation cost, the benefit of offloading each subtask is not useful. Therefore, after a certain threshold of CCR, both LE and RE would perform better than algorithms which do offloading of fine-grained subtasks. In such a case, an if-else condition could be used to utilize RE solution beyond the CCR threshold. It is to be noted that RE solution also utilizes the joint network flow scheduling for the flows corresponding to the input data of the tasks. Therefore, there is a benefit to joint task offloading and network flow scheduling as proposed in this work.

The performance difference between JDOFH and SOFS remains similar around 70% on changing the CCR from 0.1 to 2. The performance difference between JDOFH and ALT decreases slightly from 15.56% at 0.1 CCR to 11.65% at 2 CCR. In both cases, the effect of an increase in CCR does not significantly change the performance difference as the three solutions, i.e. SOFS, ALT, and JDOFH, rely on offloading of subtasks.

6 Conclusion

This paper studies the multi-hop dependent task offloading and network flow scheduling problem in CEC with the objective of minimizing the average completion time of tasks. The problem includes making a decision on when and where each subtask within a task is executed and the start time of each flow corresponding to data transfer between dependent subtasks. The problem is formulated as an MINLP optimization problem which is proven to be NP-hard. We have proposed a JDOFH algorithm that leverages the knowledge of each task graph and start time of flow to make task offloading decisions. The performance of JDOFH has been comprehensively evaluated using simulation by considering both the real application task graph of FFT and randomly generated task graphs. The performance comparison has been done by varying different simulation parameters, including the number of tasks, number of subtasks, number of devices, and CCR of task graphs. We have compared the performance of JDOFH against different benchmark solutions considering local execution, remote execution, separate task offloading and network flow scheduling, and joint solution based on list scheduling. Performance comparison shows that JDOFH leads to up to 25% and 85% improvement in average completion time compared to a joint solution based on list scheduling algorithm and other benchmark solutions respectively.

We have solved the problem for an offline setting and evaluate it using simulation experiments. In the future work, we will implement the solution for an online setting, and illustrate the efficacy of solution using a real prototype. The proposed solution, JDOFH, schedules the cosubtask stages sequentially based on the priority list. This design makes it easier to extend JDOFH to an online setting where the priority list of cosubtask stages from different tasks can be updated regularly as new tasks are generated. The subtasks from highest priority cosubtask stage can then be offloaded using the same approach as JDOFH. Another direction for future work is to implement task offloading and network monitoring components in the SDN controller.

References

[1] L. U. Khan, I. Yaqoob, N. H. Tran, S. Kazmi, T. N. Dang, and C. S. Hong, "Edge computing enabled smart cities: A comprehensive survey," *arXiv preprint arXiv:1909.08747*, 2019.

[2] A. Saeed, M. Ammar, K. A. Harras, and E. Zegura, "Vision: The case for symbiosis in the internet of things," in *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*. ACM, 2015, pp. 23–27.

[3] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.

[4] K. Wang, H. Yin, W. Quan, and G. Min, "Enabling collaborative edge computing for software defined vehicular networks," *IEEE Network*, vol. 32, no. 5, pp. 112–117, 2018.

[5] H. Zhang, P. Dong, W. Quan, and B. Hu, "Promoting efficient communications for high-speed railway using smart collaborative networking," *IEEE wireless communications*, vol. 22, no. 6, pp. 92–97, 2015.

[6] L. Chen and J. Xu, "Socially trusted collaborative edge computing in ultra dense networks," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 9.

[7] Y. Sahni, J. Cao, S. Zhang, and L. Yang, "Edge mesh: A new paradigm to enable distributed intelligence in internet of things," *IEEE access*, vol. 5, pp. 16 441–16 458, 2017.

[8] H. Al-Shatri, S. Müller, and A. Klein, "Distributed algorithm for energy efficient multi-hop computation offloading," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[9] C. F. Funai, C. Tapparello, and W. Heinzelman, "Computational offloading for energy constrained devices in multi-hop cooperative networks," *IEEE Transactions on Mobile Computing*, 2019.

[10] Z. Hong, H. Huang, S. Guo, W. Chen, and Z. Zheng, "Qos-aware cooperative computation offloading for robot swarms in cloud robotics," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 4027–4041, 2019.

[11] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial iot-edge-cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[12] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 37–45.

[13] Y. Fan, L. Zhai, and H. Wang, "Cost-efficient dependent task offloading for multiusers," *IEEE Access*, vol. 7, pp. 115 843–115 856, 2019.

[14] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 81–97. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/grandl_graphene

[15] C.-C. Hung, G. Ananthanarayanan, L. Golubchik, M. Yu, and M. Zhang, "Wide-area analytics with multiple resources," in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–16.

[16] A. Munir, T. He, R. Raghavendra, F. Le, and A. X. Liu, "Network scheduling aware task placement in datacenters," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 2016, pp. 221–235.

[17] S. Gazzaz and F. Nawab, "Collaborative edge-cloud and edge-edge video analytics," in *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 484–484.

[18] C. Neff, M. Mendieta, S. Mohan, M. Baharani, S. Rogers, and H. Tabkhi, "Revamp2t: Real-time edge video analytics for multi-camera privacy-aware pedestrian tracking," *IEEE Internet of Things Journal*, 2019.

[19] M. P. Alves, F. C. Delicato, I. L. Santos, and P. F. Pires, "Lw-coedge: a lightweight virtualization model and collaboration process for edge computing," *World Wide Web*, pp. 1–49, 2019.

[20] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3512–3524, 2018.

[21] Z. Hu, D. Li, Y. Zhang, D. Guo, and Z. Li, "Branch scheduling: dag-aware scheduling for speeding up data-parallel jobs," in *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 1–10.

[22] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, and F. Yang, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things Journal*, 2020.

