TASK PARTITIONING AND OFFLOADING IN COLLABORATIVE EDGE COMPUTING ENVIRONMENTS

YUVRAJ SAHNI

PhD

The Hong Kong Polytechnic University

2021

The Hong Kong Polytechnic University

Department of Computing

Task Partitioning and Offloading in Collaborative Edge Computing Environments

Yuvraj Sahni

A thesis submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

August 2020

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

___________________(Signed)

_____Yuvraj Sahni_____(Name of student)

Abstract

In the past decade, edge computing has become popular as it pushes the computation and data storage closer to data sources to address issues with cloud computing such as privacy, network congestion, latency, etc. Collaborative edge computing (CEC) is a new paradigm of edge computing, where 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. One of the fundamental issues in CEC is partitioning and offloading the tasks among heterogeneous edge devices. However, they are difficult problems to solve due to two unique features of CEC: 1) the data required for a task are from multiple edge devices, and 2) tasks can be offloaded to an edge device at a multi-hop distance due to heterogeneity among edge device resources. The transfer of data to an edge device at a multi-hop distance leads to contention among network flows. This makes it difficult to estimate the communication cost of transmitting data as the network link could be occupied by another network flow.

This thesis studies the task partitioning and offloading problems in CEC considering different application models. We mathematically formulate the problems, design algorithm to solve the problems, and conduct extensive simulation experiments to evaluate the proposed solutions. This thesis makes four main contributions.
1) Propose a framework named Edge Mesh as an abstraction of CEC for our study. Edge Mesh distributes decision-making within the network by sharing data and computation resources among mesh network of edge devices. We describe the functionalities, research framework, and the main principles for designing Edge Mesh and its functions.
2) Solve the problem of data-aware task allocation in CEC, where we consider both the placement and transmission of data to make task allocation decision. Compared

to the traditional problem, the input data for each dependent task is distributed at different edge devices leading to contended network flows. We jointly formulate the task allocation (start time and device for each task) and network flow scheduling (start time of flow) problem. We propose a multi-stage greedy algorithm (MSGA) that solves the problem by jointly considering the placement of tasks and adjustment of network flows.
3) Solve the problem of multi-hop offloading of multiple DAG tasks in CEC. This problem jointly makes a decision of offloading dependent subtasks within each DAG task and scheduling network flows that are generated to transfer data between dependent subtasks. We propose a joint dependent task offloading and flow scheduling heuristic (JDOFH) that solves the problem by leveraging the knowledge of all tasks and start time of network flows.
4) Solve the problem of multi-hop multi-task partial offloading in CEC. Each independent task is partitioned into two parts, i.e. local and remote, where the remote part can be offloaded to an edge device at multi-hop distance. We address the challenging issue of dependency among different variables, including partial offloading ratio, the remote device for each task, start time of the task, and start time of flows. We propose a joint partial offloading and flow scheduling heuristic (JPOFH) that decides partial offloading ratio by considering both waiting times at the devices and start time of input data flows.

In summary, this thesis systematically investigates the requirements and solves the task partitioning and offloading problems in CEC. The proposed solutions address the issues resulting from distributed data sources and multi-hop task offloading in CEC. We also outline future directions, including distributed solutions for dynamic task partitioning and offloading, real-world prototype, integration with blockchain, 5G, etc.

Publications Arising from the Thesis

1. Sahni, Y., Cao, J., Yang, L., & Ji, Y. (2021). Multi-Hop Multi-Task Partial Computation Offloading in Collaborative Edge Computing. IEEE Transactions on Parallel and Distributed Systems, 32(5), 1133-1145. [‡]

2. Sahni, Y., Cao, J., Yang, L., & Ji, Y. (2020). Multi-Hop Offloading of Multiple DAG Tasks in Collaborative Edge Computing. IEEE Internet of Things Journal[‡]

3. Sahni, Y., Cao, J., & Yang, L. (2019). Data-aware task allocation for achieving low latency in collaborative edge computing. IEEE Internet of Things Journal, 6(2), 3512-3524.[‡]

4. Sahni, Y., Cao, J., Zhang, S. and Yang, L., (2017). Edge Mesh: A new paradigm to enable distributed intelligence in Internet of Things. IEEE Access, 5, pp.16441-16458.[‡]

5. Sahni, Y., Cao, J., & Jiang, S. (2019). Middleware for Multi-robot Systems. In Mission-Oriented Sensor Networks and Systems: Art and Science (pp. 633-673). Springer, Cham.

6. Sahni, Y., Cao, J. and Shen, J., (2018). Challenges and Opportunities in Designing Smart Spaces. In Internet of Everything (pp. 131-152). Springer, Singapore.

7. Yang, L., Yang, D., Cao, J., Sahni, Y., & Xu, X. (2020). QoS Guaranteed Resource Allocation for Live Virtual Machine Migration in Edge Clouds. IEEE Access, 8, 78441-78451.

[‡]Presented in this thesis

8. Yang, L., Liu, B., Cao, J., Sahni, Y., & Wang, Z. (2019). Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds. IEEE Transactions on Services Computing.

9. Yang, L., Liu, B., Cao, J., Sahni, Y., & Wang, Z. (2017, June). Joint Computation Partitioning and Resource Allocation for Latency Sensitive Applications in Mobile Edge Clouds. In 2017 IEEE 10th International Conference on Cloud Computing (CLOUD) (pp. 246-253). IEEE.

10. Li, F., Cao, J., Wang, X., Sun, Y., & Sahni, Y. (2017, June). Enabling software defined networking with qos guarantee for cloud applications. In 2017 IEEE 10th International Conference on Cloud Computing (CLOUD) (pp. 130-137). IEEE.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Prof. Jiannong Cao, for his expertise, insightful comments, and patience. I want to thank him for providing me with the opportunity and teaching me so much about research and life in general. I would also like to thank Prof. Yusheng Ji at NII, Japan, for her comprehensive guidance during my exchange study there. I have learned a lot from her about the attitude towards research and professional skill of writing academic papers. I consider myself very fortunate to have the chance to learn from both professors.

I would also like to give my regards to Dr. Lei Yang for all his help and guidance for my research. I really appreciate the time and suggestions he gave to help formulate and solve the different problems in this thesis.

I also owe thanks to my co-authors for their efforts and suggestions - Dr. Shigeng Zhang, Dr. Jiaxing, Shen, Dr. Fuliang Li, and Mr. Shan Jiang.

I am also thankful to other past and current members working on edge computing in the IMCL lab, directed by Prof. Cao, for their collaboration and support. They are Dr. Kongyang Chen, Mr. Mingjin Zhang, Mr. Qianyi Chen, Ms. Jia Wang, Mr. Zhixuan Liang, Mr. Jinlin Chen, and Ms. Dan Wu, .

I have been fortunate to work alongside other members in the IMCL lab including Dr. Wengen Li, Dr. Yuqi Wang, Mr. Yu Yang, Ms. Yanni Yang, Mr. Hanqing Wu, Mr. Dan Li, Mr. Ruosong Yang, Mr. Zhuo Li, Mr. Zhiyuan Wen, and many others.

Finally, I would like to thank my parents, sister, and other family members for their continuous encouragement and support.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background

In the past decade, many applications of Internet of Things (IoT) such as Smart Home, Smart Cities, Smart Healthcare etc. have been deployed where the devices in our surroundings are interconnected to provide better services and comfort to humans. More recently, we witness the emergence of applications in Industrial IoT, supply chains and other areas where the scale of the systems, the number of devices and data being generated continuously increases. However, the cost of sending all the data to a centralized server, as done in cloud computing, for processing and decision-making is high due to large bandwidth consumption and high end-to-end delay. Therefore, the recent trend in 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 has been used in Industry IoT to support real-time processing [1], anomaly detection and fault monitoring [2] [3], improve operational efficiency and management [1], etc. The work in [1] illustrates many scenarios for using edge computing for analytics and decision-making such as to determine whether a worker is not wearing protective equipment or wearing it incorrectly, optimizing inventory management by processing data in real-time, etc. 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 feed 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 of 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 led to the emergence of collaborative edge computing. Collaborative edge computing (CEC) is referred to as a computing paradigm where multiple stakeholders (IoT devices, edge devices, or cloud) collaborate with each other by sharing data and computation resources to satisfy individual and/or global goals. This definition includes both vertical collaboration, where different layers such as device layer, edge computing layer, and cloud layer collaborate with each other, and horizontal collaboration, where different edge devices in edge computing layer collaborate with each other [5].

In this thesis, we aim to answer the research question on how to partition and offloading tasks in a collaborative edge computing environment. We investigate the requirements and issues of task partitioning and offloading in CEC. We consider that each task can require input data from multiple edge devices at different locations and the tasks can be offloaded to an edge device at a multi-hop distance. The transfer of data in the multi-hop network leads to contentious network flows. We mathematically formulate the problems for different application models, and propose heuristic algorithms to solve the problems taking into consideration the distributed data sources and joint task offloading and network flow scheduling. We have done a comprehensive evaluation of the proposed solutions using simulation experiments. The main contributions of the thesis are:

1) Proposed Edge Mesh [6], an abstraction of CEC for our study, where edge devices collaborate with each other in a mesh network by sharing computation and data resources. Edge Mesh enables scalable connectivity and distributed decision-making within the network.

2) Mathematically formulate and solve the data-aware task allocation problem in CEC. We solve the task allocation problem jointly considering the placement of input data for each dependent task on different edge devices and the data transfer, which leads to conflicting network flows.

3) Solve the multi-hop offloading of multiple DAG tasks in CEC. We mathematically formulate and solve the joint problem of offloading dependent subtasks within multiple DAG tasks and scheduling network flows due to transfer of data between dependent subtasks.

4) Solve the multi-hop multi-task partial offloading in CEC. We formulate and solve the problem of partitioning and offloading independent tasks into local and remote parts. The problem jointly considers scheduling network flows generated due to transfer of data for offloading of the remote part.

The rest of the chapter is organized as follows. Section 2 gives details about the proposed Edge Mesh. Section 3 describes the research framework of Edge Mesh. Section 4 elaborates on the research scope of this thesis.

1.2 Edge Mesh

Edge Mesh is an abstraction of CEC that leverages a mesh network architecture of edge devices to enable scalable connectivity and distributed decision-making within the network. Edge Mesh enables both horizontal and vertical integration between devices. Edge Mesh provides many benefits, including distributed processing, low latency, fault tolerance, better scalability, better security, and privacy, which makes it suitable for emerging IoT applications such as Industrial IoT, connected health, etc. Fig 1.1 shows the architecture of Edge Mesh which consists of four types of devices [6].

1. End Devices: End devices are IoT devices that are responsible for sensing and actuation. Devices in a smart home such as camera, lights, thermostat, etc. are some examples of end devices.

2. Edge Devices: Edge devices are system devices with computing or networking resource in between end devices and Cloud. These devices are responsible for decision-making and enabling interaction between end devices. One example of an edge device can be a smartphone, which is connected to both smart home end devices as well as Cloud. Gateway is another example of an edge device, which helps in connecting End devices that use different communication protocols and can also be used for various computing tasks such as processing, storage, load balancing, etc. [7].

3. Routers: Routers are used for relaying data between edge devices. Their function is just to route the data and are not used for processing or enabling decision-making like edge devices. Routers and edge devices together form a mesh network which is used for computation and data sharing among edge devices.

4. Cloud: Cloud provides abundant computing resources including networks, storage, processing, application, services, etc. Traditional IoT systems use a centralized Cloud server for enabling decision-making. However, in the case of Edge Mesh, major decision-making is done by Edge devices instead of Cloud. Cloud is integrated with other devices to be utilized for very specific application requirements that cannot be met using Edge devices. An example of such requirement would be obtaining remote access to devices which cannot be done by use of local Edge Devices or performing big data analytics on historical data.

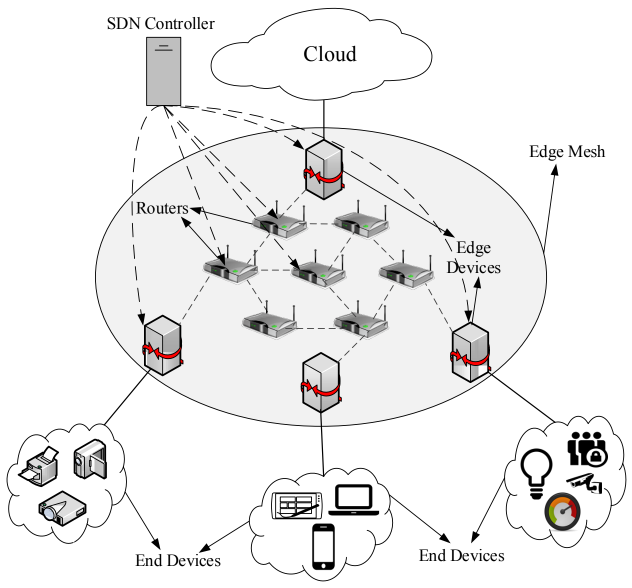

Figure 1.1 Edge Mesh Architecture

The main objective of Edge Mesh is to enable distributed intelligence and decision-making within the network using edge devices. Each edge device is individually responsible for collecting data from end devices, performing localized processing to reduce the amount of data, and collaboration with other edge devices to make decisions. Edge Mesh can also be integrated with software-defined network (SDN) controllers to support software-defined network management. SDN controllers can be placed alongside some selected edge devices for managing the traffic flows among end devices. Edge devices can also act as routers to exchange data among different devices. The cooperation between different edge devices enables different functionalities including task management, QoS provisioning, data sharing, and security and privacy. Edge Mesh has the following four design principles:

1. Decentralized architecture: The intelligence is distributed among edge devices instead of using centralized server. The use of decentralized architecture improves scalability and reliability of the network.

2. Multi-hop connectivity: Since the devices have limited communication range, it is difficult to connect geographically distributed devices. Edge devices are connected using a mesh network which enables communication between faraway devices.

3. Gateway for interoperability: The devices and communication protocols used in IoT applications are heterogeneous which makes it difficult to enable interaction between them. Edge Mesh uses edge devices which act as gateways to support interoperability.

4. Pushing intelligence from cloud to edge devices: Cloud computing suffers from many issues including latency, security, privacy, and mobility. Therefore, Edge Mesh moves computation closer to data sources, enables distributed data storage on edge devices, and allocates tasks among collaborative edge devices and cloud

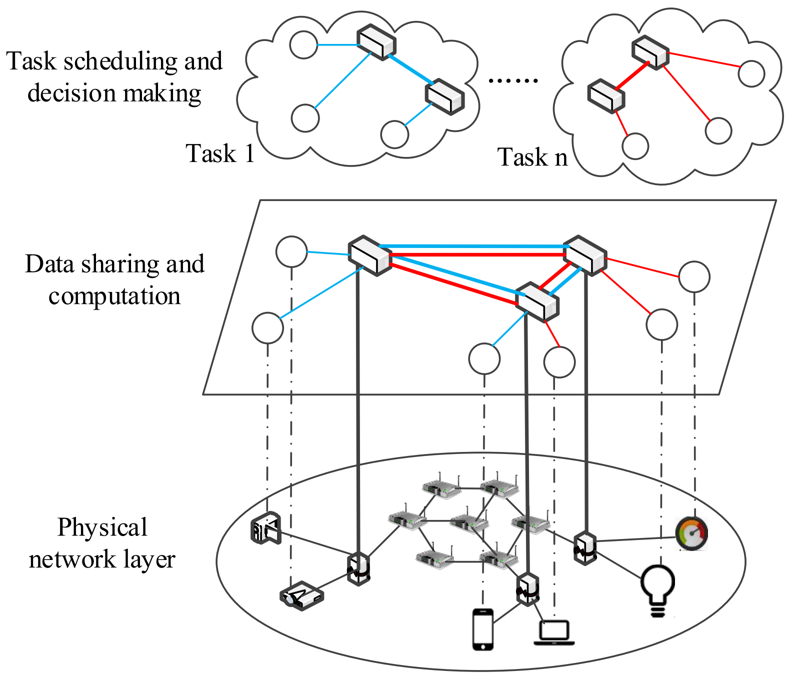
Task scheduling and decision making
Task 1
......
Task n
Data sharing and computation
Physical network layer

Figure 1.2 Layered Edge Mesh

One of the fundamental problems in Edge Mesh is to schedule tasks among edge devices, which includes task partitioning and offloading. Fig 1.2 shows the layered Edge Mesh to illustrate the task management process. The lower layer shows the connection between edge devices in Edge Mesh. The middle overlay layer shows data sharing and computation performed for application tasks on top of edge devices. Here, we do not show the routers but they are responsible for interconnecting different edge devices. The top layer shows the distribution of tasks among different edge devices. Each task, generated at one of the devices, can be partitioned into multiple subtasks and offloaded to different edge devices. The decision for task partitioning and offloading can be done either at a centralized controller or distributed among different devices. The details about the task management in Edge Mesh are described in the paper [6].

