Data-Aware Task Allocation for Achieving Low Latency in Collaborative Edge Computing

Yuvraj Sahni, Jiannong Cao, *Fellow, IEEE,* and Lei Yang

Abstract—The recent trend in the Internet of Things (IoT) is to distribute and move the computation from centralized cloud devices to edge devices which are closer to data sources. Researchers have proposed collaborative edge computing for IoT where the data and computation tasks are shared among a network of edge devices. One of the important problems in collaborative edge computing is to schedule tasks among edge devices to minimize latency and other performance metrics. Compared to existing works in wireless sensor networks and IoT, there are two additional challenges while scheduling tasks in collaborative edge computing. First, we need to consider the transfer of input data required by different tasks as the data is generated by sensing devices which are located at different geographical places. Second, existing works solve the problem of task scheduling without considering network flow scheduling which can lead to network congestion and long completion times. In this paper, we study the data-aware task allocation problem to jointly schedule task and network flows in collaborative edge computing. We mathematically model the joint problem to minimize the overall completion time of the application. We have proposed a multi-stage greedy adjustment (MSGA) algorithm where the task scheduling is done by considering both placement of tasks and adjustment of network flows. Performance comparison done using simulation shows that MSGA leads to up to 27% improvement in completion time as compared to benchmark solutions.

Index Terms—Task Scheduling, Network flow scheduling, Collaborative Edge Computing, Internet of Things.

1 Introduction

The recent trend in the Internet of Things (IoT) is to use Edge computing to distribute and move the computation from centralized cloud to edge devices which are closer to data sources. Edge computing is beneficial for Industrial IoT applications which require real-time processing such as determining whether a worker is not wearing protective equipment or wearing it incorrectly [1]. It is also useful for continuously monitoring the equipment and environment in shop floor and alerting the responsible personnel in case of concern [2] [3]. Edge computing can also help in improving the efficiency of operations in Industry. One such example is optimizing inventory management in Industrial IoT where the data collected from workers smartphone and wearables can be processed in real-time to dynamically track, and optimize various activities including storing, packing picking and transporting products. Such a solution can help reduce labour cost and improve efficiency in inventory management [1]. Another important application of edge computing is real-time video analytics [4]. Intelligent transportation systems and other Smart City applications require a large number of video feeds to be processed instantaneously. Edge computing helps in reducing the amount of data to be sent to the cloud by processing the video feeds close to the source. Analytics at the edge devices can help autonomous vehicles to make decisions, for example, detect hard-to-see pedestrians, based on the processed video feed from surrounding cameras [4].

These applications not only require pushing the computation down from the cloud to edge devices but also collaboration and exchange of data among different devices which has led to the emergence of collaborative edge computing. The work in [5] shows distributing tasks among collaborative edge devices gives better performance than offloading to a single edge server. Collaborative edge computing can be defined as distributing the computation tasks and data among a distributed network of edge devices, having computation, communication, and storage capability, that collaborate with each other. Many existing works such as [6], [7], [8], etc. have been proposed to enable sharing of computation tasks among devices. Based on the principle of collaborative edge computing, we have proposed a paradigm named Edge Mesh in [9] where decision-making is done inside the network by sharing data and computation tasks among edge devices instead of sending all the data to a centralized server. Edge Mesh helps in enabling distributed intelligence in IoT and provides many benefits including lower latency, higher reliability, etc.

A major challenge in Edge Mesh (and other systems based on collaborative edge computing) is to distribute tasks among edge devices. Unlike cloud computing paradigm where all the data is sent to a server, edge devices have direct access to only a subset of sensing devices which are connected to it. A task executed on an edge device can require data from different sensing devices. However, the sensing devices which are located at different geographical places will be connected to different edge devices. Therefore, data needs to be shared among edge devices, which are

- *This work was supported in part by the National Key R&D Program of China under Grant 2018YFB1004801, 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 152244/15E.*
- *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).*

connected using a multi-hop network, to enable decision making and execution of tasks. Task allocation decisions must be done by considering both the placement of input data and the network bandwidth consumed in transmitting the data.

In this paper, we study the data-aware task allocation problem to jointly schedule task and network flows in the collaborative edge computing with the objective of minimizing the completion time of the application. Scheduling the tasks without considering network bandwidth consumed by flows leads to network congestion and long completion times as shown by the motivational example in Section 2. The paper considers network congestion occurs when the bandwidth of all the flows passing through a link is greater than the link's capacity. The problem considers the placement of input data and the network bandwidth consumed in transferring data to schedule tasks. The data transfer results from the transfer of both the input data and the data exchanged between dependent tasks in an application. The problem is to decide when and where each task within an application is allocated and how network flows are scheduled such that there is no network congestion.

The problem is more challenging than existing works in wireless sensor networks (WSN) [10] [11] [12] and IoT [13] [14] as they usually assume that the data required for tasks is already available on devices and do not consider network congestion while scheduling tasks. There are some existing works in grids and data centers such as [15], [16], [17], [18], [19], [20], [21], etc. which have incorporated the transmission cost of input data. However, these works in grids and data centers do not consider the network congestion which cannot be ignored in case of collaborative edge computing. The work in [22] solves the problem of placement of tasks depending on the network scheduling policy. However, unlike data-aware task allocation, it does not jointly schedule tasks and network flows. The data-aware task allocation problem requires mathematical modelling of the joint task and network flow scheduling for collaborative edge computing where the edge devices are connected using a multi-hop path.

The main contributions of this work are:

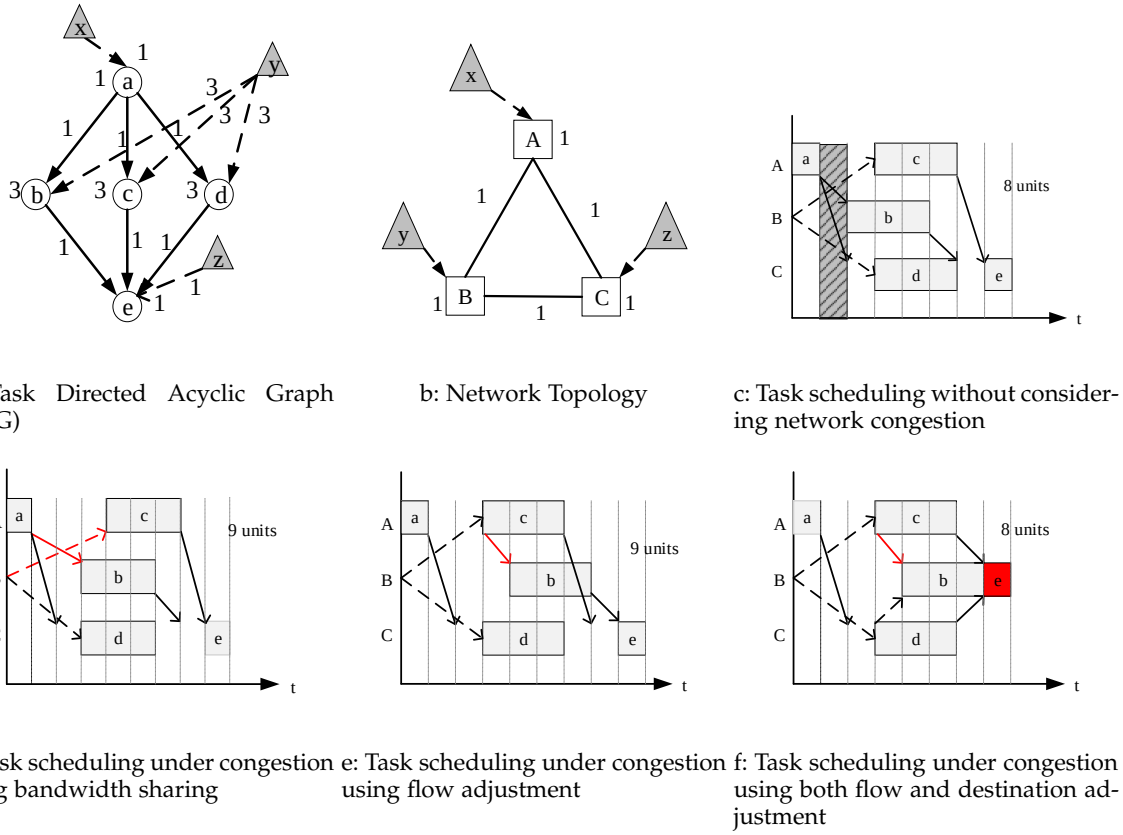
- We have mathematically formulated a data-aware task allocation problem considering the distribution of input data for different tasks and scheduling of network flows for collaborative edge computing. To the best of our knowledge, it the first work to jointly study task and network flow scheduling for collaborative edge computing, where the input data required for different tasks are distributed, and the objective is to minimize the completion time of the application.
- We have proposed a multi-stage greedy adjustment (MSGA) algorithm. It consists of three stages: creating an initial schedule without considering network congestion, detecting the network flow conflicts, and resolving the network flow conflicts by adjusting both the placement of tasks and the bandwidth of the flows. MSGA solves the issues associated with adjusting both the placement of tasks and bandwidth of flows.
- We have conducted simulation experiments to evaluate and compare the performance, in terms of completion time of application and running time, of MSGA against benchmark solutions. The benchmark solutions schedule flows based on either first-come-first-serve (FCFS) policy or the earliest finish time (EFT) priority. We have done performance comparisons by changing different parameters in the simulation including the number of tasks, the number of devices, number of input data sources, and the amount of the input data. The performance comparison shows that MSGA leads to up to 27% improvement in completion time as compared to benchmark solutions.