[23] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[24] W. Shao, F. Xu, L. Chen, H. Zheng, and F. Liu, "Stage delay scheduling: Speeding up dag-style data analytics jobs with resource interleaving," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–11.

[25] Y. Zhao, C. Tian, J. Fan, T. Guan, and C. Qiao, "Rpc: Joint online reducer placement and coflow bandwidth scheduling for clusters," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 2018, pp. 187–197.

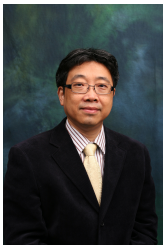
[26] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 421–434, 2015.

[27] L. Rupprecht, W. Culhane, and P. Pietzuch, "Squirreljoin: network-aware distributed join processing with lazy partitioning," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1250–1261, 2017.

[28] Y. Zhao, S. Luo, Y. Wang, and S. Wang, "Cotask scheduling in

cloud computing," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, 2017, pp. 1–6.
[29] Y.-H. Chiang, T. Zhang, and Y. Ji, "Joint cotask-aware offloading and scheduling in mobile edge computing systems," *IEEE Access*, vol. 7, pp. 105 008–105 018, 2019.
[30] C.-F. Liu, S. Samarakoon, M. Bennis, and H. V. Poor, "Fronthaul-aware software-defined wireless networks: Resource allocation and user scheduling," *IEEE Transactions on Wireless Communications*, vol. 17, no. 1, pp. 533–547, 2017.
[31] T. De Schepper, S. Latré, and J. Famaey, "A transparent load balancing algorithm for heterogeneous local area networks," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 160–168.
[32] J. Lee, M. Uddin, J. Tourrilhes, S. Sen, S. Banerjee, M. Arndt, K.-H. Kim, and T. Nadeem, "mesdn: Mobile extension of sdn," in *Proceedings of the fifth international workshop on Mobile cloud computing & services*, 2014, pp. 7–14.
[33] T. De Schepper, P. Bosch, E. Zeljkovic, K. De Schepper, C. Hawinkel, S. Latré, and J. Famaey, "Sdn-based transparent flow scheduling for heterogeneous wireless lans," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 901–902.
[34] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49–62.
[35] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE Infocom*. IEEE, 2012, pp. 945–953.
[36] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi, "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions," *Journal of Network and Computer Applications*, vol. 48, pp. 99–117, 2015.
[37] M. Smit, M. Shtern, B. Simmons, and M. Litoiu, "Partitioning applications for hybrid and federated clouds." in *CASCON*, vol. 12. Citeseer, 2012, pp. 27–41.
[38] J. Niu, W. Song, and M. Atiquzzaman, "Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications," *Journal of Network and Computer Applications*, vol. 37, pp. 334–347, 2014.
[39] Y. N. Sotskov and N. V. Shakhlevich, "Np-hardness of shop-scheduling problems with three jobs," *Discrete Applied Mathematics*, vol. 59, no. 3, pp. 237–266, 1995.
[40] J. Yang, H. Zhang, Y. Ling, C. Pan, and W. Sun, "Task allocation for wireless sensor network using modified binary particle swarm optimization," *IEEE Sensors Journal*, vol. 14, no. 3, pp. 882–892, 2013.
[41] L.-C. Canon, M. El Sayah, and P.-C. Héam, "A comparison of random task graph generation methods for scheduling problems," in *European Conference on Parallel Processing*. Springer, 2019, pp. 61–73.
[42] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 127–138, 2012.

Yuvraj Sahni received B.E. (Hons) degree in Electrical and Electronics Engineering from Birla Institute of Technology and Science, Pilani, India in 2015. He is currently working towards the Ph.D. degree at Department of Computing, The Hong Kong Polytechnic University. Hong Kong. His research interests include wireless sensor networks, edge computing, and Internet of Things.



Jiannong Cao received the B.Sc. degree in computer science from Nanjing University, China, in 1982, and the M.Sc. and Ph.D. degrees in computer science from Washington State University, USA, in 1986 and 1990 respectively. He is currently the Otto Poon Charitable Foundation Professor in Data Science and the Chair Professor of Distributed and Mobile Computing in the Department of Computing at The Hong Kong Polytechnic University, Hong Kong. He is also the director of the Internet and Mobile Computing Lab in the department and the associate director of University Research Facility in Big Data Analytics. His research interests include parallel and distributed computing, wireless networks and mobile computing, big data and cloud computing, pervasive computing, and fault tolerant computing. He has co-authored 5 books in Mobile Computing and Wireless Sensor Networks, co-edited 9 books, and published over 600 papers in major international journals and conference proceedings. He is a fellow of IEEE, a distinguished member of ACM, a senior member of China Computer Federation (CCF).

Lei Yang received the BSc degree from Wuhan University, in 2007, the MSc degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2010, and the PhD degree from the Department of Computing, The Hong Kong Polytechnic University, in 2014. He is currently an associate professor at the School of Software Engineering, South China University of Technology, China. His research interest includes mobile cloud computing, Internet of things, and big data analytic. He has published more than 30 papers in conferences and journals. He is a program committee member for many international conferences. He is a member of the IEEE.

Yusheng Ji received the B.E., M.E., and D.E. degrees in electrical engineering from The University of Tokyo. She joined the National Center for Science Information Systems, Japan, in 1990. She is currently a Professor with the National Institute of Informatics and with The Graduate University for Advanced Studies. Her research interests include network architecture, resource management, and quality of service provisioning in wired and wireless communication networks. She is/has been an Editor of the IEEE Transactions on Vehicular Technology, a Symposium Co-Chair of the IEEE GLOBECOM 2012 and the IEEE GLOBECOM 2014, and a Track Co-Chair of the IEEE VTC 2016-Fall and IEEE VTC 2017-Fall.