1.3 Scope of Research

Fig 1.3 shows the research framework for developing Edge Mesh and its functional components. The framework consists of 6 layers. The bottom two layers represent the different hardware and software possible for edge devices. A collaborative edge computing platform can be use different devices such as Raspberry Pi, Nvdia Jetson Nano, Edge TPU, or servers in general as edge devices. These devices may have different processing architecture (such as x86, armv8, etc.), different operating systems, and different drivers and libraries.

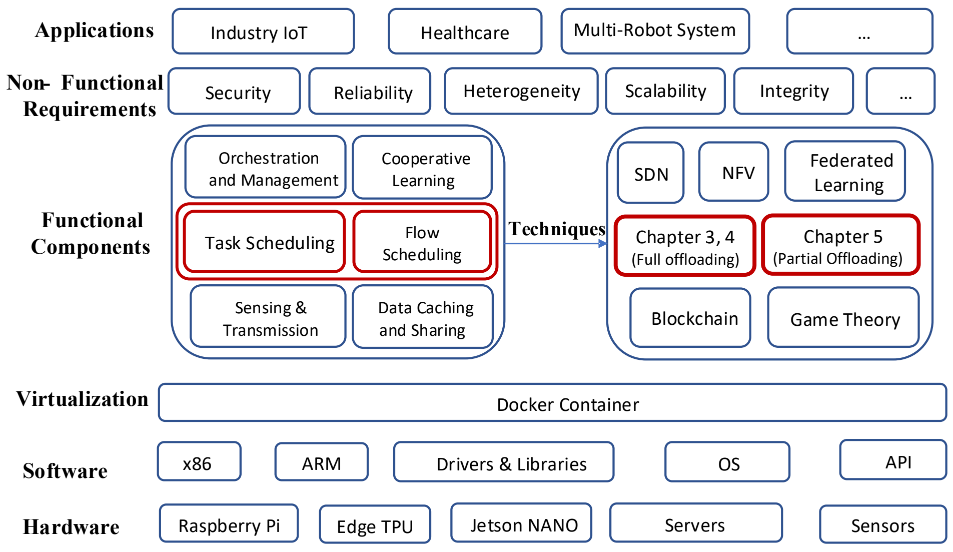

Figure 1.3 Research Framework of Edge Mesh

A virtualization layer is used on top of devices to provide both abstraction from heterogeneous devices and enable migration of computation tasks and data between devices. Docker container is a popular lightweight virtualization technology being used in many existing edge computing platforms. The core part of the platform are the functional components and techniques used to enable the functional components. There are many different functional components corresponding to sensing and transmission, data caching and sharing, task scheduling, flow scheduling, co-operative learning, and orchestration and management. The different functional components can be customized depending on the non-functional requirement of an application. For example, application such as Industry IoT or healthcare having high reliability requirement will require the task scheduling solution to consider reliability as part of objective. The different functional requirement of an application can be satisfied by selecting the appropriate functional component.

This thesis focuses on the task scheduling and flow scheduling functional components in the framework. The main focus of this thesis is formulating and proposing solutions for task partitioning and offloading problems in CEC environment. A characteristic feature of CEC is that the data required for each task can be stored on different edge devices. Therefore, input data needs to be transferred to execute a task. Further, in case of dependent tasks, data from predecessor tasks also needs to be transferred. Another characteristic feature is that tasks can be offloaded to an edge device at multi-hop distance leading to network flows. This implies that task partition and offloading in CEC is affected by other components such as data placement and network flows so, we have to take these into account jointly for effective task partitioning and offloading. Fig 1.4 shows a three-dimensional classification of different works in the thesis. The three dimensions are offloading decision (partial or full), task dependency (independent or dependent), and number of tasks (single or multiple). This classification is done based on the different application model of the problem. The different works have been studied for the same CEC environment, i.e. a static mesh network of edge devices, as shown in the Edge Mesh layer in Fig 1.1.

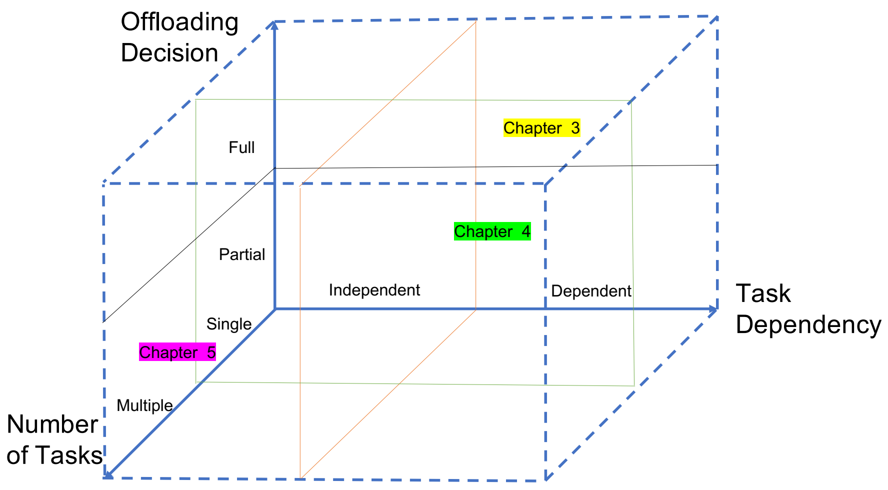

Figure 1.4 3D classification of works in the thesis

The first work in Chapter 3 addresses the problem of data-aware task allocation in CEC considering the data placement and transfer. The problem can be informally defined as allocating different tasks within a task graph to different devices such that the completion time of the application is minimized. However, compared to existing works, we assume that each task in the task graph requires multiple input data which can be stored at different edge devices. The transfer of input data and data from predecessor tasks leads to network flows that compete with each other for bandwidth resources. This problem could be solved by separating the task allocation and network flow scheduling decision; however, we have shown using a specific example and thorough evaluation that this leads to inefficient performance. We have proposed multi-stage greedy adjustment (MSGA) to solve the joint problem by simultaneously adjusting both the task allocation decision and start time of network flow. It is challenging to solve the joint problem as data transmission cost is dynamic, even for a static environment scenario, and dependent on the task allocation decision. Furthermore, a change in network flow decision for a flow can cause congestion for a flow corresponding to predecessor tasks leading to unresolved network conflicts. Due to the nature of the application model, there could also be ”ping-pong effect” where a change in the decision for one network flow causes conflict for another and vice-versa. We have done simulation experiments to evaluate the performance of MSGA and shown that it leads to up to 27% improvement in completion time as compared to benchmark solutions.

The solution proposed in Chapter 3 considers just single task graph and therefore, cannot address the problem of offloading subtasks within multiple directed-acyclic graph (DAG) tasks in CEC environment. Furthermore, the different DAG tasks can be generated at different devices at different release time, leading to additional difficulty. This leads to motivation for our second work in Chapter 4 which addresses this problem of multi-hop offloading of multiple DAG tasks in CEC with the objective of minimizing the average completion time of all DAG tasks. Similar to data-aware task allocation in Chapter 3, there are some common challenges such as conflicting network flows due to data transfer, data transmission cost being dependent on task offloading, and dependency among different subtasks within a task graph. However, MSGA is not suitable and inefficient for solving the offloading problem for multiple DAG tasks as MSGA is developed for a different scenario where each task within the DAG requires several input data. Besides, we additionally consider the release time of different DAG tasks. Therefore, we propose joint dependent task offloading and flow scheduling heuristic (JDOFH), that makes the offloading decision for each subtask within a DAG task while considering the start time of network flows in a single step. We have done simulation experiments to evaluate the performance of JDOFH and shown that JDOFH leads to up to 85%

improvement in average completion time compared to benchmark solutions which do not make a joint decision.

Compared to the other two works, the third work, described in Chapter 5, in this thesis, considers the partial offloading decision for multiple heterogeneous independent tasks in CEC environment. The partial offloading problem has been studied in other existing works; however, due to the characteristic of the CEC environment, we also need to jointly consider the network flow scheduling in this work. The partial offloading decisions for a task implies to determine the remote device where part of the task is executed and when, i.e. the start time, the task is executed. The problem is mathematically formulated as a non-convex mixed-integer nonlinear programming (MINLP) problem and is shown to be NP-hard. We have relaxed the problem to a linear programming (LP) problem using McCormick envelope; however, we can only get a loose lower bound solution. It is challenging to jointly solve the problem as we need to consider the different variables including partial offloading ratio, remote offloading device, start time of the task, waiting time at the devices, and start time of network flows. We solve this problem by proposing a joint partial offloading and flow scheduling heuristic (JPOFH) that decides partial offloading ratio by using the idea from a game-theoretic solution [8] while also considering both waiting times at the devices and start time of input data flows. Performance comparison done using simulation shows that JPOFH leads to up to 32% improvement in average completion time compared to other benchmark solutions.

The problems studied in this thesis have practical relevance and can be beneficial for applications such as large-scale multi-camera video analytics where video and image data from multiple cameras is used for generating situational awareness. Several application scenarios can be modelled using the application models studied in this thesis. One example scenario is to deploy an application involving the use of a different machine learning models with multi-modality input from different sensors. Such scenario also requires other dependent services such as application web service, messaging service, database service, etc. These different services can be modelled using data-aware task allocation problem where each machine learning model is modelled as a subtask requiring multiple input data and other services are modelled as dependent subtasks. Another application scenario is multi-camera person reidentification which involves object detection at each camera. In this case, object detection can be modelled as a DAG task, as in [9], and solved using the multi-hop of multiple DAG task offloading problem studied in the thesis. Multi-hop partial computation offloading is beneficial for applications such as object character recognition (OCR) images captured at different cameras. Many applications such as OCR images can be arbitrary partitioned as studied for multi-hop partial computation offloading problem.

Chapter 2

Literature Review

This chapter describes the different computing paradigms related to edge computing. We also describe different existing works on task partitioning and offloading and highlight the novelty of problems studied in this thesis. The related works corresponding to different problems studied in this thesis are described in the respective chapters.

2.1 Edge Computing Paradigms for IoT

IoT is predicted to have 50 billion devices by 2020. Computing paradigms for IoT must handle the huge scale of devices and application areas. This variability in application and system requirements leads to different views about computation paradigms for IoT. Besides, computing in IoT is not an independent area; it impacts other aspects too such as communication, energy efficiency, physical design, software development, analytics, user experience, security etc. [10]. All these aspects together contribute towards intelligence in IoT. Cloud computing and Edge computing and two most popular computing paradigms being used for IoT.

Cloud provides abundant resources to improve communication, computation, and storage in IoT. However, cloud computing is heavily dependent on Internet connectivity which leads to high latency in remote areas. Besides, the cloud has scalability issue as it is very difficult to send data from a large number of devices due to limited bandwidth constraint. Another issue with cloud is security and privacy as data travels along intermediate networks which can be prone to attacks and if the data is stored at public cloud, then chances of unwanted access and/or compromise become higher. Since Cloud data centers are usually located in a faraway place, latency is higher which means Cloud computing paradigm is not effective for an application that requires mobility support.

Edge computing resolves the issues related Cloud computing by pushing the computation and services closer to data sources at the edge. In literature, different edge computing paradigms have been proposed. Table 2.1 gives the comparison between different edge computing paradigms. Edge computing provides four main benefits: a) reduces network traffic by reducing relying on a centralized server, b) enables real-time data analytics by offloading computing to edge devices, c) provides better scalability compared to the cloud by distributing the computation tasks, and d) enables collaboration between multiple stakeholders with different requirements. Fog computing is one specific paradigm that has been proposed especially for IoT. Fog Computing provides characteristics such as low latency, real-time interaction, distributed analytics, context awareness, geographical distribution, mobility support, which are not supported by centralized Cloud computing paradigm [11]. There are many challenging issues that need to be resolved to realize the full potential of fog computing paradigm. These issues are related to fog networking, Quality of service, interfacing and programming, computation offloading, accounting, billing, monitoring, provisioning and resource management, and security and privacy [12].

Currently researchers are working to integrate Fog and Cloud computing paradigms [13] [14]. Osmotic computing paradigm proposed in [13] aims to decompose applications into microservices and use resources in both edge and cloud to dynamically satisfy application requirements. IFCIoT is another work that has been proposed related to integration where intermediary fog layer [14] provides federated cloud services. Distributed fog nodes collect data from local systems, and updated data is then sent to federated cloud data center which can further perform big data analytics to give a globalized view of whole system [14].

Collaborative edge computing is a specific paradigm that focuses on sharing data and computation resources among edge devices, cloud, and IoT end devices. It focuses on collaboration among devices that can be owned by different stakeholders each having its own objective. Existing works usually focus on vertical collaboration, i.e. between edge devices and cloud, however, horizontal collaboration, i.e. among edge devices, can also help in maximizing the resource utilization to achieve application objectives [5]. A main challenge in collaborative edge computing is to provide incentive for edge devices to share data and computation resources. The work in [15] studies problems related to computation offloading, data caching, interference cancellation in context of collaborative edge computing in 5G networks. Another work in [16] designed a coalition game to share computation resources in small-cell networks. The work in [17] developed a social-graph model to utilize the social relationship among devices.

2.2 Task Partitioning and Offloading

Task partitioning and offloading problem has been widely studied in the literature for edge computing. Several works have also studied problems such as task allocation in wireless sensor networks or task scheduling data centers, which are closely

Table 2.1 Comparison of Edge Computing Paradigms

	Fog Computing [11]	Mobile Edge Computing [18]	Cloudlet Computing [19]	Collaborative Edge Computing [5] [15]
Motivation	To support mobility, location-awareness, and low latency for IoT applications	To increase performance, and provide enriched services in cellular networks	To enable fast response and meet peak bandwidth demands for mobile applications	Enable multiple stakeholders to share and cooperate data
Node Devices	Routers, Switches, Access Points, Gateways	Servers running in base stations	Data center in a box	Servers running in base stations
Node Location	Varying between End devices and Cloud	Base station	Local/Outdoor installation	Base station
Software Architecture	Fog Abstraction layer based	Mobile Orchestrator based	Cloudlet Agent-based	Mobile Orchestrator based
Context Awareness	Medium	High	Low	High
Proximity	One or Multiple Hops	One Hop	One Hop	One Hop
Access Mechanism	Bluetooth, Wi-Fi, Mobile Networks	Mobile Networks	Wi-Fi	Mobile Networks
Internode Communication	Supported	Partial	Partial	Supported

related to problems studied in this thesis.

Task allocation problem has been studied in wireless sensor network (WSN) for both single-hop [20] and multi-hop networks [21], [22]. This thesis also considers task partitioning and offloading in a mesh network of devices where the edge devices can be connected in a multi-hop network. However, compared to the task allocation problems in WSN, we also consider the multiple sources of input data for each dependent task. Previous works such as [22] also consider network resources to make task allocation decision. However, the problem is solved for a single DAG in each time slot. In contrast, we have also solved the task partitioning and offloading problem in this thesis for different application models, including independent tasks, dependent tasks, and multiple DAG tasks.