The rest of the paper is as follows. In Section 2, we have given a motivational example to illustrate the importance of data-aware task allocation problem. In Section 3, we give the system model and problem formulation. In Section 4, we discuss the proposed solution, MSGA, for the data-aware task allocation problem. In Section 5, we have done the performance evaluation. In Section 6, we discuss some related works. Finally, we give the conclusion in Section 7.

2 Motivational Example

Fig 1 shows the task allocation example to illustrate the importance of the joint task and network flow scheduling. The problem is to allocate a set of tasks within an application shown in Fig 1a to a set of devices shown in Fig 1b such that overall completion time is minimized. Fig 1a shows the task graph of the application where the circle nodes represent the tasks and triangle nodes represent the input data required by the task. The weight of circle nodes represents the computation load of the task, and the weight of link connecting circle nodes represents the dependency, i.e. if the two tasks are allocated at different devices then an amount of data equal to edge weight need to be transferred. The weight of edges connecting triangle nodes and circle nodes represents the amount of input data to be transferred to the task. Fig 1b shows a network of 3 devices where the weight on the square nodes (or devices) represents the processing power and the weight of the edge connecting different devices represents the bandwidth capacity of the link. The Fig 1b also includes triangle nodes connected to devices which represent the location of input data. Assuming that there is no network congestion and we can use the given bandwidth capacity of each link for data transfer, we can easily find the schedule shown in Fig 1c which minimizes the completion time. The completion time of the application using this schedule is 8 units.

The schedule shown in Fig 1c includes network flows represented by different lines. The task schedule is: task a is executed on the device A, task b on the device B, task c on the device A, task d on the device C, and task e on the device C. The figure includes a shaded area from time 1 to 2 units where two different network flows, a $\rightarrow$ b and y $\rightarrow$ c are passing through the same link AB which connects devices A and B. The flow from a to b is the result to transfer of data between dependent tasks whereas the flow from y to c is the result of transfer of input data y of task c. As mentioned before, the network congestion occurs when the bandwidth of all the flows passing though a link is greater than the link's capacity. We can resolve this network congestion by using bandwidth sharing policy where the overall bandwidth of the link is shared between different



a: Task Directed Acyclic Graph (DAG)

b: Network Topology

c: Task scheduling without considering network congestion

d: Task scheduling under congestion using bandwidth sharing

e: Task scheduling under congestion using flow adjustment

f: Task scheduling under congestion using both flow and destination adjustment

Fig. 1: Motivational example to show joint task and network flow scheduling

flows. The new schedule created using bandwidth sharing policy, shown in Fig 1d, results in the increased completion time of 9 units. The task schedule remains unchanged in the new schedule. The adjusted flows are marked in red in the figure. The bandwidth is shared for time 1 to 3 units for both the flows in Fig 1d.

We can create another schedule by changing the start time of each flow instead of sharing bandwidth as shown in Fig 1e. The completion time using flow adjustment is 9 units. The schedule shown in Fig 1e only adjusts the start time of flows without changing the destination. However, we can achieve even better completion time of 8 units, as shown in Fig 1f, by using both flow and destination adjustment. Task e is executed on the device B instead of the device C. Although in this example the completion time under network congestion is same as completion time without considering network congestion, it may not be the case in other situations. Fig 1d and 1e show the schedules where the network flows are separately considered from task placement which results in completion time of 9 units, whereas by jointly considering the network flows and tasks, as shown in Fig 1f, better results can be achieved. The rest of the paper will describe how to model the joint problem and propose the solution where both the placement of tasks and flow adjustment are considered to resolve network congestion.

3 System Model and Problem Formulation

Fig 2 shows the system architecture of Edge Mesh which is based on collaborative edge computing. It consists of three types of devices, i.e. end devices, edge devices, and cloud. End devices or sensing devices are responsible for sensing and actuation, edge devices for both decision-making and enabling interaction between devices, and finally the cloud which is responsible for performing big data analytics that cannot be done using edge devices. There are also some edge devices which do not have enough computation power for supporting decision-making tasks and only function as routers. The computation tasks in an application are distributed among edge devices. The system architecture also consists of SDN controller which is responsible for making all the decisions of allocating tasks to different edge devices and scheduling flows in the network. The other devices in the system architecture do not make any scheduling decisions. The SDN controller makes the decisions based on the network information collected from the edge devices and routers. The problem formulation does not include the time taken by the SDN controller to collect information and send control commands. Although SDN controller has been used before for making scheduling decisions in wired networks, the system model assumes wireless networking in Edge Mesh. There are few existing works which have used SDN controller for the wireless network. The work in [23] proposes meSDN that extends the control of SDN to mobile devices. Other works such as [24] and [25] used

SDN controller for making scheduling decisions in wireless networks. The work in [26] also implemented the prototype of the proposed algorithm in [25].

Wireless networking leads to further complications in making scheduling decisions due to its unique characteristics, including, spatial and temporal variations of wireless channel conditions and interference of wireless transmission among neighboring devices. However, the problem formulation has been done by assuming a static network condition where the dynamic wireless channel conditions and interference among the wireless transmissions have been ignored to make the problem simple. The computation tasks in an application are distributed among edge devices. As explained earlier, the tasks require input data from end devices which could be connected to different edge devices. The data-aware task allocation problem is to jointly schedule tasks and network flows in collaborative edge computing such that completion time of application is minimized. There are two other assumptions made in the problem, which are:

- The problem considers the shortest path to transfer the data. Other routing algorithms can also be selected.
- Each task starts execution after receiving all the data.

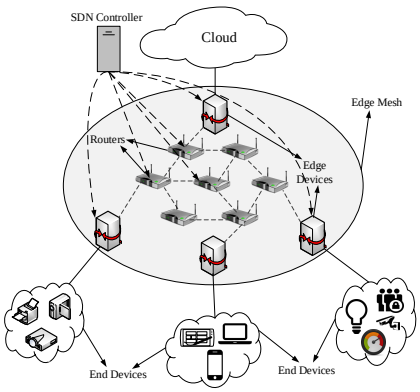


Fig. 2: System Architecture

TABLE 1: Notations used in the Paper

Variable	Meaning
G = (T, P)	task graph, where T is the set of tasks and P is set of dependencies among tasks
C = (A, E)	network model, where A is the set of devices and E is the set of edges
D	set of all input data
H = (V, Z)	dataflow model, where V is the set of dataflow tasks and Z is the set of edges
M	number of tasks
N	number of devices
K	number of dataflow tasks
Dt_i	set of input data for task i
dev_s	device where input data d_s is stored
$size_s$	amount of input data d_s in bits
c_i	computation load of processing task i
P_{ij}	length of data in bits transferred between tasks i and j
B_e	bandwidth capacity of link e
td_i	i-th dataflow task
cd_i	computation load of i-th dataflow task
$D_{i,j}$	length of data in bits transferred between dataflow task i and j
x_i	device where i-th task is executed
p_j	processing speed of device j
R_i	number of predecessor tasks for task i
Ts_{ij}	Starting time of dataflow task i executed on device j
Tf_{ij}	Finish time of dataflow task j executed on device j
$avail_j$	Earliest time when device j is free after executing all its previous tasks
Ts_i	Starting time of dataflow task i
Tf_i	Finish time of dataflow task i
$Tcomp_i$	Time to compute dataflow task i
$fl_{i,i'}$	network flow between dataflow task i and i'
$\mathbb{R}_{fl}(\tau)$	Rate of flow fl at time slot τ
E_{fl}	set of edges in the path of flow fl
f_n	current congested flow
F_cl	list of flows that conflict with the current congested flow
F_all	combined list of flows including f_n and F_cl
f_e	flow with the earliest start time in the list F_all
E_s	current execution schedule specifying when and where each task is scheduled
E_f	new schedule after flow adjustment
E_d	new schedule after destination adjustment
$dest_{f_e}$	list of all previous destinations for flow f_e
Δt_f	increase in completion time after flow adjustment
Δt_d	increase in completion time after destination adjustment

3.1 Problem Formulation

This section describes the modeling of application, network, data, and cost functions. The problem is formulated as a mixed-integer nonlinear programming problem. Table 1 summarizes the notations used in the paper.

Task graph Model: The application is modeled as a directed acyclic graph (DAG) G = (T, P), where T is the set of tasks, T = $\{i|1 \leq i \leq M\}$ and P is the set of dependencies between the tasks. The number of tasks is M. Each task i has a computation load of processing, c_i. The weight of each link, connecting tasks i and j, is P_{ij}, which represents the amount of data to be transmitted if the tasks are executed on different devices. Task i has some predecessor tasks which is given by a set R_i.

Network Model: The communication network is a mesh network of edge devices communicating with each other using a multi-hop path. The communication network is modeled as a graph C = (A, E), where A is the set of devices, A = $\{j|1 \leq j \leq N\}$, E= $\{e_{jj'}|j, j' \in A\}$ represents the link connecting device j and j'. The weight of each node j is p_j which represents the processing power of the device. The bandwidth of each link $e_{jj'}$ is $B_{e_{jj'}}$. In the problem description, we sometimes neglect the subscript and denote the link as e and the bandwidth as B_e. The number of devices in the communication network is N.

Data Model: Set of all input data D = $\{d_s|s \in 1, 2, ...S\}$. Set of data required for task i is represented by Dt_i which is a subset of D. Each data source d_s is located at device dev_s. The amount of data, in bits, for each data source d_s is $size_s$.

Dataflow Model: We model the application together with data model using a data flow graph, H = (V, Z), where V = $\{u|1 \leq i \leq K\}$ represents the set of dataflow tasks, and Z = $\{(u, v)|u, v \in V\}$ represents the sets of edges. Set V of dataflow tasks is equal to T ∪D, T is the set of tasks in the application model, and D is the set of input data in the data model. Each node u in set V represents a dataflow task and the weight of the node u, cd_u, which represents the computation load of the dataflow task u. The nodes corresponding to input data have zero computation load.

TABLE 2: Set of input data for each task

Task	Input Data
1	A (10), B(20)
2	C(20), D(10)
3	B(20), D (10)
4	E (20)

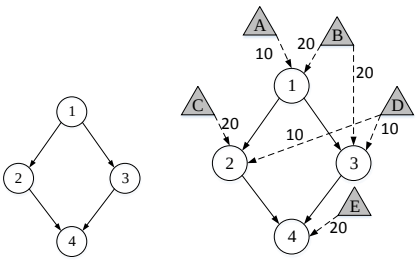


Fig. 3: (a) Example of task graph model, (b) Corresponding dataflow model

The total number of nodes in the dataflow graph is K. Z is the set of all links which includes links between different tasks in graph G and the links between input data and the corresponding task. The weight of the link connecting different node u and v in dataflow graph is $D_{u,v}$, which represents the amount of data to be transferred between two dataflow tasks u and v. The weight of links connecting nodes corresponding to input data and the task is equal to the amount of the input data required for the task. Fig 3 shows an example of task graph model and the corresponding data flow model. The input data required for different tasks in Fig 3a are shown in Table 2.

Cost Model: The completion time of an application is defined as the maximum time when the task belonging to the application is completed. The completion time is modeled using the EST (earliest start time) policy discussed in [27]. We need to calculate the time cost for processing and communicating data for each task to model the completion time. The computation cost of executing task i at device j is given by equation (1).

$$Tcomp_{i,j} = \frac{c_i}{p_j} \tag{1}$$

The start and finish time of a task i executed at device j, i.e. $Ts_{i,j}$ and $Tf_{i,j}$, is given by equation (2) and (3) respectively.

$$Ts_{i,j} = \max(avail_j, \max_{1 \le r \le R_i}(Tf_r + Ttask_{r,i})) \tag{2}$$

$$Tf_{i,j} = Ts_{i,j} + Tcomp_{i,j} \tag{3}$$

where $avail_j$ is the time when device j finishes executing any previously scheduled task, R_i is the set of the predecessor tasks of task i, $Ttask_{r,i}$ is the time taken to transfer data from predecessor task r to current task i. We can calculate the time cost for communicating data from predecessor task r to task i, both allocated on different devices, by dividing the amount of data transferred between tasks, $D_{r,i}$, with the rate of the network flow.

The problem is to determine when and where (on which device) each task should be executed, and schedule the flow of data transmission on the network links, such that the completion time of the application is minimized. The term task referred in problem description here is the dataflow task described above. Let Ts_i denotes the time that the task i starts to execute, x_i denotes the device where the task i is executed, where $1 \le i \le K$ and $1 \le x_i \le N$. The completion time of task i is Tf_i, where $Tf_i = Ts_i + Tcomp_i$. $Tcomp_i$ is the time to compute task i. We check every edge (i, i') in the dataflow graph, if the two connective tasks i and i' are not assigned to the same device, i.e., $x_i \neq x_{i'}$, then a flow $fl_{i,i'}$ is to be scheduled in the network. Each flow is associated with some properties such as when and where the flow has been generated, the destination of the flow, etc. These properties are defined as follows.