Several works have studied task scheduling problem in data centers considering different application models such as independent tasks, single DAG task [23], or multiple DAG tasks [24]. Similar to problems in this thesis, some works have also considered data placement into consideration for task scheduling [25] [26]. Many problems in data centers do not consider bandwidth resources as task are scheduled in a cluster environment [27]. Some works such as [28], [29], etc. have considered network bandwidth to make a task scheduling decision. However, compared to the problems in this thesis, they do not jointly consider network flow scheduling, where we also need to make a decision on the start time of the flow. We have also solved the partial offloading problem for independent tasks while jointly considering network flow scheduling.

Existing works have solved the task partitioning and offloading problem in edge computing considering issues such as dynamic offloading [30], multiple devices [31], hybrid edge-cloud environment [32], etc. Existing works usually solve the task offloading problem considering single-hop connectivity between devices. Some recent works such as [33], [34], [35], [36], etc. have also addressed the problem for a multi-hop network. However, these problems do not jointly consider network flow scheduling. We have solved the task partitioning and offloading problem considering the network flow scheduling. Furthermore, we also consider that each task can require input data from multiple sources. We have solved the problem for both full and partial offloading of tasks taking into consideration the scheduling of network flows.

Chapter 3

Data-Aware Task Allocation[*]

sahni2020multi

3.1 Introduction

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 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 chapter, we study the data-aware task allocation problem to jointly schedule task and network flows in collaborative edge computing with the objective of

[*]Based on work published in [37]

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 3.3. The problem considers the placement of input data and the network bandwidth consumed in transferring data to schedule tasks. The data transfer is both due to 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) [20] [21] [22] and IoT [38] [39] 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 [40], [25], [41], [42], [26], [43],[44], 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 [29] 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 Edge Mesh. 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 made performance comparisons by changing different parameters in the simulation including the number of tasks, 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 chapter is organized as follows. In Section 2, we describe the related works, In Section 3, we have given a motivational example to illustrate the importance of data-aware task allocation problem. In Section 4, we give the system model and problem formulation. In Section 5, we discuss the proposed solution, MSGA, for the data- aware task allocation problem. In Section 6, we have done the performance evaluation. Finally, we give the conclusion in Section 7.

3.2 Related Work

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

Existing works related to tasks allocation in WSN have considered both single hop [20] and multi-hop [21], [22] WSN. The work in [22] 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 [45]. There are few works that also consider task allocation problem in IoT such as [38] and [39]. In [38], 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 [39] 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 [6]. However, unlike existing works, this work jointly schedules tasks and network flows for collaborative edge computing. However, the work in [6] 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 [40], [25], [26], [41], [42], [43],[44] etc. A related work has been done for dependent tasks in [46]. Authors in [46] 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 [29] 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 [47] [48].The work in [47] 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 [48], 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. These works consider MapReduce jobs or join queries whereas, our work considers generic task DAG model. We also consider the placement of the input data of different tasks in the DAG while making task allocation decision.

3.3 Motivational Example

The problem is to allocate a set of tasks within an application shown in Fig 3.1 to a set of devices shown in Fig 3.2 such that overall completion time is minimized. Fig 3.1 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 trans-

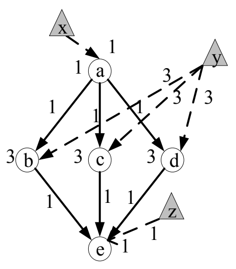

Figure 3.1 Task DAG

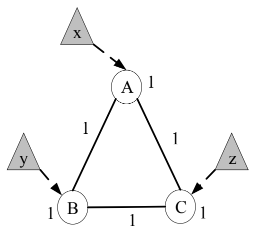

Figure 3.2 Network Topology

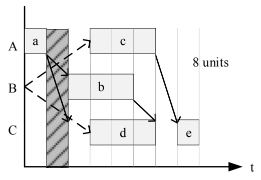

Figure 3.3 Task scheduling without considering network congestion

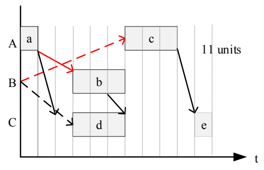

Figure 3.4 Task scheduling under congestion using bandwidth sharing

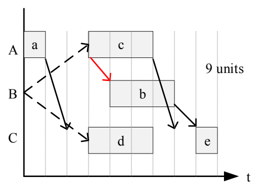

Figure 3.5 Task scheduling under congestion using flow adjustment

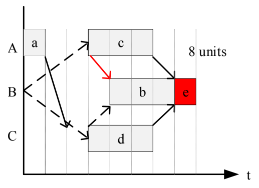

Figure 3.6 Task scheduling under congestion using both flow and destination adjustment

ferred. The weight of edges connecting triangle nodes and circle nodes represents the amount of input data to be transferred to the task. Fig 3.2 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 3.2 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 3.3 which minimizes the completion time. The completion time of the application using this schedule is 8 units.

The schedule shown in Fig 3.3 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. 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 flows. The new schedule created using bandwidth sharing policy, shown in Fig 3.4, 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 2 to 3 units for both the flows in Fig 3.4.

We can create another schedule by changing the start time of each flow instead of sharing bandwidth as shown in Fig 3.5. The completion time using flow adjustment is 9 units. The schedule shown in Fig 3.5 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 3.6, by using both flow and destination adjustment. 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 3.4 and Fig 3.5 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 3.6, better results can be achieved. The rest of the chapter 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.

3.3.1 Problem Formulation

The data-aware task allocation problem is to jointly schedule tasks and networks flows in Edge Mesh such that completion time of application is minimized. There are two assumptions made in the problem, which are:

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

This section describes the modeling of application, network, data, and cost functions. The problem is formulated as a mixed-integer nonlinear programming problem.

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 i' is $P_{ii'}$, 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 $\cup$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. 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 3.8 shows an example of task graph model and the corresponding data flow model. The input data required for different tasks in Fig 3.7 are shown in Table 3.9.

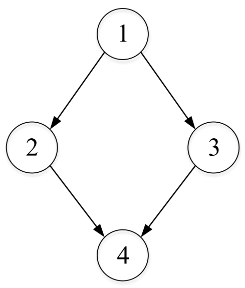

Figure 3.7 Example of task graph model

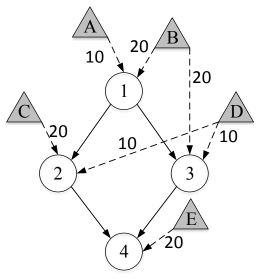

Figure 3.8 Corresponding dataflow model

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

Figure 3.9 Set of input data for each task

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 [49]. 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 (3.1).

$$Tcomp_{i,j} = \frac{c_i}{p_j} \tag{3.1}$$

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.

$$Ts_{i,j} = \max(avail_j, \max_{1 \leq r \leq R_i}(Tf_r + Ttask_{r,i})) \tag{3.2}$$

$$Tf_{i,j} = Ts_{i,j} + Tcomp_{i,j} \tag{3.3}$$

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 i-th task starts to execute, x_i denotes the device where the i-th task is executed, where $1 \leq i \leq K$ and $1 \leq x_i \leq N$. The completion time of i-th task is Tf_i, where $Tf_i = Ts_i + Tcomp_i$. $Tcomp_i$ is the time to compute i-th task. 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, 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 transfer of input data then 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 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.

Data-Aware Task Allocation problem

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

Objective:

$$minimize \quad \max_{i \in V}\{Tf_i\} \tag{3.4}$$

Constraints:

$$\forall (i, i') \in Z, \quad Tf_i \leq Ts_{i'} \tag{3.5}$$

$$\begin{aligned} &\forall (i, i') \in V, \quad |x_i - x_{i'}| * Q \\ &+ (Ts_i - Tf_{i'}) * (Tfi - Ts_{i'}) \geq 0 \end{aligned} \tag{3.6}$$

$$\forall (i, i') \in Z, St_{fl_{i,i'}} \geq Tf_i \tag{3.7}$$

$$\forall (i, i') \in Z, Et_{fl_{i,i'}} \leq Ts_{i'} \tag{3.8}$$

$$\begin{aligned} &\forall (i, i') \in Z, \\ &|x_i - x_{i'}| * (\sum_{\tau = St_{fl}}^{Et_{fl}} \mathbb{R}_{fl}(\tau) - D_{i,i'}) = 0 \end{aligned} \tag{3.9}$$

$$\begin{aligned} &\forall \tau \in [0, T-1], \forall e \in E, \\ &\sum_{fl_{i,i'}} [\mathbb{R}_{fl}(\tau) * \mathbb{Y}(e, E_{fl})] \leq B_e \end{aligned} \tag{3.10}$$

where Q in Equation (3.6) 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 (3.4) is the objective function to minimize the total completion time of the application. Equation (3.5) indicates that the dependent task can only start after its preceding task is completed. Equation (3.6) 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. Equation (3.7) defines that the flow can start only after its release time, which is the finish time of preceding task. Equation (3.8) defines that the flow must finish before its deadline, which is the start time of the current task. Equation (3.9) represents that the rate of flow should be such that it is finished in time. Equation (3.10) represents that the amount of bandwidth consumed by each link at given time must be less than the given bandwidth capacity of the link.

The data-aware task allocation problem is NP-hard as it 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 [49].

3.4 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 3.10. The advantage of MSGA is that we can substitute part of the algorithm to create a different solution. For example, instead of using a greedy algorithm for creating 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.

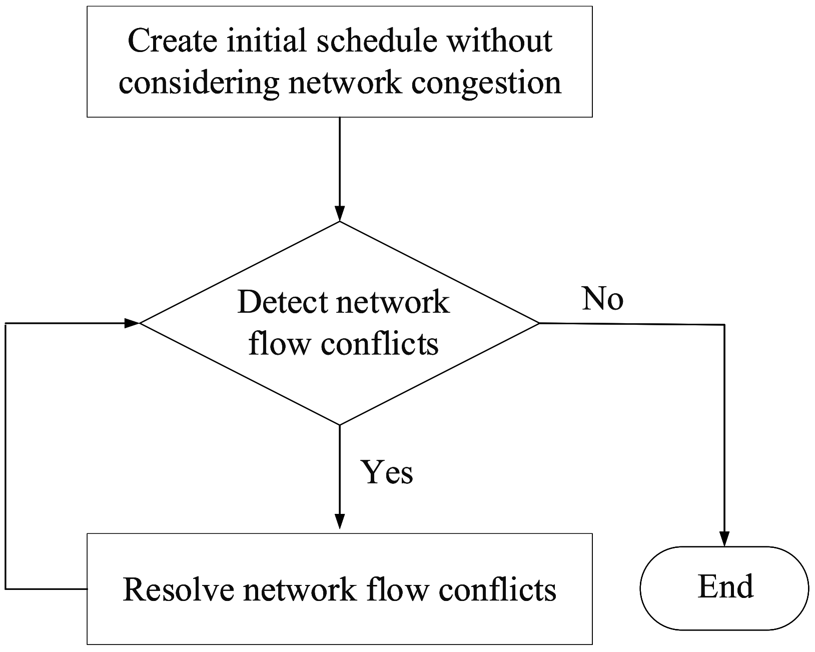
Create initial schedule without considering network congestion
Detect network flow conflicts
No
Yes
Resolve network flow conflicts
End

Figure 3.10 Flowchart of MSGA

The first stage is creating an initial schedule using a greedy method based on list scheduling without considering the network congestion (**Algorithm1, Line 1**). The method is similar to HEFT algorithm proposed in [49] 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 Equation (3.11) and (3.12) respectively. Unlike the Equations (3.2) and (3.3), 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 the initial schedule.

$$Ts_{i,j} = \max(avail_j, \max_{1 \leq r \leq R_i}(Tf_r + Ttask_{r,i}), \max(Tdata_i)) \tag{3.11}$$

$$Tf_{i,j} = Ts_{i,j} + Tcomp_{i,j} \tag{3.12}$$

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 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 (3.13). 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.

$$T_{f_{i,i'}} = \frac{D_{i,i'}}{\min_{e \in E_{fl}} B_e} \tag{3.13}$$

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 third stage of the algorithm to resolve conflict (**Algorithm1, Line 4 -15**). The second and third stage are repeated until their 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 chapter. 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, and initial schedule, E_s, is calculated (**Algorithm 1, Line 9**). The second method of destination adjustment is applied on 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 a 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. These issues are:

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**).

3.4.1 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 made for all conflicts which are 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)).

3.5 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 simulation experiments have been conducted on MacOS with 2.7 GHz Dual-Core Intel Core i5 processor and 8 GB RAM. The parameters used for the simulation are similar to the one used previously in [22] and [45].

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 initial schedule using 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 $F_{cl}$ that are in conflict with the current congested flow
         $f_n$ in the existing schedule $E_s$ ;
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, $E_f$, considering the flow adjustment;
9      Calculate the delay in overall completion time, $\Delta t_f$, caused by flow
         adjustment ;
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 $E_d$ considering the destination adjustment ;
12     Calculate the increase in overall completion time, $\Delta t_d$ caused by
         destination adjustment;
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 $E_s$;
```

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 $\Delta t_d$
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          $t_{i,f} = Inf$ ;
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 $E_d$ considering the destination adjustment;
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 $E_d, \Delta t_d$;
```

3.5.1 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. 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 ± 10%]. The bandwidth of each link is selected to be [250Kbps ± 10%].

Parameters for Application Model: We have implemented a random DAG generator for the task graph using the level by the level method mentioned in [50]. 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 ± 10%] and data transferred between two dependent tasks is selected to be [750 bits ± 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 ± 10%].

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. In terms of 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 of 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 work is similar to the one in [6]. 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 the gene selected randomly. We use power mutation described in [51]. 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

Table 3.1 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 ± 10%
Computation load of each task	300 KCC ± 10%
Processing power of each device	50 MCPS ± 10%
Data transmission between dependent tasks	750 bits ± 10%
Bandwidth of each link	250 Kbps ± 10%

is selected for crossover, and mutation ratio is 0.02.

3.5.2 Simulation Results

The default parameters used for simulation are shown in Table 3.1. Table 3.2 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 first benchmark solution, EFT (earliest finish time) to represent second benchmark solution, and GA to represent the genetic algorithm. Compared to other algorithms, MSGA can 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

Table 3.2 Performance Comparison for default parameters

Metric	**FCFS**	**EFT**	**GA**	**MSGA**
Completion time (sec)	0.2188	0.2456	0.2175	0.1805
Running time (sec)	33.3162	34.9928	27676	93.1934

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 algorithms are proposed for offline environment where the task offloading decision is determined before the execution. Therefore, the overhead of the running time does not affect the task execution schedule. Furthermore, the running time is dependent on the algorithm implementation and the machine where the simulation experiments are conducted. 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.

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 3.11 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 3.12. Since MSGA also considers changing the placement of the tasks, its running time 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.

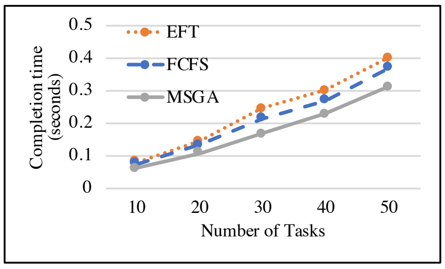

Figure 3.11 Effect of number of tasks on completion time

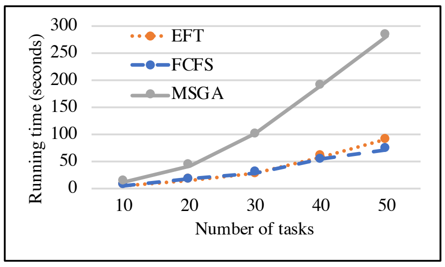

Figure 3.12 Effect of number of tasks on running time

Effect of changing number of devices

Fig 3.13 shows the performance comparison, in terms of completion time, by changing the number of devices from 50 to 250. Performance comparison indicates that the completion time decreases as the number of devices are increased because the number of connections are 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 are 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 too large then the difference between MSGA and benchmark solutions will start to decrease as the benchmark solutions can give good results for a very large number of devices. This can be observed in Fig 3.13 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 3.14. 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.