- Source: It is defined as the device from which the flow starts. The source of flow $fl_{i,i'}$ is x_i. If the flow is due to the transfer of input data, then the source is the device where input data is located.
- Destination: It is defined as the destination device where the flow ends. The destination of flow $fl_{i,i'}$ is x'_i. x'_i represents the device where the task i' is executed.
- Path: It is defined as a sequence of links through which the flow passes. We use a set $E_{fl_{i,i'}}$ of links in the network model to represent the path of flow $fl_{i,i'}$. In the following descriptions, we sometimes neglect the subscripts, and denote it by E_{fl}.
- Release time: The release time of flow $fl_{i,i'}$ is defined as time that the data is ready by the precedent task i, which is represented by $Ts_i + Tcomp_i$.
- Start time: The start time of the flow $fl_{i,i'}$ is defined as the time when the flow actually starts from the precedent task i. It is represented by $St_{fl_{i,i'}}$. In the following description, we sometimes neglect the subscript, and denote it by St_{fl}. The start time of the flow is greater than or equal to its release time.
- Deadline: It is defined as the latest time that the data transmission should be completed. The deadline of flow $f_{i,i'}$ is represented by $Ts_{i'}$, which is the time when the task i' needs to start.
- End time: It is defined as the time when the flow $fl_{i,i'}$ reaches its destination. It is represented by $Et_{fl_{i,i'}}$. In the following description, we sometimes neglect the subscript, and denote it by Et_{fl}. The end time of the flow is less than or equal to its deadline.
- Data amount: The data amount of flow $fl_{i,i'}$ is $D_{i,i'}$. $D_{i,i'}$ has been defined previously as the weight of the edge (i, i') in dataflow model.
- Rate: The rate of flow is defined as the data amount transmitted per time slot. The rate represents the network bandwidth allocated to the flow. The rate of flow is associated with time, i.e. it can vary with time. The rate of flow $fl_{i,i'}$ is represented by $\mathbb{R}_{fl_{i,i'}}(\tau)$ for time slot τ. We use the short form $\mathbb{R}_{fl}(\tau)$ to denote the rate of flow $fl_{i,i'}$. The rate of flow will be equal to zero for τ greater than end time of flow or less than start time of flow, i.e. $\mathbb{R}_{fl}(\tau) = 0$ when $\tau < St_{fl}$ or $\tau > Et_{fl}$. The

value of the rate of flow will be some non-zero value between the start and end time of the flow.

3.1.1 Data-Aware Task Allocation problem

The data-aware task allocation problem is formulated as an optimization problem described as follows.

Objective:

$$minimize \quad \max_{i \in V}\{Tf_i\} \tag{4}$$

Constraints:

$$\forall (i,i') \in Z, \quad Tf_i \leq Ts_{i'} \tag{5}$$

$$\forall (i,i') \in V, \quad |x_i - x_{i'}| * Q + (Ts_i - Tf_{i'}) * (Tfi - Ts_{i'}) \geq 0 \tag{6}$$

$$\forall (i,i') \in Z, St_{fl_{i,i'}} \geq Tf_i \tag{7}$$

$$\forall (i,i') \in Z, Et_{fl_{i,i'}} \leq Ts_{i'} \tag{8}$$

$$\forall (i,i') \in Z, \quad |x_i - x_{i'}| * (\sum_{\tau=St_{fl}}^{Et_{fl}} \mathbb{R}_{fl}(\tau) - D_{i,i'}) = 0 \tag{9}$$

$$\forall \tau \in [0, T-1], \forall e \in E, \quad \sum_{fl_{i,i'}} [\mathbb{R}_{fl}(\tau) * \mathbb{Y}(e, E_{fl})] \leq B_e \tag{10}$$

where Q in Equation (6) is a great positive constant which approaches to infinity, and $\mathbb{Y}$ is an function of e and E_{fl}, which represents whether edge e is part of set E_{fl}. If $e \in E_{fl}$, $\mathbb{Y}(e, E_{fl}) = 1$; otherwise $\mathbb{Y}(e, E_{fl}) = 0$.

Note that Equation (4) is the objective function to minimize the total completion time of the application. Equation (5) indicates that the dependent task can only start after its preceding task is completed. Equation (6) shows that one device can execute only one task at a time. In case multiple tasks are scheduled to the same device, the tasks are executed sequentially. This constraint comes from the assumption that we are considering devices with single core processor to make the problem simple. Although there are mobile devices with multicore processors, the underlying task scheduling problem will still remain the same. However, the constraint will have to be modified to consider the execution of multiple tasks (equal to the number of cores) simultaneously. A future work related to this problem can consider devices multi-core processors. Equation (7) defines that the flow can start only after its release time, which is the finish time of the preceding task. Equation (8) defines that the flow must finish before its deadline, which is the start time of the current task. Equation (9) represents that the rate of flow should be such that it is finished in time. Equation (10) represents that the amount of bandwidth consumed by each link at a given time must be less than the given bandwidth capacity of the link.

The data-aware task allocation problem is NP-hard as it is an extension of task scheduling problem that jointly considers task and network flow scheduling. The decision problem of the task scheduling has been proven to be NP-complete [27].

4 Multi-Stage Greedy Adjustment Algorithm

We have proposed a multi-stage greedy adjustment (MSGA) algorithm for the data-aware task allocation problem. The flowchart of MSGA is shown in Fig 4. The advantage of MSGA is that we can substitute part of the algorithm to create different solutions. For example, instead of using a greedy algorithm for creating an initial schedule we can utilize evolutionary algorithm such as the genetic algorithm. We can also use different policies for resolving the network flow conflicts. The steps in the MSGA are shown in Algorithm 1. The input for the Algorithm 1 includes the task graph of M tasks, the network of N devices, and the set of input data Dt_i for each task i in the task graph.

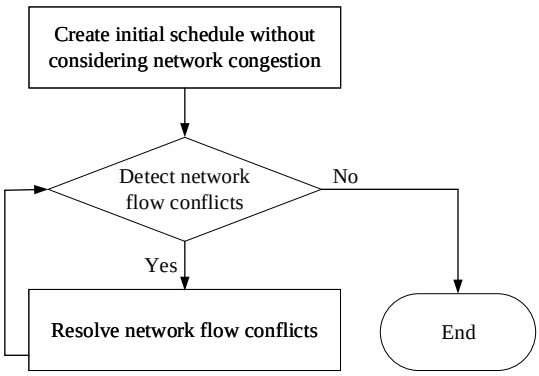


Fig. 4: Flowchart of MSGA

The first stage is creating an initial schedule using the greedy method based on list scheduling without considering the network congestion (**Algorithm1, Line 1**). The method is similar to HEFT algorithm proposed in [27] except we need to consider the time taken to transfer input data additionally. We first create a priority list of tasks where the priority of the task is calculated based on the node and edge weight which represent the computation and communication load of the task respectively. The task with the longest path to the end node of the task graph is given the highest priority. Starting with the highest priority task, we assign the task to the device which finishes it at the earliest time. The start and finish time of a task i executed at device j, i.e. $Ts_{i,j}$ and $Tf_{i,j}$, is given by equations (11) and (12) respectively. Unlike the equations (2) and (3), we consider the time when input data is available at the device separately from the time taken to transfer data between dependent tasks. We use the dataflow model for calculation related to network flows and task graph model for calculation related to task scheduling. This enables us to consider the computation tasks separately and makes it easier to create an initial schedule.

$$Ts_{i,j} = \max(avail_j, \max_{1 \leq r \leq R_i}(Tf_r + Ttask_{r,i}), \max(Tdata_i)) \tag{11}$$

$$Tf_{i,j} = Ts_{i,j} + Tcomp_{i,j} \tag{12}$$

where $avail_j$ is the time when device j finishes executing any previously scheduled task, R_i is the set of the predeces-

sor tasks of task i, $Ttask_{r,i}$ is the time taken to transfer data from predecessor tasks, $Tdata_i$ is the time taken to transfer input data, and $Tcomp_{i,j}$ is the computation time to execute task i at device j.

The finish time of a task is dependent on the processing time of the application on the allocated device, time to transfer the data from preceding tasks, and time to transfer the input data. There are two types of flows in the network. The first type of flow corresponds to transfer of data from preceding tasks which can only start after the completion of preceding task. The second type of flow corresponds to transfer of input data to the device where task is allocated. Ideally, second flow can start at time 0, but due to the constant bandwidth available on each link, it usually leads to network congestion if all flows are started at same time. The time to transfer data flow $f_{i,i'}$ is calculated using equation (13). We have ignored any network congestion, at this stage, so we use the entire bandwidth of the link even if there are multiple network flows passing through the link at the same time.

$$T_{f_{i,i'}} = \frac{D_{i,i'}}{\min_{e \in E_{fl}} B_e} \tag{13}$$

where $D_{i,i'}$ denotes the amount of data to be transferred in the flow $f_{i,i'}$, E_{fl} denotes the set of the edges in the path of flow $fl_{i,i'}$, B_e is the bandwidth of the edge e.

The initial schedule includes information about network flows, including, start time, finish time, rate, amount of data, source and destination of flows. This information is recorded to be utilized later for detecting and resolving network conflicts (**Algorithm 1, Line 2**). We sort the flows in increasing order of their start time and detect the earliest network conflict (**Algorithm 1, Line 3**). A conflict is defined when the bandwidth utilized by all the flows passing through a link at any given time is more than the given bandwidth of the link. Once the conflict has been detected, we move on to the third stage of the algorithm to resolve conflict (**Algorithm1, Line 4 -15**). The second and third stage are repeated until there is no more network conflict.

Resolving network flow conflict: The first step in resolving network conflict is finding a list of other flows F_{cl} which are in conflict with the current congested flow f_n (**Algorithm1, Line 4**). We first find the list of all other flows, F_{sl} which share the same time duration as the current flow f_n. Then, based on the definition of the conflict mentioned above, we find flows in the list F_{cl}. The schedule before adjustment is recorded as E_s. We utilize two different methods to resolve network flow conflicts. First is flow adjustment where we change the start time of the flows and second is destination adjustment where we change the destination of the flows. We have sometimes used the term bandwidth adjustment instead flow adjustment in the paper. Flow adjustment method is based on first-come-first-serve (FCFS) strategy where the flow that starts first is given priority over other flow. The congested flow f_n and flows in list F_{cl} are sorted together, combined list F_{all}, in increasing order of start time and the flow with the earliest start time, f_e is selected for adjustment (**Algorithm 1, Line 5-6**). All the other flows, except f_e, in the combined list F_{all} are delayed until the flow f_e is completed (**Algorithm 1, Line 7**). Based on the flow adjustment, we update the schedule to E_f by modifying when the tasks are scheduled (**Algorithm 1, Line 8**). All the devices are still scheduled at the same device as in the initial schedule E_s. The consideration of input data transfer in the problem requires us to consider the complete task nodes while updating the schedule, whereas for a traditional task allocation the schedule could have been updated by only considering the successive task nodes. After updating the schedule, the difference in overall completion time, Δt_f between the new updated schedule, E_f, an initial schedule, E_s, is calculated (**Algorithm 1, Line 9**). The second method of destination adjustment is applied to the flow f_e selected in Line 6 of Algorithm 1 (**Algorithm 1, Line 10 - 12**). The detailed steps involved in destination adjustment method are specified in Algorithm 2. The method which leads to the minimum increase in overall completion time is selected as the new existing schedule (**Algorithm 1, Line 13 - 16**). This process of detecting and resolving conflicts continues until there is no network conflict in the final schedule.

Destination Adjustment: The basic idea of destination adjustment method is to change the destination of flow to another destination which helps in removing the network conflict. In the destination adjustment method, we first assign the network bandwidth to the all the other flows in list F_{all} except f_e (**Algorithm 2, Line 2**). We then utilize the remaining bandwidth to assign a new destination for the flow which finishes the destination task of the flow in earliest time (**Algorithm 8, Line 3 - 8**). There are two main issues with changing the destination of the flow:

1. The schedule has a "ping-pong effect" where the destination of the flow fluctuates between two devices in later iterations.

2. Changing the destination of a flow can result in new network conflicts in previous flows which could prevent resolving the current congested flow f_e.

These issues take place because, unlike traditional task allocation problem, the current problem also involves input data transfer. Therefore, when the destination device is changed for a flow it affects not only successive flows but also preceding flows as the input data flows can start from time t = 0. The destination adjustment algorithm solves the issues by using two techniques. The first issue is resolved by maintaining a list of destination, $dest_{f_e}$, for each flow f_e. If after the destination adjustment the new destination is part of the list $dest_{f_e}$ for the flow f_e, then we set the increase in overall completion time after destination adjustment, Δt_d, equal to infinity (**Algorithm 2, 11 - 14**). This ensures that destination selected in the previous iteration is not selected again, thus, preventing "ping-pong effect". The second issue is resolved by checking whether the flow f_e is conflicted or not after destination adjustment. If the flow is still conflicted, we set the increase in overall completion time after destination adjustment, Δt_d, equal to infinity (**Algorithm 2, 11 - 14**). This ensures that destination adjustment is only considered if the adjustment resolves network conflict for the flow f_e.

The enhancements together prevent the Algorithm 1 to select the destination adjustment strategy instead of flow adjustment in case of an issue (**Algorithm 1, 13 - 16**). The algorithm does not consider the current destination of the flow for adjustment (**Algorithm 2, Line 3-4**), which helps in

removing redundancy as the algorithm would have rejected the destination as it is part of the list $dest_{f_e}$. Similar to flow adjustment method, this method also updates the schedule (**Algorithm 2, Line 10**). The destination of the flow is changed before updating the schedule so we do not need to change the destination while updating schedule, similar to flow adjustment method. After the adjustment, we add the new destination of the flow, irrespective of whether the previous conditions are satisfied or not, to the list $dest_{f_e}$ (**Algorithm 2, Line 15**).

Algorithm 1: Multi-stage greedy adjustment (MSGA) algorithm

Input: The task graph of M tasks, the network of N edge devices, and the set of input data Dt_i for each task i in the task graph
Output: The execution schedule specifying when and where each task is scheduled, i.e. the start time Ts_i and the device x_i for each task i

1 Compute an initial schedule using a greedy method based on list scheduling;
2 Record the execution time of each task and flow associated with it including the start time, finish time, data, rate, source, and destination of flow ;
3 **while** *detect a conflict point* **do**
4 Find list of flows F_{cl} ;
5 Create a combined list F_{all} consisting of f_n and F_{cl};
6 Sort the list in increasing order of start time and select the flow f_e with the earliest start time;
7 Delay transmission of all flows in list f_{cl} until the flow f_e is completed;
8 Create a new schedule, E_f ;
9 Calculate Δt_f ;
10 Change the destination of the flow f_e in the schedule E_s to a new destination that can finish the dependent task in the earliest time ;
11 Create a new schedule E_d ;
12 Calculate Δt_d ;
13 **if** $\Delta t_d \geq \Delta t_f$ **then**
14 $E_s = E_f$, i.e. Set the new schedule as E_f;
15 **else**
16 $E_s = E_d$, i.e. Set the new schedule as E_s;
17 **end**
18 **return** E_s;

4.1 Complexity Analysis

The computation complexity of MSGA is O($|Z|$*(N + M)), where N is the number of devices in the network, $|Z|$ is the number of edges in the dataflow model, and M is the number of the tasks in the task graph. The computation complexity is calculated by considering the most complex operation in Algorithm 1 which is the destination adjustment part shown in Algorithm 2. For each conflict, the destination adjustment can choose among any of the N devices as the new destination. Once the new destination is selected, we calculate the modified schedule by considering all M tasks in the task graph model. This destination adjustment is done for all conflicts which is equal to the total number of edges in the dataflow model, i.e. $|Z|$. Hence, the complexity of the Algorithm 1 is O($|Z|$*(N + M)).