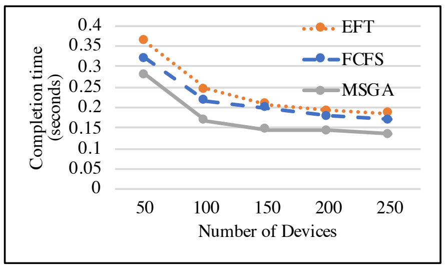

Figure 3.13 Effect of number of devices on completion time

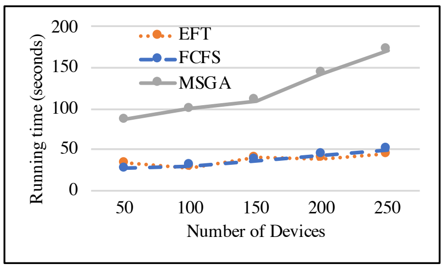

Figure 3.14 Effect of number of devices on running time

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 3.15. 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 difference 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 3.16. 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.

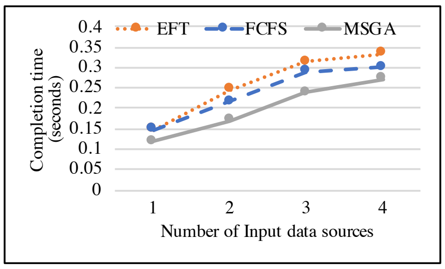

Figure 3.15 Effect of number of input data sources on completion time

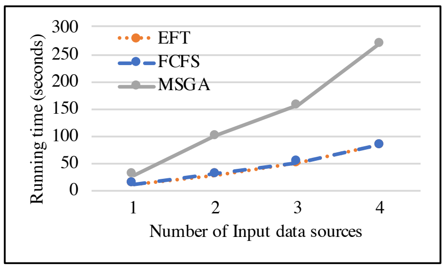

Figure 3.16 Effect fo number of input data sources on running time

Effect of changing amount of input data

Fig 3.17 shows the effect of increasing the amount of average input data from 1600 bits to 6400 bits. Fig 3.17 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 when the input data increases. 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 significant increase of completion time when the amount of input data is large, 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 3.18. The reason is that as that there is no significant difference in network congestion on increasing the amount of input data.

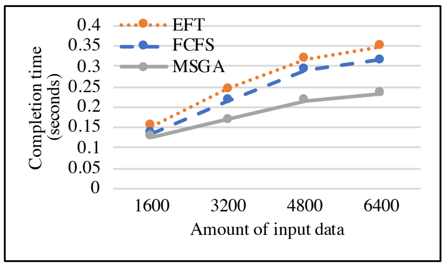

Figure 3.17 Effect of amount of input data on completion time

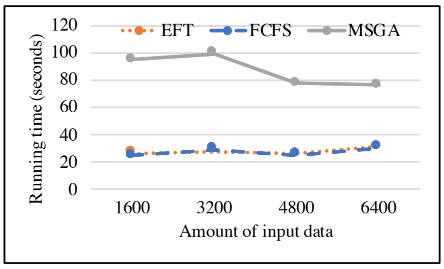

Figure 3.18 Effect of amount of input data on running time

3.6 Conclusion

In this chapter, we study the data-aware task allocation problem where task and network flows are jointly scheduled with the objective of minimizing the completion time of application. The problem is solved for a a single DAG application task graph where each task in the DAG requires input data both from predecessor tasks and other distributed sources. 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.

The data-aware task allocation problem can be investigated further by including other objective functions such as jointly optimizing both energy consumption and completion time of the task. We also do not consider offloading the tasks to centralized cloud server. Further, the dynamics of network and workload have also not been considered in this work. We assume static environment where both the task and network resources are known beforehand and do not change while allocating sub-tasks.

Chapter 4

Multi-Hop Offloading of Multiple DAG Tasks[*]

4.1 introduction

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

[*]Based on work published in [52]

[34], [33], [35], [36], 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 chapter 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 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 [23], [24], [27], etc. These works, however, make task scheduling decision without jointly considering network flow scheduling. Some works such as [28] consider network resources but do not leverage dependency among tasks to make task offloading decision. Other works such as [29] 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 [53], [54], etc. The work in [53] discusses the conceptual idea of collaborative edge computing for video processing. The work in [54] 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 [55], small-cell base stations [16], etc. The work in [56] developed a lightweight virtualization model to support collaboration among edge devices in smart city and other related applications.

The main contribution made in this work 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 chapter is organized 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.

4.2 Related Works

Existing works in literature have addressed different types of computation offloading problem. However, most of these works consider single-hop computation offloading problem. Some recent works such as [33], [34], [35], [36], 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 [49] 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 [49]. Recently some have studied the problem of scheduling dependent tasks for various scenarios including single DAG task [23], multiple DAG tasks [24], within a cluster [27], across geo-distributed clusters [28], etc. The work in [27] proposes a scheduler, Graphene, to place multiple tasks with dependencies within a cluster by identifying the troublesome subtasks within each task. The work in [28] proposes Tetrium for scheduling tasks with dependencies in geo-distributed clusters by considering both computation and network resources. The work in [23] gave a lower bound solution for scheduling a single DAG task. Another recent work [57] 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 task. These works, however, either do not jointly consider network flow scheduling such as in [27], [23], [57], or do not take dependencies among subtasks to make decisions [28]. Another recent work [58], similar to our work, also considers different stages within a DAG task to make scheduling decisions. However, this work [58] does not consider multiple DAG tasks and network flow scheduling.

Some works in literature such as [29], [59], [47], [48], etc. have considered network bandwidth while making task scheduling decisions. The work in [29] assumes the underlying network scheduler to make task scheduling decisions. Another work [59] 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 [60] and [61] 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 [60] considers each DAG job as stages of cotasks and makes task scheduling decisions. However, it does not consider network flow scheduling. The work in [61] considered cotask offloading problem for independent tasks and did not consider network flow scheduling.

Our previous work in [37] studied data-aware task allocation problem while jointly considered network flow scheduling. However, the work in [37] made scheduling decision for a single task DAG and did not consider heterogeneous release time of tasks. Besides, this work 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 [37].

4.3 System Model and Problem Formulation

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

4.3.1 System Model

Fig 1.1 shows the system architecture of Edge Mesh, an abstraction collaborative edge computing, where the intelligence is distributed and pushed within the network by sharing computation resources and data between mesh network of edge devices [6]. Edge devices is such an architecture can be heterogenous in computation capacity and can also serve as routers as shown in Fig 1.1. Due to 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. SDN controller 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 1.1. There are some previous works such as [62], [63], etc. which have used the SDN controller to make scheduling decisions in wireless networks. The work in [64] proposed meSDN to extend the control of SDN to mobile devices. The work in [65] implemented a prototype for the proposed algorithm in [63].

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 [37]. 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 [66] [67]. Furthermore, the offloading problem in this work is solved for DAG tasks; however, we do not focus on how the application is modelled as a dependency graph. The work in [68] 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 [69], [70], 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 offloading 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 [23], [35], [71] etc. have used similar assumptions to calculate the total cost.

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 1.1, 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

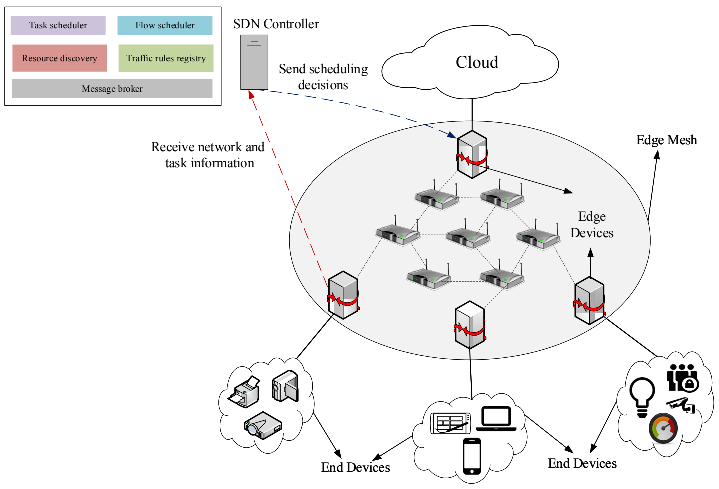

Figure 4.1 System Architecture

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 4.2. 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.

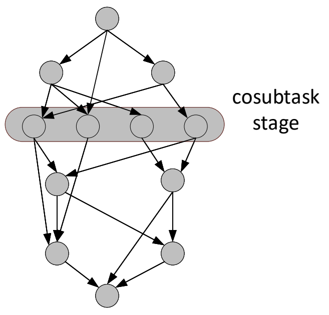

Figure 4.2 Example DAG Task

4.3.2 Problem Formulation

This section describes the constraints and formulates the problem as an optimization problem.

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 (4.1).

$$\sum_{k \in V} X_{ijk} = 1, \qquad \forall i \in A, \quad j \in T_i \tag{4.1}$$

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 (4.2).

$$|X_{ijk} - X_{pqk}| * L + (Ts_{ij} - Tf_{pq})) * (Tf_{ij} - Tspq) >= 0 \\ \forall i, p \in A, \quad j \in T_i, \quad q \in T_p, \quad k \in V \tag{4.2}$$

Since each task contains dependent subtasks, a subtask can start only after preceding subtasks have finished as represented by Equation (4.3).

$$Ts_{ij} \geq Tf_{iv} \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \tag{4.3}$$

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 (4.4).

$$Ts_{ij} \geq Flt_{ivj} \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \tag{4.4}$$

The relationship between finish time and start time of a subtask is represented by Equation (4.5). The finish time of a subtask is equal to sum of start time and time to execute the subtask at the device.

$$Tf_{ij} = Ts_{ij} + \frac{CL_{ij}}{PS_k} * X_{ijk} \quad \forall i \in A, \quad j \in T_i, \quad k \in V \tag{4.5}$$

There are some additional constraints, Equation (4.6)-(4.8), 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.

$$X_{iN'_i z_i} = 1, \quad \forall i \in A \tag{4.6}$$

$$Ts_{iN'_i} = Trel_i, \quad \forall i \in A \tag{4.7}$$

$$Tf_{iN'_i} = Trel_i, \quad \forall i \in A \tag{4.8}$$

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 (4.9).

$$St_{ivj} = Trel_i \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \tag{4.9}$$

The network flow start only after the preceding subtask has finished as represented by Equation (4.10).

$$St_{ivj} = Tf_{iv} \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \tag{4.10}$$

This work assumes that two flows cannot pass through a link at the same time as represented by Equation (4.11). 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.

$$\begin{gathered}|Y_{wke} * X_{ijk} * X_{ivw} - Y_{lme} * X_{prl} * X_{pqm}| * L \quad + \\ (St_{ivj} - Flt_{prq}) * (Flt_{ivj} - St_{prq}) >= 0 \\ \forall i, p \in A, \quad j, v \in T_i, \quad r, q \in T_p, \quad k, w, l, m \in V\end{gathered} \tag{4.11}$$

The finish time of a flow can be calculated directly once the start time is known. Equation (4.12) represents the relationship between start time and finish time of a data flow.

$$\begin{gathered}Flt_{ivj} = St_{ivj} + \frac{D_{ivj} * H_{wk}}{R_{wk}} * X_{ijk} * X_{ivw} \\ \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij}, \quad k, w \in V\end{gathered} \tag{4.12}$$

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 [23]. The completion time of a task, represented by Equation (4.13), is the difference between the time instance when the last subtask is finished and the time instance when the task is released.

$$F_i = \max_{j=1,\dots N_i'} Tf_{ij} - Trel_i, \quad \forall i \in A \tag{4.13}$$

The constraint of the range of binary decision variable X_{ijk} is represented by Equation (4.14).

$$X_{ijk} = \{0, 1\}, \quad \forall i \in A, \quad j \in T_i, \quad k \in V \tag{4.14}$$

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:

$$\underset{X_{ijk}, Tf_{ij}, Ts_{ij}, St_{ivj}, Flt_{ivj}, F_i}{\text{minimize}} \frac{\sum_{i=1}^{J} Tft_i}{J}, \tag{4.15}$$

subject to (4.1) - (4.14) $\quad \forall i \in A, \quad j, v \in T_i, \quad k \in V$

This problem can be reduced to shop-scheduling problem [72] 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 [72]. Hence, the problem in this work is also NP-hard. Since we cannot find an optimal solution in polynomial time. Therefore, we have proposed a heuristic solution.

4.4 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 4.2. 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 of-

floading 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 (4.17). The rank metric for each cosubtask stage within a task is calculated as defined in Equation (4.18).

$$val_{ij} = \max_{v \in Pd_{ij}}(val_{iv} + \frac{D_{ivj}}{\overline{\overline{BW}}}) + \frac{CL_{ij}}{\overline{\overline{PS}}}, \quad \forall i \in A, \quad j \in T_i \tag{4.16}$$

$$rval_{ij} = val_{ij} + Trel_i \quad \forall i \in A, \quad j \in T_i \tag{4.17}$$

$$rank_{is} = \max_{j \in U_{is}}(rval_{ij}) \quad \forall i \in A \tag{4.18}$$

Cosubtask stages are given priority in the increasing order of rank metric calculated in Equation (4.18). 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 [60].

The schedule for the selected subtask is determined by selecting the device, which leads to minimum finish time, as shown in Equation (4.22). However, in order to do that, we need to calculate the start time of the subtask at each device, as shown in Equation (4.19). 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.

$$\begin{aligned} Tas_{ijk} &= \max\{Tavail_{ijk}, \max_{v \in Pd_{ij}}(Tcomm_{ijvk} + \frac{D_{ivj} * H_{wk}}{R_{wk}})\} \\ &\forall i \in A, \quad j, v \in T_i, \quad k \in V \end{aligned} \tag{4.19}$$

$$\begin{aligned} Tcomm_{ivjk} &= \max\{Tf_{iv}, \max_{e \in P_{wk}}(Y_{wke} * Y_{lme} * Flt_{prq})\} \\ &\forall i, p \in A, \quad j, v \in T_i, \quad r, q \in T_p, \quad k, w, l, m \in V \end{aligned} \tag{4.20}$$

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 (4.21). The device corresponding to the minimum finish time calculated in Equation (4.22) is selected for executing the subtask as shown in Equation (4.23).

$$Taf_{ijk} = Tas_{ijk} + CT_{ijk}, \quad \forall i \in A, \quad j, v \in T_i, \quad k \in V \tag{4.21}$$

$$Tf_{ij} = \min_{k \in V} Taf_{ijk}, \qquad \forall i \in A, \quad j \in T_i \tag{4.22}$$

$$X_{ijk^*} = 1, \qquad \forall i \in A, \quad j \in T_i \tag{4.23}$$

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 (4.24) to (4.26).

$$Ts_{ij} = Tas_{ijk^*}, \quad \forall i \in A, \quad j \in T_i \tag{4.24}$$

$$St_{ivj} = Tcomm_{ijvk^*}, \quad \forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \tag{4.25}$$

$$\begin{aligned} Flt_{ivj} &= St_{ivj} + \frac{D_{ivj} * H_{wk^*}}{R_{wk^*}} \\ &\forall i \in A, \quad j \in T_i, \quad v \in Pd_{ij} \end{aligned} \tag{4.26}$$

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 $O(|A|^2 * \overline{N}^2 * |V|^2 * \overline{P})$. 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$O(|A|^2 * \overline{N}^2 * |V|^2 * \overline{P})$.