Algorithm 2: Destination adjustment algorithm

Input: The task graph of M tasks, the network of N edge devices, and the set of input data Dt_i for each task t_i in the task graph, congested flow f_e, list of flows F_{cl} congested at same time, list of destination, $dest_{f_e}$, for flow f_e
Output: The updated schedule E_d after adjustment and the increase in completion time Δt_d

1 Find the destination task, t_i for the congested flow f_e ;
2 Calculate the remaining bandwidth after scheduling other flows in the list F_{cl} using the initial bandwidth
for $i \leq N$ **do**
3 **if** $i == dev_i$ **then**
4 $t_{i,f} = \infty$;
5 **else**
6 Calculate the finish time of task t_i considering remaining bandwidth
7 **end**
8 Select the device which finishes the task t_i in earliest time;
9 Update the information of flows directly affected by destination adjustment;
10 Create a new schedule E_d ;
11 **if** f_e *is no more congested* **and** *new destination of* f_e *is not part of* $dest_{f_e}$ **then**
12 $\Delta t_d = Newcompletiontime - Oldcompletiontime$;
13 **else**
14 $\Delta t_d = Inf$;
15 Add the new destination of f_n into the list f_e;
16 **return** $E_d, \Delta t_d$;

5 Evaluation

We have done the simulation using MATLAB to evaluate and compare the performance of the MSGA with benchmark solutions. The performance evaluation has been done using two performance metrics: completion time of the application and running time of the algorithm. The parameters used for the simulation are similar to the one used previously in [12] and [28].

5.1 Simulation Setting

Parameters for Network Model: We generate a network of edge devices where devices are randomly deployed in a 100m x 100m area, and any two devices less than 35m apart are connected to each other. The size of the area is similar to the one used in [29]. Other works related to wireless sensor networks have also used the network area of the same size. It is All the devices are connected to each other using a multi-hop path. The weight of the device represents the processing power, and weight of the link connecting two devices represents the bandwidth capacity of the link. The devices are heterogeneous in terms of processing power which is selected to be [50MCPS $\pm$ 10%]. The bandwidth of each link is selected to be [250Kbps $\pm$ 10%].

Parameters for Application Model: We have implemented a random DAG generator for the task graph using the level by level method mentioned in [30]. The task graph contains M nodes where each node represents the task and weight of the node represents the computation load of the task. The nodes are connected using edges whose weight represents the amount of data to be transferred between dependent tasks. The number of levels in the task graph is selected to be a normal distribution in the range [M/4, M/2]. Each level contains at least one node and the number of nodes in each level is selected randomly. The computation load of each task is selected to be [300KCC $\pm$ 10%] and data transferred between two dependent tasks is selected to be [750 bits $\pm$ 10%]. The amount of data to be transferred is calculated based on the communication-to-computation ratio (CCR) of 0.5.

Parameters for Data Model: We generate a set of 20 input data which are selected to be located on random devices. We randomly select a subset of input data (2 for default case) for each task. The amount of input data is selected to be [3200 bits $\pm$ 10%].

5.1.1 Benchmark Solutions

We have compared the performance of MSGA with two benchmark solutions. The benchmark solutions also follow the three-stage methodology of MSGA. The first and second stage of the benchmark solutions are same as that of MSGA, however, in the third stage, we use a different method for resolving conflicts. The first benchmark uses the first-come-first-serve (FCFS) policy to adjust the flows. The flow which starts at the earliest time is given highest priority while adjusting the flows in the first benchmark solution. Other congested flows are delayed until the highest priority flow is finished. The second benchmark uses another priority method where the flow which finishes first is given highest priority. Concerning Algorithm 1, sorting is done in increasing order of finish time of flows instead of the start time in Line 6. This implies that all other conflicted flows are delayed until the flow the highest priority (earliest finish time) is finished. Compared to MSGA, both of these benchmark solutions resolve the network conflict by adjusting the flows only whereas MSGA also utilizes changing the destination of flows, i.e. changing the placement of tasks.

We have also implemented a genetic algorithm (GA) where the network congestion is resolved using the FCFS policy. Compared to other benchmarks, the genetic algorithm improves the solution iteratively by changing the placement of tasks. The genetic algorithm implemented in this paper is similar to the one in [9], however, we have changed the encoding of genes, crossover operator, and mutation operator. We encode each gene as a vector of integers representing the allocated device for each task. A simple crossover operator has been used where two genes exchange the second partition of a gene selected randomly. We use power mutation described in [31]. The parameters used for GA are: number of chromosomes in the initial population is 10, the number of generations is 100, 80% of the original population is selected for crossover, and mutation ratio is 0.02.

TABLE 3: Default parameters used for simulation

Parameter	Value
Number of tasks	30
Number of devices	100
Number of input data sources	2
Amount of input data	3200 bits $\pm$ 10%
Computation load of each task	300 KCC $\pm$ 10%
Processing power of each device	50 MCPS $\pm$ 10%
Data transmission between dependent tasks	750 bits $\pm$ 10%
Bandwidth of each link	250 Kbps $\pm$ 10%

5.2 Simulation Results

The default parameters used for simulation are shown in Table 3. Table 4 shows the comparison between benchmark solutions and MSGA in terms of both completion time and running time of the algorithms. We have used short form FCFS (first-come-first-serve) to represent the first benchmark solution, EFT (earliest finish time) to represent the second benchmark solution, and GA to represent the genetic algorithm. Compared to other algorithms, MSGA is able to achieve better performance in terms of completion time. However, the better performance comes at the cost of higher running time of MSGA compared to the benchmark solutions, FCFS and EFT. It can be observed that genetic algorithm is also able to achieve similar performance as FCFS and EFT in terms of completion time but the running time of GA is very high. It is almost 830 times higher than other benchmark solutions. Due to such high running time, we have not considered the genetic algorithm for performance comparison while changing other parameters. The results obtained for performance comparison have been averaged out for 30 iterations. Each iteration is different in terms of both the task graph and the wireless network generated randomly. The completion time and running time shown in Table 4 are calculated using 95 % confidence interval except for GA since it is not considered for performance comparison later. The result can be interpreted as, for example, the average completion time using MSGA is between 0.1523 and 0.1847 with 95% confidence. The error margin is around 10% for 95% confidence interval if the results are averaged out for 30 iterations.

5.2.1 Effect of changing number of tasks

We evaluate the effect on the performance of algorithms by changing the number of tasks from 10 to 50 while keeping other parameters constant. Fig 5 shows the performance comparison in terms of completion time where the completion time increases on increasing the number of tasks. The performance difference, in terms of completion time, between MSGA and benchmark solutions ranges from 16% to 22%. The performance difference decreases from 22% to 15.8% as the number of tasks is increased from 30 to 50 tasks. However, when the number of tasks is 10 or 20, the performance difference is less around 16% because the number of devices is 100 which is large enough for benchmark solutions to give good results.

We have also compared the running time of MSGA with benchmark solutions as shown in Fig 6. As the number

TABLE 4: Performance Comparison for default parameters

Metric	FCFS	EFT	GA	MSGA
Completion time (sec)	0.2158 ± 0.0236	0.2454 ± 0.0287	0.2175	0.1685 ± 0.0162
Running time (sec)	33.3678 ± 1.3327	35.0966 ± 1.4268	27676	100.7590 ± 7.4757

of tasks is increased, the running time also increases as both more tasks and flows need to be scheduled. However, the increase in running time of MSGA is higher compared to other benchmark solutions, FCFS and SJF, as MSGA includes destination adjustment algorithm too where a new modified schedule is calculated by considering all M tasks in the task graph model. However, for benchmarks solutions, the new schedule can be easily calculated by shifting the start time of other flows. The running time of MSGA is around 2-3 times more than that of benchmark solutions. However, compared to the genetic algorithm the running time of MSGA is still very low, and MSGA also gives better result in terms of completion time. This shows a trade-off between the two performance metrics, i.e. completion time and running time.

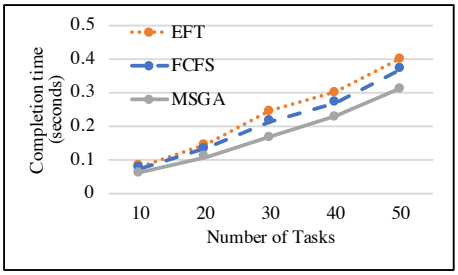


Fig. 5: Effect of number of tasks on completion time

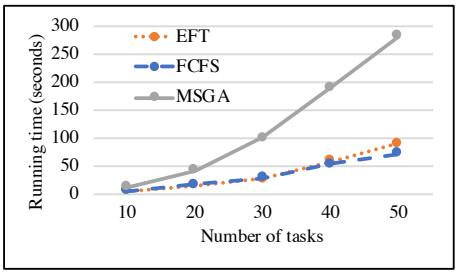


Fig. 6: Effect of number of tasks on running time

5.2.2 Effect of changing number of devices

Fig 7 shows the performance comparison, in terms of completion time, by changing the number of devices from 50 to 250. Performance comparison shows that the completion time decreases as the number of devices is increased because the number of connections is increased due to increase in density of devices. The increase in connections leads to less completion time as the transfer time is now decreased. The performance difference between MSGA and benchmark solution increases as the number of devices are increased. The reason is that as the number of devices is increased, MSGA has more chances of changing the destination of flows. However, this trend continues only up to a certain point. If the number of devices is high, then the difference between MSGA and benchmark solutions will start to decrease as the benchmark solutions can give good results if the number of devices is high. This can be observed in Fig 7 where the difference in performance first increases from 12.4% at 50 devices to 27.2% at 150 devices and then decreases to 21.3% when the number of devices is 250.

The running time of MSGA will increase more than that of benchmark solutions as shown in Fig 8. This increase happens because MSGA utilizes changing the destination of flow to other devices which is dependent on the number of devices in the network.

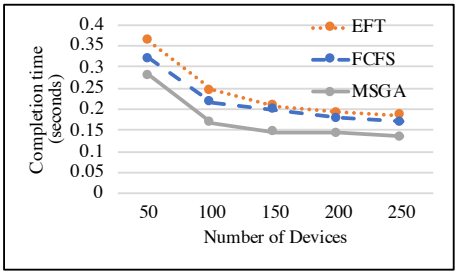


Fig. 7: Effect of number of devices on completion time

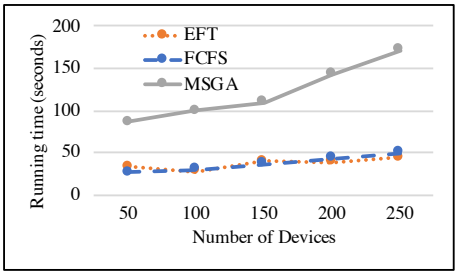


Fig. 8: Effect of number of devices on running time

5.2.3 Effect of changing number of input data sources

We have evaluated the effect of changing the number of input data sources from 1 to 4 on completion time as shown in Fig 9. The results show that as the number of input data sources is increased the value of completion time increases. This increase in completion time is due to increase in transfer time as network congestion increases on increasing the number of input data sources. The performance difference between MSGA and benchmark solutions also decreases from 21% to around 10% on increasing the number of input data sources. The decrease in performance is the result of increased network congestion which makes it difficult to change the destination of flow. Since the network congestion cannot be resolved efficiency due to increase in number parallel data transmissions, there is not a significant difference between MSGA and benchmark solutions.

The running time of both MSGA and benchmark solutions increases as the number of input sources are increased as shown in Fig 10. Due to increased network congestion for large values of input data sources, the increase in running time of MSGA is more than that of benchmark solutions. The reason is the network congestion increases, and more time is spent by destination adjustment part of MSGA which leads to a higher increase in running time.

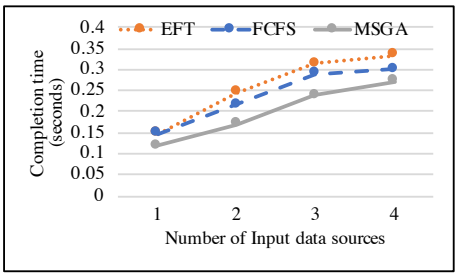


Fig. 9: Effect of number of input data sources on completion time

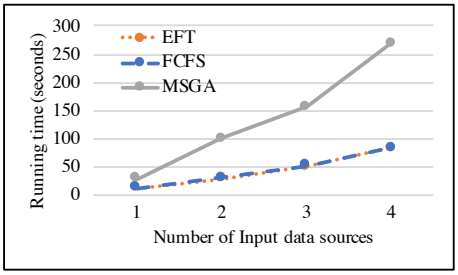


Fig. 10: Effect fo number of input data sources on running time

5.2.4 *Effect of changing amount of input data*

Fig 11 shows the effect of increasing the amount of average input data from 1600 bits to 6400 bits. Fig 11 shows that as the amount of average input data is increased the value of completion time increases. The increase in completion time is due to increase in the amount of transfer time for input data has increased. An interesting observation is that as the amount of average input data is increased, the performance difference between MSGA and benchmark solutions also increases from 7.36% for 1600 bits to 25.75% for 6400 bits. The reason for the observed increase in performance difference is that benchmark solutions delay the flows based on priority which results in a significant rise in completion time when the amount of input data is substantial, whereas MSGA changes the destination to resolve network conflict which does not require delaying other conflicted flows.

The running time remains almost same as the amount of average input data is increased as shown in Fig 12. The reason is that as that there is no significant difference in network congestion by increasing the amount of input data.

6 Related Work

In this section, we introduce related works in task allocation belong to different categories, including, wireless sensor networks and data centers.

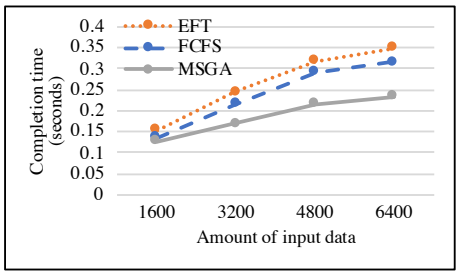


Fig. 11: Effect of amount of input data on completion time

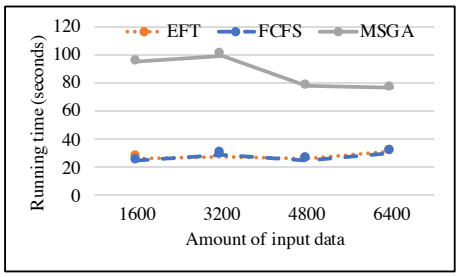


Fig. 12: Effect of amount of input data on running time

Existing works related to tasks allocation in WSN have considered both single hop [10] and multi-hop [11], [12] WSN. The work in [12] adjusts the schedule length to resolve any network conflicts, however, compared to our work, it does not consider the input data of tasks. Multi-objective task allocation problem has also been studied in [28]. There are few works that also consider task allocation problem in IoT such as [13] and [14]. In [13], authors study a task mapping problem which considers features specific to IoT including embedded system constraints such as minimising energy consumption and resource constraints, shared sensing, and continuous processing for large scale mobile networks. Authors in [14] propose a distributed consensus-based algorithm for task allocation in IoT. These works do not consider transmission cost for input data to device where task is allocated. The transmission cost has been considered for task allocation in IoT in our previous work [9]. However, unlike existing works, this work jointly schedules tasks and network flows for collaborative edge computing. However, the work in [9] only considers single hop communication networks and the objective is to minimize only the total energy consumption. In this work, we minimize completion time of the application for collaborative edge computing where the devices are connected jointly schedule task and network flows which has not been done in other existing works.