Algorithm 3: 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     $Tft_i \leftarrow \max Tf_{ij}$
```

4.5 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 simulation experiments have been conducted on MacOS with 2.7 GHz Dual-Core Intel Core i5 processor and 8 GB RAM. The parameters used for simulation are in a similar range to the one used previously in [23] and [73].

4.5.1 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 [73] and [37]. 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 [74]. 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 l is constrained to have at least one number, and the number of nodes in each level 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.

4.5.2 Benchmark Solutions

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 comple-

tion 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 [49] 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 [75], 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

Table 4.1 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

algorithm. ALT schedules the tasks sequentially unlike JDOFH where co-subtask 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) [75] as subtask with earliest start time will have the earliest deadline for the corresponding network flow.

4.5.3 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 [49], [23], 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 5.1 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.

The CDF plot shown Fig 4.3 shows that both joint solutions, JDOFH and ALT lead to less completion time for tasks compared to other benchmark solutions. JDOFH achieves slightly better performance than ALT as it considers that waiting time and priority of subtasks. Table 5.1 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 5.1 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 work focuses on proposing a better solution in terms of completion time. The algorithms are proposed for offline environment where the task offloading decision is determined before the execution. Therefore, the overhead of the running time does not affect the task execution schedule. Furthermore, the running time is dependent on the algorithm implementation and the machine where the simulation experiments are conducted.

We have also done performance comparison, in terms of completion time, by

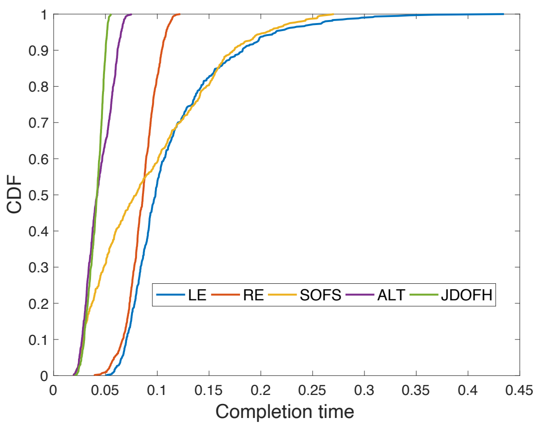

Figure 4.3 CDF plot for FFT DAG

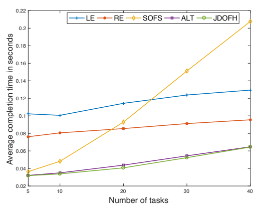

Figure 4.4 Effect of changing number of tasks for FFT DAG

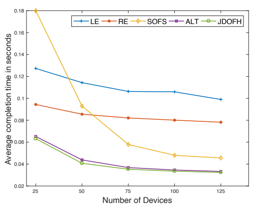

Figure 4.5 Effect of changing number of devices for FFT DAG

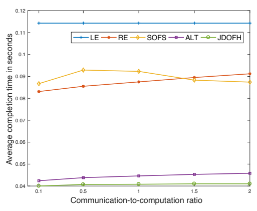

Figure 4.6 Effect of changing CCR of task graph for FFT DAG

varying different simulation parameters. Fig 4.4, 4.5, and 4.6 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.

4.5.4 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 4.5.1. Compared to FFT DAG, the number of subtasks in each randomly generated DAG can be higher and different. Table 5.2 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. Fig 4.7 shows the CDF plot for completion time comparing the performance of different 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

Table 4.2 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

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.

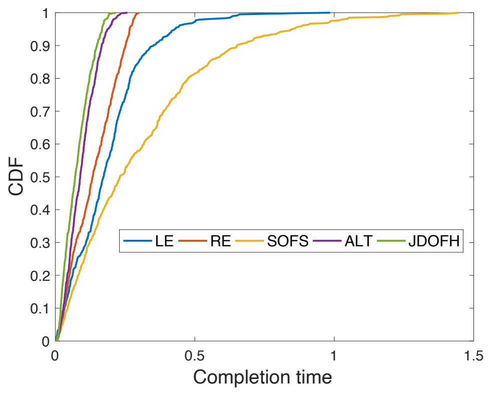

Figure 4.7 CDF plot for randomly generated DAG with default parameters setting

Effect of change in number of task

Fig 4.8 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 increase in waiting time at the devices to execute the sub-tasks and increase in total number of network flows.

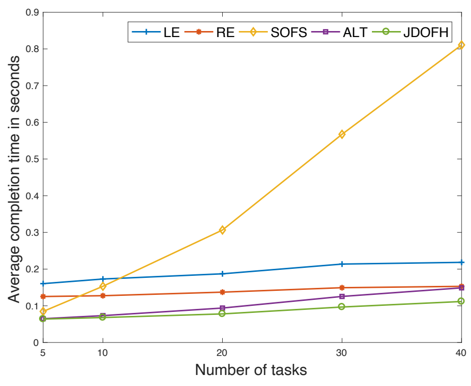

Figure 4.8 Effect of changing number of tasks for randomly generated DAG

The performance difference between LE and JDOFH decreases from 60.22% at 5 tasks to 48.85% at 40 tasks. This decrease in gap is observed because LE is not affected by increase in number of network flows with increase of number of tasks. 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 increase in network flows there is decrease in 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 increase in 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 low number of tasks, there is less contention among resources so both JDOFH and ALT perform almost equivalently. However, as the number of tasks are increased, JDOFH perform better as it leverages cosubtasks stages to determine execution schedule. ALT, on the other hand, requires large waiting time at the devices to finish each task sequentially based on the order of release time.

Effect of change in number of subtasks

We have evaluated the effect of changing the number of subtasks within each randomly generated task DAG as shown in Fig 5.3. 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.

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 signifi-

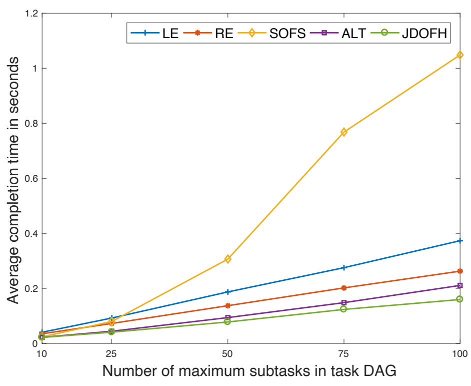

Figure 4.9 Effect of changing number of subtasks for randomly generated DAG

cant 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.

Effect of change in number of devices

Fig 5.5 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.

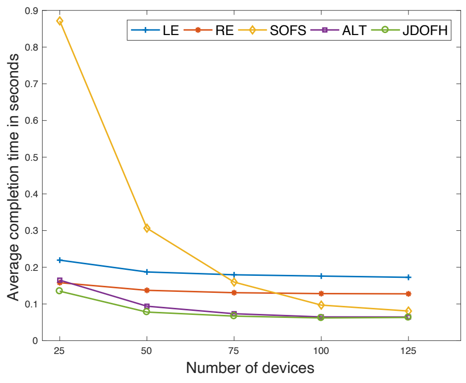

Figure 4.10 Effect of changing number of devices for randomly generated DAG

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.

Effect of change in communication to computation ratio (CCR)

Fig 4.11 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.

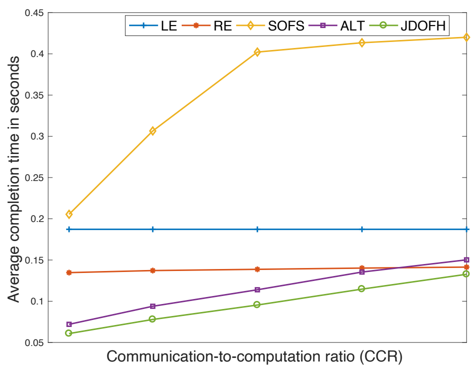

Figure 4.11 Effect of changing CCR of task graph for randomly generated DAG

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.

4.6 Conclusion

In this chapter, we study 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 this problem making some assumptions including static network and no explicit modelling of wireless interference. We can extend this problem by considering the dynamics of network and device resources. Another direction for extending this work is considering the scheduling of different subtasks, i.e, the order of execution, offloaded on a same device. The problem formulation considers the start time of the each subtask, however, the proposed solution makes scheduling decision for each subtask based on a defined priority metric.

Chapter 5

Multi-Hop Multi-task Partial Offloading*

5.1 Introduction

This chapter studies the problem of multi-hop multi-task partial offloading in collaborative edge computing with the objective of minimizing the average completion of all tasks. The problem jointly considers partial offloading of independent tasks generated at different edge devices to any other edge device at a multi-hop distance and network flow scheduling. The partial offloading of a task implies that a task is partitioned into two components where one component is locally executed and other is offloaded to a remote device. Network flow scheduling includes not only making a decision on routing path but also start time of input data flows. The decision of partial offloading is interdependent on network flow scheduling.

There are some works in literature such as [34], [33], [35], [36], etc. that have studied multi-hop computation offloading problem. Partial offloading problem has

*Based on work published in [76]

also been studied in previous works such as [77] and [78]. However, these works do not consider both partial offloading in a network with multiple remote edge devices and network flow scheduling which leads to an increase in the number of decision variables and makes the problem more challenging. Another main challenging issue is that data transmission cost is no longer constant and difficult to estimate as it depends on the partial offloading decision. The partial offloading decision can affect the start time of input data flows which is included in data transmission cost. 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. Another application scenario is optical character recognition (OCR) images which can be arbitrarily divided as assumed in this work [79].

The main contribution made in this work are:

1. We have mathematically formulated the joint multi-hop multi-task partial computation offloading and network flow scheduling problem in collaborative edge computing with the objective of minimizing the average completion time of independent tasks. We consider the heterogeneity in independent tasks where each task has different computation load, input data, and release time. We also consider heterogeneity in device processing speed and link bandwidth.

2. We show the problem is NP-hard and therefore, propose a joint partial offloading and flow scheduling heuristic (JPOFH) which creates a priority of tasks considering release time and computation load and calculates the partial offloading ratio considering the waiting time at the devices and start time of input data flows. We also relaxed the formulated MINLP problem to an LP problem using McCormick envelope and solve it using Mosek solver in CVX to give a lower bound solution.

3. We have conducted simulation experiments to evaluate and compare the performance, in terms of average completion time of tasks and running time, of JPOFH against benchmark solutions. The benchmark solutions consider different possibilities, including local execution, remote execution, and separate partial offloading and flow scheduling solution. We make a comprehensive performance comparison by changing the values of different input parameters, including the number of tasks, number of devices, number of routing paths, and amount of input data. The performance comparison shows that JPOFH leads to up to 32% improvement in completion time compared to benchmark solutions.

The rest of the chapter organized 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 discuss relaxation of formulated problem and the proposed solution, JPOFH, for the multi-hop multi-task partial computation offloading problem. In Section 5, we have done the performance evaluation. Finally, we give the conclusion in Section 6.

5.2 Related Works

Computation offloading problem in edge computing has been well-studied. Some works such as [80] have also given performance guaranteed solution for computation offloading using an optimization framework. Most existing works study the computation offloading problem for single-hop. However, there are few recent works such as [34], [33], [35], [36], etc. that have studied multi-hop computation offloading problem. The work in [34] gave a distributed solution to make full task offloading decision to minimize the energy consumption. The work in [33] proposes an iterative algorithm for task assignment to devices in a multi-hop cooperative network. However, the work in [33] makes many assumptions in deriving the routing cost and does not consider partial offloading. The work in [35] jointly formulates the computation offloading and routing decision problem and solves using a game-theoretic approach by showing the existence of Nash equilibrium. The problem in [35] is solved for both unsliceable and sliceable tasks. The work in [36] also proposes a game-theoretic solution for multi-hop computation offloading problem; however, it considers local, edge, and cloud computing for the execution of tasks. Compared to these existing works which usually assume offloading of tasks to a single remote edge or cloud device, our work considers partial offloading to all available edge devices. Our work also considers network flow scheduling where we make decisions not only on the routing path but also start time of input data flows.

There are some works such as [21] and [81] which have also considered task allocation problem for multi-hop wireless sensor networks (WSN). However, these works do not consider partial offloading and network flow scheduling done in our work. The partial offloading problem has been addressed for single-hop in many works such as [77] and [78]. The work in [77] studied multi-user partial computation offloading problem considering both edge and cloud. They also iterative design heuristic algorithm to make the offloading decision dynamically. The work in [78] studied the partial computation offloading problem where mobile devices can partially offload the tasks to both single or multiple cloud servers with the objective of optimizing the energy consumption and latency of the application. There are some works such as [82], which also solve the computation peer offloading problem between small-cell base stations. Some works such as [29], [59], [47], [48], etc. have considered network condition while scheduling tasks. The work in [29] proposes a task scheduling framework that utilizes the underlying network scheduler to make task placement decisions. Another work [59] solves the joint reducer placement and coflow bandwidth scheduling problem. Both works in [47] and [48] considered network bandwidth to make the task scheduling decision.

The different related works in literature solve some combination of partial computation offloading, peer offloading, multi-server offloading, or network flow scheduling problem. To the best of our knowledge, there is no related work which has addressed all the concerns. This work jointly formulates the partial computation offloading and network flow scheduling problem for multi-hop collaborative edge computing. Our previous work also studied data-aware task allocation problem in collaborative edge computing [37] where dependent tasks with input data at different devices are placed in a multi-hop mesh network. The data-aware task allocation problem can be mapped to a multi-hop full computation offloading for dependent tasks. However, compared to this work, data-aware task allocation did not consider partial offloading of tasks, the release time of tasks, and decision on the routing path. We also consider additional constraints for network flow scheduling, including no flows can pass through a link at the same time, whereas the work in [37] just considered the limited link bandwidth constraint.

The formulation methodology used in this work is similar to that in [23], where the authors study the problem of offloading dependent task and give a lower bound solution. However, compared to the work in [23], we study the multi-hop multi-task partial offloading computation for independent tasks.

5.3 System Model and Problem Formulation

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

5.3.1 System Model

The system model used for this problem is same as the one used for multi-hop offloading for multiple DAG tasks. Fig 1.1 shows the system architecture of Edge Mesh, a collaborative edge computing paradigm which pushes the decision-making within the network by sharing computation resources and data between mesh network of edge devices [6]. Some of the devices in the Edge Mesh may not have enough computation capacity and can act as routers, as shown in Fig 1.1. The computation tasks are generated at different edge devices and can be offloaded to other edge devices, even at a multi-hop distance. The decision to offload and schedule tasks is made at a centralized SDN controller which collects the network information from both edge devices and routers. Although the system model assumes that edge devices are connected using a wireless network, we have not fully considered all the issues due to wireless network such as spatial and temporal variation of wireless channel conditions, interference of wireless transmission among neighbouring devices [37]. The problem formulation in this work has been done assuming a static network condition.

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

Network model: The communication network is a mesh network of edge devices, shown to Edge Mesh circle in Fig 1.1, 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 = $\{j|1 \leq j \leq M\}$, and E is the set of links connecting different devices, E = $\{e_{\mathrm{jv}}|j, v \in V\}$. 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_j, which represents the processing speed of each device j. The devices can be heterogeneous in processing speed. Each device is assumed to have a queue with positions equal to the number of tasks, i.e. N.

Any two devices are assumed to have maximum K available routing paths between them. A binary parameter W_{jvk} (1 for yes, 0 for no) is used to represents whether there is a k^{th} routing path between devices j and v. Another binary parameter Y_{kejv} (1 for yes, 0 for no) is used to represent if an edge e lies on k^{th} routing path between device j to device v. The number of hops and bandwidth of the k^{th}routing path between devices j and v is represented by H_{jvk} and R_{jvk} respectively.

Application model: An application is composed of a set of independent tasks, A = $\{i|1 \leq i \leq N\}$, where N is the total number of tasks. Each task i is associated with a computation load of CL_i and an amount of input data equal to D_i. Each task i is assumed to be generated at an edge device $z_i|z_i \in V$ at release time $Trel_i$. When a task i is partially offloaded to a device j, the input data corresponding to task i is also sent to device j. This transfer of input data is referred to as an input data flow.

Some of the assumptions made in the problem formulation are:

Some of the assumptions made in the problem formulation are:

1. The local and offloaded component of a task can be executed independently.
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. The problem is solved for a static condition where the values of different parameters are known beforehand.
4. Each device can execute one task at a time, while other tasks wait in the queue at the device.
5. Preemptive scheduling of tasks is not allowed.
6. No two flows are allowed to pass through a link at the same time to consider the interference among simultaneous wireless transmissions.

5.3.2 Problem Formulation

This section describes the constraints and formulates the problem as an optimization problem.

Constraints

There are several constraints in the problem related to task offloading and flow scheduling decision. Equation (5.1)-(5.4) represent constraints corresponding to task offloading decision. The problem considers that each task is offloaded only to a single remote device as represented by Equation (5.1).

$$\sum_{r=1}^{N} X_{ijr} * \sum_{s=1}^{N} X_{ivs} = 0, \quad \forall i \in A, \quad j \in V \backslash \{z_i\}, \quad v \in V \backslash \{z_i, j\} \tag{5.1}$$

Each task consisting of local and offloaded component is placed only once as represented by Equation (5.2).

$$\sum_{j=1}^{M} \sum_{r=1}^{N} X_{ijr} = 1, \qquad \forall i \in A \tag{5.2}$$

Since the number of positions at the queue on each device is equal to the number of tasks, each task can occupy only one position on each device. This is represented by Equation (5.3).

$$X_{ijr} * X_{ijs} = 0, \quad \forall i \in A, \quad j \in V, \quad r = 1, ...N, \quad s = 1, ...r-1, r+1, ...N \tag{5.3}$$

Furthermore, each position on each device can be occupied by only one task as represented by Equation (5.4).

$$X_{ijr} * X_{ujr} = 0, \quad \forall i \in A, \quad u \in A\backslash\{i\}, \quad j \in V, \quad r = 1, ...N \tag{5.4}$$

The constraints corresponding to flow scheduling decision are represented by Equation (5.5)-(5.8). There is an input data flow corresponding to offloading the task to a remote device as represented by Equation (5.5).

$$\sum_{r=1}^{N} X_{ijr} * (\sum_{k=1}^{K} F_{ijk} - 1) = 0, \qquad \forall i \in A \quad j \in V \tag{5.5}$$

The input data flow can start only after the task has been released as represented by Equation (5.6).

$$S_{ijk} \geq Trel_i, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.6}$$

The problem assumes that a single routing path is used to transmit each input data flow as represented by Equation (5.7).

$$\sum_{k=1}^{K} F_{ijk} \leq 1, \qquad \forall i \in A, \quad j \in V \tag{5.7}$$

As mentioned earlier, it is assumed that no two flows are allowed to pass through a link at the same time. Therefore, if there are two flows passing through a link, one of the flows is delayed until the other one is finished as represented by Equation (5.8).

$$|Y_{kez_ij} * F_{ijk} - Y_{wez_uv} * F_{uvw}| * L + (S_{ijk} - S_{uvw} - (H_{z_uvw} * \frac{D_u}{R_{z_uvw}} * \sum_{s=1}^{N} X_{uvs} * F_{uvw}))*$$
$$(S_{ijk} + (H_{z_ijk} * \frac{D_i}{R_{z_ijk}} * \sum_{r=1}^{N} X_{ijr} * F_{ijk}) - S_{uvw}) >= 0$$
$$\forall i, u \in A, \quad j, v \in V, \quad k, w = 1, ...K, \quad e \in E, \quad r = 1, ...N \tag{5.8}$$

where, L represents a large number.

There are also some constraints on the finish time of the task. Equation (5.9) represents that finish time of component of task i executed on device j at position r is greater than the start time of input data flow of task i component offloaded to device j through k^{th} routing path by the sum of computation time of component task i executed at device j and time to send the input data to execute tasks i component at device j.

$$Ft_{ijr} - S_{ijk} \geq X_{ijr} * CT_{ij} + W_{z_ijk} * H_{z_ijk} * \frac{D_i}{R_{z_ijk}} * \sum_{r=1}^{N} X_{ijr} * F_{ijk}$$
$$\forall i \in A, \quad j \in V, \quad k = 1, ...K, \quad r = 1, ...N \tag{5.9}$$

The finish time of a successive task executed on a device is at least equal to the sum of the finish time of preceding task executed on that device at a previous position and computation time of the task. This constraint is represented by Equation (5.10).

$$Ft_{ijr} >= Ft_{uj(r-1)} + X_{ijr} * CT_{ij}, \quad \forall i, u \in A, \quad j \in V, \quad r = 1, ...N \tag{5.10}$$

The total finish time of task is greater than the finish time of any component of a task as represented by Equation (5.11).

$$Tft_i >= Ft_{ijr}, \quad \forall i \in A, \quad j \in V, \quad r = 1, ...N \tag{5.11}$$

Equation (5.12) - (5.16) represent the range of each decision variable.

$$0 \leq X_{ijr} \leq 1, \quad \forall i \in A, \quad j \in V, \quad r = 1, ...N \tag{5.12}$$

$$F_{ijk} = \{0, 1\}, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.13}$$

$$S_{ijk} \geq 0, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.14}$$

$$Ft_{ijr} \geq 0, \quad \forall i \in A, \quad j \in V, \quad r = 1, ...N \tag{5.15}$$

$$Tft_i \geq 0, \quad \forall i \in A \tag{5.16}$$

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 ($Tft_i - Trel_i$) 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 multi-task partial computation offloading problem, **P1**, can be formatted as:

$$\underset{X_{ijr}, F_{ijk}, S_{ijk}, Ft_{ijr}, Tft_i}{\text{minimize}} \quad \frac{\sum_{i=1}^{N} Tft_i}{N}, \tag{5.17}$$

subject to (5.1) - (5.16)

$\forall i \in A, \quad j \in V, \quad k = 1, ...K, \quad r = 1, ...N$

This problem is NP-hard since it includes the Generalized Assignment Problem (GAP) as a special case (for a fully connected network). Since we cannot find an optimal solution in polynomial time. Therefore, we have proposed a heuristic solution JPOFH (joint partial offloading and flow scheduling heuristic).

5.4 Joint Partial Offloading and Flow Scheduling Heuristic (JPOFH)

This section first gives detail about the proposed JPOFH algorithm and then gives a lower bound solution by relaxing the formulated problem.

5.4.1 JPOFH Algorithm

JPOFH determines the execution schedule specifying the selected device, finish time, the partial offloading ratio for each task, and the start time and selected path of each input data flow. JPOFH is developed considering three principles:

1. Using a priority list of task to determine the execution schedule.
2. Scheduling each task for execution only after the previous task in the priority list is executed.
3. Determining the partial offloading ratio considering both the waiting time at the device and start time of the input data flow.

Each task in JPOFH is scheduled with the objective of minimizing the finish time, which is defined in Equation (5.18). The terms $T_{ijk_{loc}}$ and $T_{ijk_{off}}$ represent the local execution time and offloaded execution time for task i partially offloaded to device j using k^{th} routing path. $T_{ijk_{loc}}$ is defined in Equation (5.19) as the sum of computation time at the local device and waiting time to execute the task at the local device. The waiting time at the local device is represented by $Tbusy_{z_i}$ where z_i stands for the local device for task i. $T_{ijk_{off}}$ is defined in Equation (20) as the sum of computation time at the offloaded device and maximum time to send the input data to the offloaded device, and waiting time to execute the task at offloaded device. The waiting time at the offloaded device is represented by $Tbusy_j$ where j stands for device j where task i is offloaded.

$$Tft_i = \min_{j \in V, k=1,...K} \max\{T_{ijk_{loc}}, T_{ijk_{off}}\}, \quad \forall i \in A \tag{5.18}$$

$$\begin{aligned} T_{ijk_{loc}} &= (1 - x_{ijk}) * CT_{iz_i} + \max\{Tbusy_{z_i}, Trel_i\}, \\ &\forall i \in A, \quad j \in V, \quad k = 1, ...K \end{aligned} \tag{5.19}$$

$$\begin{aligned} &T_{ijk_{off}} = x_{ijk} * CT_{ij} + \max\{Tbusy_j, Trel_i, S_{ijk} + x_{ijk} * H_{z_ijk} * \frac{D_i}{R_{z_ijk}}\} \\ &\forall i \in A, \quad j \in V, \quad k = 1, ...K \end{aligned} \tag{5.20}$$

The Equation (5.22) determines the partial offloading ratio, x_{ijk}, for task i to be offloaded at device j using k^{th} routing path. The value of x_{ijk} is determined by setting local execution time equal to the upper bound of offloaded execution time defined in Equations (5.19) and (5.21) respectively. The work in [8] used a similar idea to derive the optimal solution for computation offloading using Nash equilibrium. However, [8] did not consider network flow scheduling required for multi-hop partial offloading.

$$\begin{aligned} &T_{ijk_{off}} \leq x_{ijk} * CT_{ij} + x_{ijk} * H_{z_ijk} * \frac{D_i}{R_{z_ijk}} + \max\{Tbusy_j, Trel_i, S_{ijk}\} \\ &\forall i \in A, \quad j \in V, \quad k = 1, ...K \end{aligned} \tag{5.21}$$

$$x_{ijk} = \frac{CT_{iz_i} + \max\{Tbusy_{z_i}, Trel_i\} - \max\{Tbusy_j, S_{ijk}, Trel_i\}}{CT_{iz_i} + CT_{ij} + H_{z_ijk} * \frac{D_i}{R_{z_ijk}}} \qquad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.22}$$

Both offloaded execution time and partial offloading ratio require the value of S_{ijk}. We calculate S_{ijk} using Equation (5.23) which is based on the assumption that no two flows can pass through the same link at the same time. Equation (5.23) represents that if the input data flow for task i passes through the same link in the routing path of any other input data, then it can start only after previous tasks have finished their flow.

$$S_{ijk} = \max_{u=1,...i, p \in P_{ijk}} (Trel_i, Y_{kpz_ij} * Y_{wpz_uv} * (S_{uvw} + x_{uvw} * H_{z_uvw} * \frac{D_i}{R_{z_uvw}})) \qquad \forall i \in A \backslash \{1\}, \quad j \in V, \quad k = 1, ...K \tag{5.23}$$

where, P_{ijk} denotes the edges in the k^{th} routing path for an input data flow of task i offloaded to device j and v, w are the offloaded device and routing path selected for task u.

The details of JPOFH are given in Algorithm 1. Before starting the algorithm, JPOFH initializes the value of S_{ijk} to be equal to $Trel_i$ (line 2). JPOFH starts by creating a priority list of tasks based on increasing order of rank metric defined in Equation (5.24). The rank metric considers both the release time and computation load of a task.

$$rank_i = Trel_i + \frac{CL_i}{\overline{PS}}, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.24}$$

where, $\overline{PS}$ is the average value of PS.

Starting with the first task in the priority list, JPOFH calculates the value of S_{ijk} (line 9). Then, JPOFH calculates the value of x_{ijk}, $T_{ijk_{loc}}$, and $T_{ijk_{off}}$ (line 10 -11). The selected device for offloading, j_i^*, and routing path, k_i^*, each task i is determined based on the $\min_{j\in M,k\in K}\{T_{ijk_{loc}}\}$ value (line 18), which is also equal to finish time of the task i (line 19). The algorithm returns the selected device j_i^*, finish time Tft_i, partial offloading ratio $x_{ij_i^*k_i^*}$, start time of input data flow $S_{ij_i^*k_i^*}$, and selected routing path k_i^* for each task i (line 23).

The computation complexity of JPOFH is $O(N^2 * M^2 * K)$. The computation complexity is calculated by considering the most complex operation in JPOFH algorithm, which is the calculation of S_{ijk} in line 9. There are three for loops for calculating S_{ijk} one loop each corresponding to the number of tasks (N), number of devices (M), and number of routing paths (K). The calculation of S_{ijk} in Equation (23) also requires maximum operation over different values of previous tasks in the priority list, i.e. $N-1$ in the worst case, and all the edges in the routing path, i.e. $M-1$ in the worst case. Hence, the complexity of JPOFH is $O(N^3 * M * K)$.

5.4.2 Lower Bound Solution

We have also proposed a lower bound solution by relaxing the formulated MINLP problem. The problem is non-convex due to the following three issues:

1. Bilinear terms such as $F_{ijk}. * \sum_{r=1}^{N} X_{ijr}$, $X_{ijr} * X_{ujr}$, $\sum_{r=1}^{N} X_{ijr} * \sum_{s=1}^{N} X_{ivs}$, and $X_{ijr} * X_{ijs}$, which make the problem non-convex.
2. Equation (8) is non-convex as it is of the form *quadratic expression* $\geq$ *constant*
3. Binary decision variable, F_{ijk}

The three issues in **P1** can be addressed by relaxing the problem in three stages.

Algorithm 4: Joint Partial Offloading and Flow Scheduling Heuristic (JPOFH)

Input: The set of N tasks, the network of M edge devices, and k routing paths between all device pairs

Output: The execution schedule specifying the selected device, finish time, offloading ratio for each task, and the start time and selected routing path of each input data flow