There are also some existing works that consider data distribution on task allocation in grids and data centres [15], [16], [19], [17], [18], [20], [21] etc. A related work has been done for dependent tasks in [32]. Authors in [32] formulate a integer linear programming problem to jointly solve the heterogeneous data allocation and task scheduling (HDATS) problem of assigning processors to real-time tasks and allocate data. The work in [22] solves the problem of placement of tasks depending on the network scheduling policy. However, these works in grids and data centers do not jointly schedule task and network flows as done in this

work.

There are some related work which have considered network condition while scheduling tasks [33] [34].The work in [33] proposes a system named Iridium which optimizes the placement of both data and tasks while considering WAN bandwidth. This work avoids network congestion by avoiding sending too much data over a narrow link. Another related work in SquirrelJoin [34], which is a distributed join processing technique that uses lazy partitioning to adapt to network skew conditions. SquirrelJoin specifically considers receiver-side skew where large amount of data is assigned to faster receivers so that receivers affected by receiver-side skew have to process less data. These works are different from the data-aware task allocation problem studied in this paper. These works consider MapReduce jobs or join queries whereas, our work considers generic task DAG model. The placement of the input data of different tasks in the DAG needs to be considered while making task allocation decision.

7 Conclusion

This paper studies the data-aware task allocation where task and network flows are jointly scheduled with the objective of minimizing the completion time of application. The problem can be easily extended further to multi-objective problem such as minimizing the completion time and total energy consumption of all tasks. This problem is useful for systems based on collaborative edge computing in IoT such as Edge Mesh, where the data and computation tasks is distributed among edge devices. The problem considers both the placement of data and network bandwidth consumed in transferring data to schedule tasks. We have proposed a solution, MSGA, for the data-aware task allocation. The three stages in MSGA are: creating an initial schedule without considering network congestion, detecting network flow conflicts, and resolving network flow conflicts. We adjust both the destination of flow and bandwidth to resolve the network flow conflicts. The advantage of proposing three-stage algorithm is that we can easily modify the algorithm, for example instead of using a greedy algorithm based on list scheduling for creating initial schedule, we can use another algorithm. We have done simulation experiments to evaluate and compare the performance of MSGA with the benchmark solutions which only consider adjusting the bandwidth of flows to resolve network flow conflicts. Performance comparison shows that the proposed solutions can lead to up to 27% improvement in completion time compared to benchmark solutions. Although the proposed solution requires more running time than benchmark solutions as it also considers adjusting destination of flows, the running time is far less compared to genetic algorithm.

References

[1] D. Georgakopoulos, P. P. Jayaraman, M. Fazia, M. Villari, and R. Ranjan, "Internet of things and edge cloud computing roadmap for manufacturing," *IEEE Cloud Computing*, vol. 3, no. 4, pp. 66–73, 2016.

[2] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, 2017.

[3] S. Verma, Y. Kawamoto, Z. M. Fadlullah, H. Nishiyama, and N. Kato, "A survey on network methodologies for real-time analytics of massive iot data and open research issues," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1457–1477, 2017.

[4] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[5] 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.

[6] F. Van den Abeele, J. Hoebeke, G. K. Teklemariam, I. Moerman, and P. Demeester, "Sensor function virtualization to support distributed intelligence in the internet of things," *Wireless Personal Communications*, vol. 81, no. 4, pp. 1415–1436, 2015.

[7] I. Stojmenovic, "Machine-to-machine communications with in-network data aggregation, processing, and actuation for large-scale cyber-physical systems," *IEEE Internet of Things Journal*, vol. 1, no. 2, pp. 122–128, 2014.

[8] 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.

[9] Y. Sahni, J. Cao, S. Zhang, and L. Yang, "Edge mesh: A new paradigm to enable distributed intelligence in internet of things," *IEEE Access*, 2017.

[10] Y. Yu and V. K. Prasanna, "Energy-balanced task allocation for collaborative processing in wireless sensor networks," *Mobile Networks and Applications*, vol. 10, no. 1-2, pp. 115–131, 2005.

[11] Y. Tian and E. Ekici, "Cross-layer collaborative in-network processing in multihop wireless sensor networks," *IEEE transactions on mobile computing*, vol. 6, no. 3, 2007.

[12] Y. Jin, J. Jin, A. Gluhak, K. Moessner, and M. Palaniswami, "An intelligent task allocation scheme for multihop wireless networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 444–451, 2012.

[13] B. Billet and V. Issarny, "From task graphs to concrete actions: a new task mapping algorithm for the future internet of things," in *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*. IEEE, 2014, pp. 470–478.

[14] G. Colistra, V. Pilloni, and L. Atzori, "The problem of task allocation in the internet of things and the consensus-based approach," *Computer Networks*, vol. 73, pp. 98–111, 2014.

[15] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*. IEEE, 2002, pp. 352–358.

[16] J. Taheri, A. Y. Zomaya, and S. U. Khan, "Genetic algorithm in finding pareto frontier of optimizing data transfer versus job execution in grids," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 6, pp. 1715–1736, 2016.

[17] S. Kumar and N. Kumar, "Network and data location aware job scheduling in grid: improvement to gridway meta scheduler," *International Journal of Grid and Distributed Computing*, vol. 5, no. 1, pp. 87–100, 2012.

[18] C. Augonnet, J. Clet-Ortega, S. Thibault, and R. Namyst, "Data-aware task scheduling on multi-accelerator based platforms," in *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*. IEEE, 2010, pp. 291–298.

[19] R. McClatchey, A. Anjum, H. Stockinger, A. Ali, I. Willers, and M. Thomas, "Data intensive and network aware (diana) grid scheduling," *Journal of Grid computing*, vol. 5, no. 1, pp. 43–64, 2007.

[20] P. Zhang, Y. Gao, and M. Qiu, "A data-oriented method for scheduling dependent tasks on high-density multi-gpu systems," in *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*. IEEE, 2015, pp. 694–699.

[21] M. Szmajduch and J. Kolodziej, "Data-aware scheduling in massive heterogeneous systems." in *ECMS*, 2015, pp. 601–607.

[22] 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.

[23] 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*. ACM, 2014, pp. 7–14.

[24] 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, 2018.

[25] T. De Schepper, S. Latré, and J. Famaey, "A transparent load balancing algorithm for heterogeneous local area networks," in *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*. IEEE, 2017, pp. 160–168.

[26] 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 *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*. IEEE, 2017, pp. 901–902.

[27] 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.

[28] 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, 2014.

[29] Y. Jin, S. Vural, A. Gluhak, and K. Moessner, "Dynamic task allocation in multi-hop multimedia wireless sensor networks with low mobility," *Sensors*, vol. 13, no. 10, pp. 13 998–14 028, 2013.

[30] A. Olteanu and A. Marin, "Generation and evaluation of scheduling dags: How to provide similar evaluation conditions," *Computer Science Master Research*, vol. 1, no. 1, pp. 57–66, 2011.

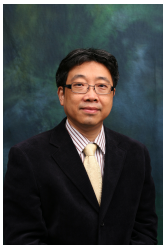
[31] K. Deep, K. P. Singh, M. L. Kansal, and C. Mohan, "A real coded genetic algorithm for solving integer and mixed integer optimization problems," *Applied Mathematics and Computation*, vol. 212, no. 2, pp. 505–518, 2009.

[32] Y. Wang, K. Li, H. Chen, L. He, and K. Li, "Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints," *IEEE transactions on emerging topics in computing*, vol. 2, no. 2, pp. 134–148, 2014.

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

[34] 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.

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 a Chair Professor of 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 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.