```
1  $S_{ijk} \leftarrow Trel_i, \forall i \in A, j \in V, k = 1, ...K$;
2  Create an index I of tasks in increasing order of rank metric;
3  for $t \leftarrow 1$ to $N$ do
4      $i \leftarrow I(t)$;
5      for $j \leftarrow 1$ to $M$ do
6          for $k \leftarrow 1$ to $K$ do
7              if $W_{ijk} \neq 0$ then
8                  if $t \geq 2$ then
9                      Calculate $S_{ijk}$ using Equation (23);
10                 Calculate $x_{ijk}$ using Equation (22);
11                 Calculate $T_{ijk_{loc}}$ and $T_{ijk_{off}}$ using Equations (19) and (20)
                     respectively;
12             else
13                 $x_{ijk} \leftarrow 0$;
14                 $T_{ijk_{loc}} \leftarrow CT_{iz_i} + \max\{Tbusy_{z_i}, Trel_i\}$;
15                 $T_{ijk_{off}} \leftarrow Inf$;
16         end
17     end
18     Find $j_i^*$ and $k_i^*$ based on $\min_{j \in M, k \in K}\{T_{ijk_{loc}}\}$ ;
19     $Tft_i \leftarrow \min_{j \in M, k \in K}\{T_{ijk_{loc}}\}$ ;
20     $Tbusy_i \leftarrow Tft_i$ ;
21     $Tbusy_{j_i^*} \leftarrow Tft_i$ ;
22     return $j_i^*, Tft_i, x_{ij_i^*k_i^*}, S_{ij_i^*k_i^*}, k_i^*$ ;
23 end
```

In first stage, we can relax bilinear terms in **P1** into linear terms. The bilinear term $F_{ijk}. * \sum_{r=1}^{N} X_{ijr}$ can be relaxed by using McCormick envelope. The bilinear term is replaced by a new decision variable $Z_{ijk}, \forall i \in A, j \in V, k = 1, ...K$ and by adding four additional constraints. The Equations (5.5) and (5.9) can be changed to Equations (5.25), and (5.26) respectively:

$$\sum_{k=1}^{K} Z_{ijk} = \sum_{r=1}^{N} X_{ijr}, \qquad \forall i \in A, \quad j \in V \tag{5.25}$$

$$\begin{gathered} Ft_{ijr} - S_{ijk} \geq X_{ijr} * CT_{ij} + W_{z_ijk} * H_{z_ijk} * \frac{D_i}{R_{z_ijk}} * Z_{ijk} \\ \forall i \in A, \quad j \in V, \quad k = 1, ...K, \quad r = 1, ...N \end{gathered} \tag{5.26}$$

The four additional constraints to replace bilinear term $F_{ijk}. * \sum_{r=1}^{N} X_{ijr}$ are:

$$Z_{ijk} \geq 0, \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.27}$$

$$Z_{ijk} \geq F_{ijk} + \sum_{r=1}^{N} X_{ijr} - 1, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.28}$$

$$Z_{ijk} \leq F_{ijk}, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.29}$$

$$Z_{ijk} \leq \sum_{r=1}^{N} X_{ijr}, \quad \forall i \in A, \quad j \in V \tag{5.30}$$

Same relaxation method can be applied to other bilinear terms.

In the second stage, we can relax the non-convex constraint in Equation (5.8). We first expand and change the Equation (5.13) to Equation (5.31) by considering the relaxation done previously. The absolute function in Equation (5.8) is also relaxed to a bilinear term in Equation (5.31).

$$
\begin{aligned}
&(Y_{kez_ij} * F_{ijk} * Y_{wez_uv} * F_{uvw}) * L + S_{ijk} * S_{ijk} + H_{z_ijk} * \frac{D_i}{R_{z_ijk}} * Z_{ijk} * S_{ijk} - \\
&2 * S_{ijk} * S_{uvw} - H_{z_ijk} * \frac{D_i}{R_{z_ijk}} * Z_{ijk} * S_{uvw} + S_{uvw} * S_{uvw} - (H_{z_uvw} * \\
&\frac{D_u}{R_{z_uvw}} * Z_{uvw} * S_{ijk}) - (H_{z_uvw} * \frac{D_u}{R_{z_uvw}} * Z_{uvw} * H_{z_ijk} * \\
&\frac{D_i}{R_{z_ijk}} * Z_{ijk}) + H_{z_uvw} * \frac{D_u}{R_{z_uvw}} * Z_{uvw} * S_{uvw} >= 0 \\
&\qquad \forall i, u \in A, \quad j, v \in V, \quad k, w = 1, ...K, \quad e \in E, \quad r, s = 1, ...N
\end{aligned}
\tag{5.31}
$$

Equation (5.31) has many bilinear terms that can be relaxed to linear decision variables by using McCormick envelope.

In the third stage, we relax the binary decision variable, F_{ijk}, to a continuous variable. The range of continuous decision variable F_{ijk} can be set as:

$$
0 \leq F_{ijk} \leq 1, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.32}
$$

The new relaxed problem, **P2**, can be defined with the same objective function and replacing the constraints having bilinear terms with relaxed constraints. The relaxed problem, **P2**, is a convex linear programming (LP) problem that has been solved using CVX [83] to give a lower bound solution. However, this lower bound solution is not feasible as many constraints are violated. The lower bound solution can still serve as a benchmark solution for performance evaluation.

5.5 Evaluation

We have done simulation to evaluate and compare the performance of JPOFH with other benchmark solutions. The performance evaluation has been done for two performance metrics: average completion time and running time of the algorithm.

The simulation experiments have been conducted on MacOS with 2.7 GHz Dual-Core Intel Core i5 processor and 8 GB RAM. The parameters used for simulation are in similar range to the one used previously in [23] and [73].

5.5.1 Simulation Setting

Parameters for Network Model: We generate a network of edge devices where devices are deployed randomly using a 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 [73] and [37]. 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 30%. The bandwidth of each link is selected from a normal distribution with mean 20Mbps and variance 30%.
Parameters for Application Model: The N tasks in the application model are generated at a device selected randomly. The computation load of each task is selected from a normal distribution with mean 300KCC (Kilo Clock Cycles) and variance 30% and the input data for each task transferred is selected from a normal distribution with mean 50 Kb and variance 30%. The release time of each task is selected

from a normal distribution with mean 6 ms and variance 30%. The value of mean release time is calculated as the ratio of mean computation load of tasks to mean processing speed of devices.

5.5.2 Benchmark solutions

We have compared the performance the JPOFH algorithm with four benchmark solutions.

1. Lower Bound (LB): LB is the solution described in section 5.4 for the relaxed problem. Compared to JPOFH, LB is obtained after relaxing many constraints including offloading each task to only one remote device, executing no more than one task at each device, and not allowing any two flows to pass through a link at the same time. Due to these relaxations, LB gives an infeasible solution; however, it provides a loose lower bound for performance comparison.

2. Local execution (LE): LE is obtained by executing all the tasks at the local devices where the tasks are generated. LE is easy to obtain as it does not require consideration of flow scheduling. The finish time and waiting time for local execution are defined according to Equation (5.33) and (5.34) respectively.

$$Tft_i = CT_{iz_i} + \max\{Tbusy_{z_i}, Trel_i\} \tag{5.33}$$

$$Tbusy_{z_i} = Tft_i, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.34}$$

3. Remote execution (RE): RE is obtained by executing the tasks at a remote device. Compared to JPOFH, RE only considers full offloading of tasks where the offloading is done using a greedy heuristic based on the priority list used in JPOFH. The flow scheduling is done in RE in a similar way as JPOFH algorithm.

4. Separate offloading and flow scheduling (SOFS): SOFS is obtained by separating the partial task offloading and flow scheduling problem. Compared to JPOFH, where the tasks are executed on devices based on priority order, SOFS follows the first-come-first-serve in executing the tasks. Besides, the partial offloading ratio for SOFS, shown in Equation (5.35), does not consider the start time of input data flows and waiting time at the devices. The flow scheduling is done in SOFS in a similar way as JPOFH algorithm. However, flow scheduling in SOFS only requires calculation for the selected device and routing path for offloading the task.

$$x_{ijk} = \frac{CT_{iz_i}}{CT_{iz_i} + CT_{ij} + H_{z_ijk} * \frac{D_i}{R_{z_ijk}}}, \quad \forall i \in A, \quad j \in V, \quad k = 1, ...K \tag{5.35}$$

5.5.3 Simulation Results

The default parameters used for the performance comparison are number of tasks as 5, number of devices as 10, number of routing paths between any source-destination pair as 3, and the amount of input data selected from a normal distribution with mean 50 Kb and variance 30%. The evaluation has been done under two different settings of selecting the device where each task is generated: fixed device and random device. In the fixed device setting, each task is generated on a separate device. In the simulation, we assume task i is generated on device j, where $i = j$. On the other hand, in the random device setting, the tasks can be generated randomly at any device. This implies that each device may have more than one task being generated in the random device setting. Table 5.1 and 5.2 show the performance comparison between JPOFH and benchmark solutions for fixed device and random device setting respectively. The results in both tables have been averaged for 30 iterations and the error margin is calculated for 95% confidence interval. The input values

Table 5.1 Performance Comparison for default parameters (fixed device)

Metric	**LB**	**LE**	**RE**	**SOFS**	**JPOFH**
Completion time (sec)	0.0079 ± 0.0003	0.0138 ± 0.0008	0.0128 ± 0.0005	0.0115 ± 0.0008	0.0103 ± 0.0003
Running time (sec)	9479.8 ± 199.13	8.70E-05 ± 6.80E-05	0.0087 ± 0.0018	0.0021± 0.0018	0.0101 ± 0.0024

for the application and network model in each iteration are different and generated randomly.

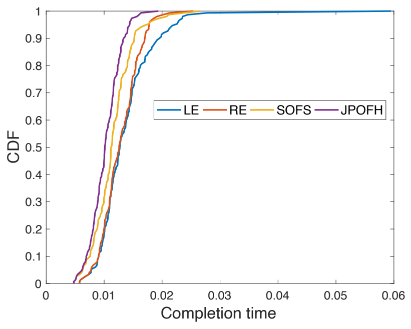

Figure 5.1 CDF plot for default parameters fixed device setting

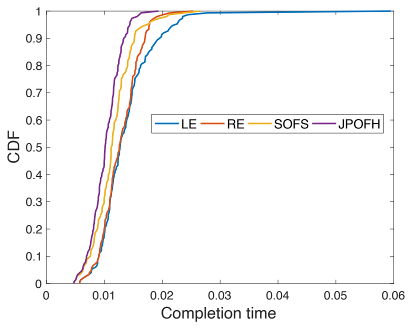

Figure 5.2 CDF plot for default parameters random device setting

We have captured the distribution of completion time of tasks and compared the performance of different solutions as shown in Fig 5.1 and 5.2. CDF plots for both fixed and random device setting shows that JPOFH leads to less completion time compared to other benchmark solutions. The details about the difference in average completion time can be observed from Table 5.1 and 5.2 that show JPOFH performs better than the three benchmark solutions (LE, RE, and SOFS) in terms of average completion time. JPOFH is around 8% to 26% better in terms of average completion time depending on benchmark solution for both fixed and random device setting. The error margin for completion is less than 5% for 95% confidence interval. The running time of JPOFH is higher than the three benchmark solutions; however, it is still within the same range as other solutions.The algorithms are proposed for offline environment where the task offloading decision is determined before the execution. Therefore, the overhead of the running time does not affect the task execution schedule. Furthermore, the value of running time changes depending on the system being used for simulation and the algorithm implementation. However, we can make some observations based on the trend in the values. The value of running time also shows large variation as we consider values from all iterations, even including outlier values. It should be noted that we have shown LB in the table for comparison; however, as mentioned previously, LB gives an infeasible solution due to relaxations of various constraints. We used Mosek solver in CVX to find the LB solution.

The rest of this section gives a detailed performance comparison by changing the values of the number of tasks, the number of devices, the number of routing paths, and the amount of input data. In order to provide better insight, we have used the fixed device setting for further evaluation unless specified otherwise. By fixing the device where each task is generated, we can study the trend in the values of average completion time and running time for a specific parameter.

Effect of number of tasks

We evaluate the effect of the number of tasks by changing the tasks from 2 to 20 as shown in Fig 5.3. We use the fixed device setting when the number of tasks is less than 10 and the random device setting after the tenth task. The average completion

Table 5.2 Performance Comparison for default parameters (random device)

Metric	**LB**	**LE**	**RE**	**SOFS**	**JPOFH**
Completion time (sec)	0.0076 ± 0.0002	0.0139 ± 0.0008	0.0124 ± 0.0004	0.0109 ± 0.0006	0.0100 ± 0.0004
Running time (sec)	9907.8 ± 162.22	1.51E-04 ± 0.0001	0.0099 ±0.0023	0.0025 ± 0.0023	0.011 ±0.0027

time increases with the number of tasks as both the waiting time at the devices and start time of network flows are increased. The difference in completion time between JPOFH and LE decreases with an increase in the number of tasks from 27.9% at 2 tasks to 16.6% at 20 tasks. This decrease in the gap can be explained as LE does not require to include waiting time at the devices and the start time of network flows which are both considered in JPOFH. However, as we increase the number of tasks to be more than the number of devices, there is some waiting time involved which slightly increases the performance difference between LE and JPOFH. RE shows a similar performance trend as LE, where the difference in the average completion time of JPOFH decreases from 20.7% at 2 tasks to 4% at 20 tasks. RE fully offloads the tasks while considering the waiting time at devices and start time of flows which leads to better performance than LE for a high number of tasks. JPOFH shows a continuous increase in performance difference with SOFS from 2% at 2 tasks to 25.5% at 20 tasks. This is because the cost of waiting time at devices and start time of network flows, which are considered separately in SOFS, is not significant when the number of devices is large compared to the number of tasks. However, as the number of tasks starts to become more than the number of devices, SOFS performs worse than JPOFH. This shows that joint decision making of partial offloading and flow scheduling, as done in JPOFH, leads to better performance in terms of average completion time compared to different benchmark solutions.

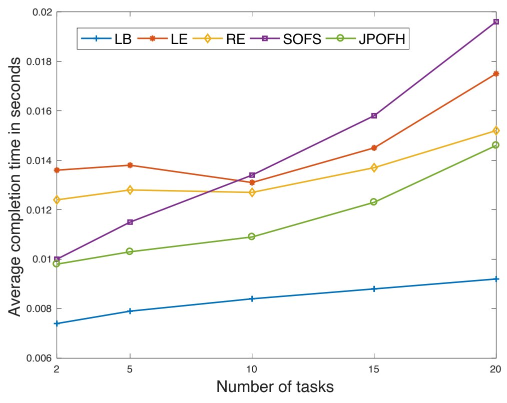

Figure 5.3 Effect of number of tasks on completion time

The effect of the number of tasks on running time of the algorithm is shown in Fig 5.4. We have shown the comparison in running time on Log10 scale as there is a huge difference in values of running between LE and LB. The running time of JPOFH is more than LE, RE, and SOFS. The running time of both RE and JPOFH increases with the number of tasks as they are similar in implementation except for the calculation of partial offloading ratio in JPOFH. LE does not show much variation in running time on increasing the number of tasks as there is no significant calculation involved. The increase in running time of SOFS is less compared to JPOFH as SOFS requires to calculate the start time of input data flow only for the selected device and routing path which decreases the running time cost significantly. The running time of LB is significantly higher than other algorithms as it includes not only the time of solving the convex optimization problem but also specifying all the possible constraint Equations. Although this time can be decreased by using a different solver or implementation approach, we can still observe the increase in running time of LB with an increase in the number of tasks.

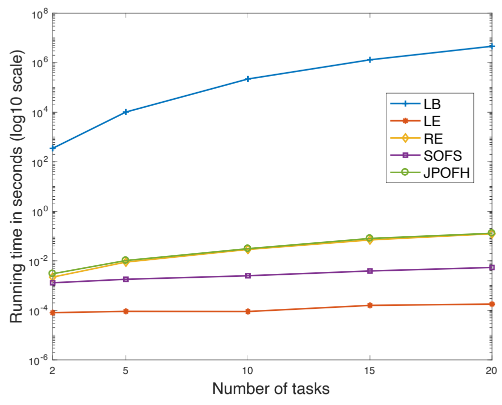

Figure 5.4 Effect of number of tasks on running time

Effect of number of devices

Fig 5.5 shows the performance comparison, in terms of average completion time, on changing the number of devices from 3 to 25. The comparison covers the full range from when the number of devices is less than the number of tasks to when the number of devices is 5 times the number of tasks. The average completion time of tasks decreases an increase in the number of devices as waiting time at the devices is reduced. The performance difference between JPOFH and LE increases from 12.2% at 3 devices to 26.6% at 25 devices. JPOFH performs even better when the number of devices is more than the number of tasks as there are more options of partially offloading the tasks leading to a decrease in waiting time. In contrast, LE does not leverage the resources available on other devices. JPOFH also shows an increase in performance difference between RE from 9.1% at 3 devices to 19.7% at 25 devices. Although RE can leverage resources on other devices, partially offloading the tasks in JPOFH leads to better performance as the different components of the tasks can be executed simultaneously. There is a decrease in performance difference between JPOFH and SOFS from 15% at 3 devices to 6% at 25 devices. As explained earlier, the cost of waiting time at the devices and start time of network flows is very low when the number of devices is significantly high compared to the number of tasks. However, JPOFH still performs better than SOFS, even for a large number of devices.

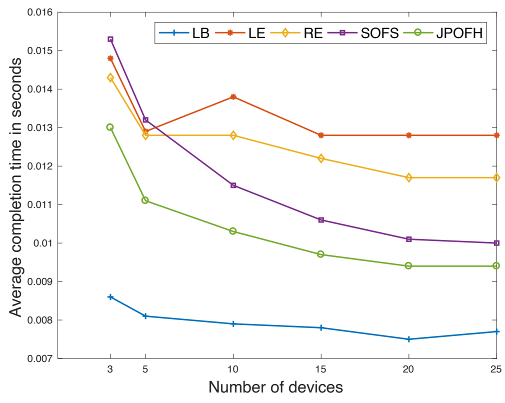

Figure 5.5 Effect of number of devices on completion time

The performance comparison, in terms of running time of the algorithm, is shown in Fig 5.6. The running time of all the algorithms, expect LE, increases with an increase in the number of devices. SOFS also does not show significant variance in running time as it does not have to calculate start time of input data flow for all devices and routing paths which is the most computation-intensive part of JPOFH algorithm.

Effect of number of routing paths

Fig 5.7 shows the performance comparison, in terms of average completion time, on changing the number of routing paths between any source-destination pair from 1

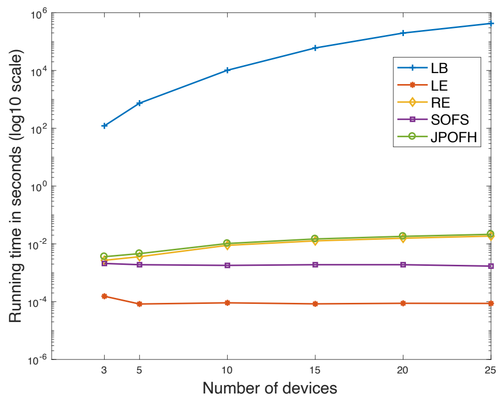

Figure 5.6 Effect of number of devices on running time

to 5. We find K shortest paths for all source-destination pairs using Yen's algorithm [84]. JPOFH uses one of the routing paths between source and destination device to send the input data for a task. Although we initially expected to have an improvement in completion time for an increased number of routing paths, however, we find that there is no significant change in the value of average completion time for a different number of routing paths available between any two devices. This is because we use predetermined routing paths with increasing value of communication delay for random network topology. Besides, we measure the performance for a specific case of the number of tasks and number of devices. In order to observe the effect of the number of routing path, we need to see the performance for specific network topology with increased network traffic. However, our aim in this work is to study the performance of the proposed solution for random network topologies and not for a specific network topology usually considered for data centers. The evaluation for specific network topologies in data center networks can be considered in future work.

The performance comparison, in terms of running time of algorithm, is shown

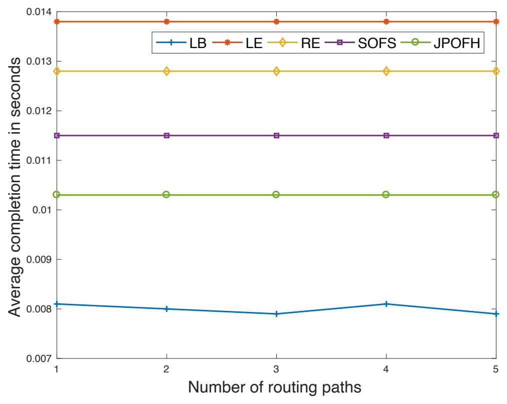

Figure 5.7 Effect of number of routing paths on completion time

in Fig Fig 5.8. We observe an increase in the running time of all algorithms, except LE for reasons explained previously, on increasing the number of routing paths. SOFS does not show significant variance in running time as it considers a selected device and routing path to calculate the start time of input data flow.

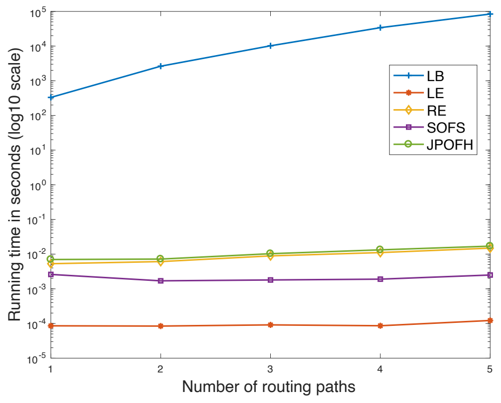

Figure 5.8 Effect of number of routing paths on running time

Effect of amount of input data

Fig 5.9 shows the performance comparison, in terms of average completion time, on changing the mean of normal distribution used to select the amount of input data from 5 Kb to 200 Kb. The change in the amount of input data reflects the change in the ratio of communication to computation cost. The comparison covers the full range from computation-intensive tasks (5 Kb input data) to communication-intensive tasks (200 Kb input data). We observe an increase in average completion time on increasing the amount of input data as communication cost is increased. The performance difference between JPOFH and LE decreases from 31.2% at average input data of 5 Kb to 16.7% at average input data of 200 Kb. This is because the increase in input data leads to an increase in communication cost only for JPOFH as tasks are executed on local devices in LE. JPOFH also shows a decrease in performance difference between RE from 18.1% at average input data of 5 Kb to 14.2% at average input data of 200 Kb. This is because the benefit of executing the partitioned components of tasks simultaneously in JPOFH, compared to RE, is limited by high communication cost in case of high average input data. There is also a decrease in performance difference between JPOFH and SOFS from 20.8% at average input data of 5 Kb to 4.2% at average input data of 200 Kb. Compared to SOFS, JPOFH considers the order of tasks for execution at the devices based on their priority which leads to better performance when average input data is low. However, the effect is ordering the tasks is limited by high communication cost in case of high average input data leading to a decrease in performance difference between JPOFH and SOFS.

The performance comparison, in terms of running time of algorithm, is shown in Fig Fig 5.10. There is no significant difference in running time of algorithms on changing the amount of input data as computation complexity of both JPOFH and

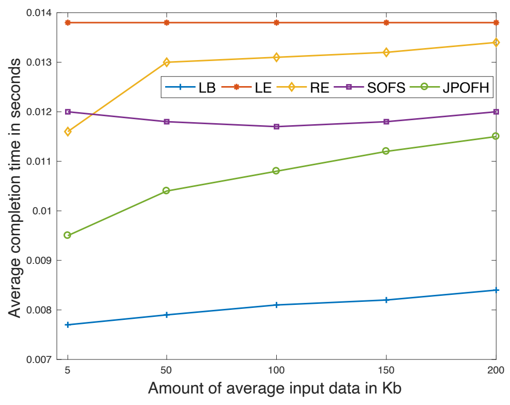

Figure 5.9 Effect of amount of input data on completion time

benchmark solutions is independent of the amount of input data.

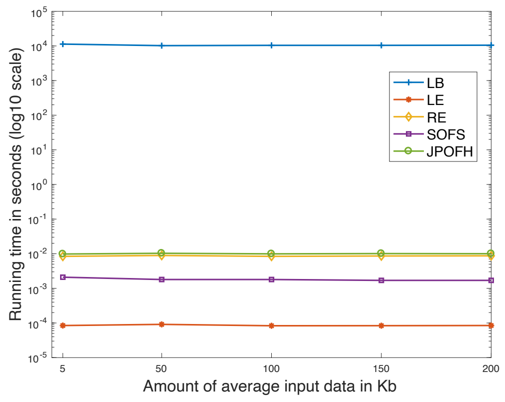

Figure 5.10 Effect of number of routing paths on running time

5.6 Conclusion

In this chapter, we study the multi-hop multi-task partial offloading problem in collaborative edge computing where heterogeneous independent tasks generated at different heterogeneous devices at different release time are partially offloaded to a remote device with the objective of minimizing the average completion time of all tasks. We need to make a decision considering both partial offloading and network flow scheduling as tasks can be offloaded to a device which is multi-hop away. We formulate the problem as an MINLP optimization problem which is proven to be NP-hard. Therefore, we propose a JPOFH algorithm which jointly solves the partial offloading and network flow scheduling problem. The MINLP problem is also relaxed to an LP problem using McCormick envelope and the solution to the relaxed problem acts as lower bound for performance comparison. We have done simulation experiments to evaluate the efficacy of the JPOFH by comparing it against benchmark solutions, including local execution, remote execution, and separate offloading and flow scheduling. We have done a comprehensive performance comparison of JPOFH with benchmark solutions by varying different input parameters, including the number of tasks, number of devices, number of routing paths, and the amount of input data. Performance comparison shows that JPOFH leads to up to 32% improvement in average completion time compared to benchmark solutions.

We have solved this problem assuming a static environment. The problem can be extended further considering the dynamics in network bandwidth and device resources. Further, we consider the wireless interference in the problem by assuming that any two network flows cannot pass through a network link at the same time. The wireless interference can be more explicitly modelled in the problem. We have given a loose lower bound solution by relaxing the original problem but due since the problem is NP-hard, we could not provide more theoretical insight. Another direction for this problem is to mathematically formulate the problem with some assumptions such that more theoretical analysis could be provided.

Chapter 6

Conclusion and Future Work

In this thesis, we propose Edge Mesh, an abstraction of CEC, that leverages a mesh network architecture of edge devices to enable scalable connectivity and distributed decision-making within the network. We investigate and address the issues in solving the problem of task partitioning and offloading in collaborative edge computing environments. The first issue is the task to be executed on an edge device can require data stored at other edge devices. This requires task offloading decisions to be aware of data placement. The second issue is the transfer of input data and the data from dependent tasks leads to multiple network flows contending for the bandwidth resources. Task offloading without considering network flow scheduling could lead to network congestion and inefficient performance in terms of completion time. The third issue is dependency among tasks and network flows which makes the problem difficult to mathematically formulate and solve. Furthermore, the different application models have a difference in terms of dependency, number of tasks, and release time, requiring difference in the problem formulation. The proposed solution for task partitioning and offloading in collaborative edge computing should jointly consider the data placement and transfer to achieve better performance.

The different works studied in Chapter 3, 5, and 4 are centered around addressing the above three issues. The data-aware task allocation problem studied in Chapter 3 considers the input data placement for each task and the network flow scheduling jointly to make task allocation decisions. This work is solved for a single application task graph where we have dependent tasks and each task may require input data from multiple different sources. The work in Chapter 4 solved the problem of joint task offloading and network flow scheduling for multiple DAG tasks. Another example is work in Chapter 5 which solved the partial offloading problem for independent tasks while also considering network flow scheduling due to the transfer of input data corresponding to the task. The three works share a common network model where we have a static mesh network of edge devices and a centralized controller that makes the partitioning and offloading decisions. There are also some common challenges in solving the problems studied in three problems. One major challenge in solving the different problems is that data transmission cost is not static as it is dependent on task allocation decision.

The three works studied in this thesis, however, differ in the problem formulation and solution due to the difference in the application model. We solve the data-aware task allocation problem in Chapter 3 by proposing a heuristic solution MSGA, where we first provide an initial solution without considering network congestion and then make an adjustment on task allocation and start time of network flows to avoid network flow conflicts. However, we cannot use the same approach for multiple DAG tasks offloading in Chapter 4 as there would be many more network flows which belong to different task graphs and we also consider the difference in release time for each task graph. Therefore, instead of solving the problem in stages as in MSGA, we propose JDOFH where we solve both the offloading and network flow scheduling for each subtask within a task at the same step. Compared to the other two works, the work in Chapter 5 considers partial offloading of independent heterogeneous tasks. We propose JPOFH that calculates the partial offloading ratio of each task by considering the waiting time at the device and start time of input data flows.

We have made some assumptions in solving the different problems and there are many issues that can be considered for future work. The dynamics of wireless network and change in workload on the devices leads to change in available network and device resources. The problems have been solved assuming a static network and not explicitly considering the effect of wireless interference. We only consider the offloading among edge devices while not considering available cloud resources and the change in incoming data from end devices. The problems can also be extended by considering other application models too such as multiple DAG with tasks with multiple input data sources. We have also not considered network and device failures while solving the task offloading problem. Another future direction is to jointly consider application adaption with the task offloading decision. For example, changing the quality of video based on the network condition or compress or change the parameters of the deep learning model. We have proposed the solutions assuming a centralized controller that has overall global knowledge. We should also look into proposing distributed solutions for task partitioning and offloading in collaborative edge computing environments. Another important issue is that the evaluation of the proposed solutions is done using simulation experiments. A real-world prototype should be used to illustrate the efficacy of the proposed solutions. We also need to consider the integration of collaborative edge computing with blockchain, 5G, and other emerging technologies.

We are currently working on some of these issues. We have done some preliminary work on developing the prototype for CEC to illustrate the sharing of computation and data resources among edge devices and the proposed task partitioning and offloading solutions. We have developed a demo prototype using the docker container platform. We develop a multi-node cluster using edge devices such as Raspberry Pi 3 Model B+ and Nvidia Jetson TX2. The docker container platform enables scalable connectivity in the developed prototype. The devices and application services used in the prototype can be scaled up/down depending on network condition and resource consumption of devices. We have done work in illustrating sharing of computation resources and data among edge devices. We illustrate the sharing of computation resources using a model partitioning example, where part of the model for object detection is executed on Nvidia TX2 and rest on the PC, which acts as a server. We also develop a demo for showing the data sharing among edge devices using multi-camera multi-person re-identification example. In this demo, the output features from detecting a person are shared with other devices to detect the person on another camera in real-time. We have tried tools such as Apache Airflow and Argo to deploy example applications with dependent tasks in a multi-node cluster using the default scheduler in Kubernetes. We are currently working on modifying the default scheduler in Kubernetes to show the efficacy of our proposed solutions.

Another ongoing work is on integrating edge computing with blockchain. In particular, we are working on a problem of modelling relationship between users submitting the transactions, miners responsible for packing the transactions in the block, and edge service providers providing the computation resources for miners. Compared to the existing works, this problem also considers the effect of the number of transactions and transaction cost on mining offloading decision.

Bibliography

[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] L. U. Khan, I. Yaqoob, N. H. Tran, S. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge computing enabled smart cities: A comprehensive survey," *IEEE Internet of Things Journal*, 2020.

[6] 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.

[7] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[8] D. Nowak, T. Mahn, H. Al-Shatri, A. Schwartz, and A. Klein, "A generalized nash game for mobile edge computation offloading," in *2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. IEEE, 2018, pp. 95–102.

[9] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, pp. 23–32, 2013.

[10] S. Ray, Y. Jin, and A. Raychowdhury, "The changing computing paradigm with internet of things: A tutorial introduction," *IEEE Design & Test*, vol. 33, no. 2, pp. 76–96, 2016.

[11] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[12] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[13] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016.

[14] A. Munir, P. Kansakar, and S. U. Khan, "Ifciot: Integrated fog cloud iot: A novel architectural paradigm for the future internet of things." *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 74–82, 2017.

[15] 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.

[16] 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.

[17] X. Chen, Z. Zhou, W. Wu, D. Wu, and J. Zhang, "Socially-motivated cooperative mobile edge computing," *IEEE Network*, vol. 32, no. 6, pp. 177–183, 2018.

[18] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing-a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[19] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[20] 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.

[21] 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, pp. 297–310, 2007.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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)*, 2016, pp. 81–97.

[28] 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.

[29] 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.

[30] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[31] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE transactions on parallel and distributed systems*, vol. 26, no. 12, pp. 3317–3329, 2014.

[32] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee," *IEEE Transactions on Communications*, vol. 66, no. 4, pp. 1594–1608, 2018.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

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

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multi-hop offloading of multiple dag tasks in collaborative edge computing," *IEEE Internet of Things Journal*, 2020.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] T. De Schepper, S. Latré, and J. Famaey, “A transparent load balancing algo-

rithm for heterogeneous local area networks," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 160–168.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] 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.

[75] 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.

[76] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multi-hop multi-task partial computation offloading in collaborative edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1133–1145, 2020.

[77] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things," *IEEE Internet of Things Journal*, 2018.

[78] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[79] B. Li, M. He, W. Wu, A. Sangaiah, and G. Jeon, "Computation offloading algorithm for arbitrarily divisible applications in mobile edge computing environments: An ocr case," *Sustainability*, vol. 10, no. 5, p. 1611, 2018.

[80] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, "Performance guaranteed computation offloading for mobile-edge cloud computing," *IEEE Wireless Communications Letters*, vol. 6, no. 6, pp. 774–777, 2017.

[81] 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, 2011.

[82] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[83] M. Grant, S. Boyd, and Y. Ye, "Cvx: Matlab software for disciplined convex programming," 2008.

[84] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.