An Approach to QoS-based Task Distribution in Edge Computing Networks for IoT Applications

by

Yaozhong Song

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved October 2018 by the
Graduate Supervisory Committee:
Sik-Sang Yau, Chair
Dijiang Huang
Hessam S. Sarjoughian
Yanchao Zhang

ARIZONA STATE UNIVERSITY
December 2018

ABSTRACT

Internet of Things (IoT) is emerging as part of the infrastructures for advancing a large variety of applications involving connections of many intelligent devices, leading to smart communities. Due to the severe limitation of the computing resources of IoT devices, it is common to offload tasks of various applications requiring substantial computing resources to computing systems with sufficient computing resources, such as servers, cloud systems, and/or data centers for processing. However, this offloading method suffers from both high latency and network congestion in the IoT infrastructures.

Recently edge computing has emerged to reduce the negative impacts of tasks offloading to remote computing systems. As edge computing is in close proximity to IoT devices, it can reduce the latency of task offloading and reduce network congestion. Yet, edge computing has its drawbacks, such as the limited computing resources of some edge computing devices and the unbalanced loads among these devices. In order to effectively explore the potential of edge computing to support IoT applications, it is necessary to have efficient task management and load balancing in edge computing networks.

In this dissertation research, an approach is presented to periodically distributing tasks within the edge computing network while satisfying the quality-of-service (QoS) requirements of tasks. The QoS requirements include task completion deadline and security requirement. The approach aims to maximize the number of tasks that can be accommodated in the edge computing network, with consideration of tasks' priorities. The goal is achieved through the joint optimization of the computing resource allocation and network bandwidth provisioning. Evaluation results show the improvement of the

approach in increasing the number of tasks that can be accommodated in the edge computing network and the efficiency in resource utilization.

Index terms edge computing, Internet of Things, load balancing, network flow, optimization, quality-of-service, and task distribution

DEDICATION

To My Parents

ACKNOWLEDGMENTS

First, I would like to thank my Ph.D. advisor, Professor Stephen S. Yau. He supported me to explore my research ideas and gave me guidance during my Ph.D. study. I would like also to thank my committee members Professors Partha Dasgupta, Dijiang Huang, Hessam S. Sarjoughian, and Yanchao Zhang for their guidance and support on my research.

Special thanks to Professor Guoliang Xue, and his students Ruozhou Yu, and Xiang Zhang, for their helpful collaboration and discussion. I would like also to thank Professor Arun Balaji Buduru, Professor Ziming Zhao, Sayantan Guha, Tamalika Mukherjee in the Information Assurance Center at Arizona State University, Dr. Tianyi Xing, Dr. Yuan Wang, Jia Yu, and Adel Alshamrani, in the School of Computing, Informatics, and Decision System Engineering at Arizona State University, for their inspiring discussions.

Finally, and most importantly, I would like to thank my family. I would like to especially thank my parents, for their 29-year great spiritual and material support.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

1.1 Overview

With its capability of interconnecting a large number of various intelligent devices across wide geographical areas, IoT has become part of the infrastructures for many advanced applications leading to smart cities and other connected communities. This trend has inspired the development of a large variety of applications with connected intelligent devices, such as healthcare applications based on wearable IoT devices, public safety applications based on surveillance IoT devices, and smart home applications based on connected IoT appliances. IoT applications and devices have been deployed to many scenarios, including healthcare environments, smart city, intelligent transportation, smart manufacturing, smart grids, etc [63].

However, despite the rapid progress of IoT-related technologies, a major bottleneck of IoT applications is the limited computing resources available of each IoT device, including CPU, storage, network bandwidth, etc. In addition, as most IoT devices are powered by batteries, energy supply is also a limitation for their operations [27]. For IoT devices, especially mobile IoT devices, factors such as weight, size, battery life, and heat dissipation are often more important than computing resources. The improvement of these factors often limits the improvement of the computing resources of IoT devices. Hence, although technologies advance, the computing resources of IoT devices will always be limited [51].

In addition, IoT applications become more sophisticated, involve more computation, require more storage and bandwidth, and consume more energy. For example, nowadays there are surveillance cameras that run computation heavy image/video processing, computer vision and machine learning programs for human/object detection. Such programs require significant amounts of computing resources. Hence, the proliferation of IoT applications further exacerbates the situation that most IoT devices are lack of resources.

A common way to overcome the limited computing resources of IoT devices in various applications is to offload some tasks of IoT applications to resourceful servers [12] [27] [28] [35] [46] [68]. The most widely used "resourceful server" is the cloud computing system, such as Amazon Web Services and Microsoft Azure. Task offloading to cloud computing systems has two major benefits. One is to improve the performance of IoT applications, as the computing resources in cloud computing systems are usually much more powerful than those in IoT devices. The other major benefit is to reduce the energy consumption for IoT devices. As the power consuming computations are offloaded, the energy consumption in IoT devices can be largely reduced, which is an important factor to increase user experiences [23] [24] [25] [27] [46]. The amount of energy can be saved for IoT devices by task offloading is related to factors such as network bandwidth and the amount of data needs to be transmitted to cloud computing systems. Researches have been conducted on quantifying the benefit of task offloading [22] [66] [67].

However, this offloading method has two major drawbacks [18] [51] [77]. One is the long and unpredictable latency introduced by data transmission through Wide Area Networks (WAN), which is especially harmful to latency-sensitive IoT applications, such

as healthcare applications and public safety applications. This long and unpredictable WAN latency brings very negative user experiences. The other major drawback is network congestion. IoT devices generate huge amounts of data, and if all the data is poured into the Internet, it will easily cause network congestion. Besides the congestion in computer networks, the huge amount of data also overloads of cloud computing systems. Cloud computing systems process not only data generated by people but also unprecedented amounts of data generated by IoT devices and other machines. The trend of global digitization brings huge effects of IoT data on global cloud computing systems [6]. According to the Cisco Global Cloud Index [6], by 2021, there will be 850 zettabytes data generated by all people, IoT devices per year, and approximately 10 percent is useful and needs to be processed. At the same time, the capacity of cloud computing systems traffic will only reach 21 zettabytes per year. Hence, sending all the IoT data to cloud computing systems is not a practical solution, as the cloud computing systems will be severely overloaded. Such a severe overload of cloud computing systems will exacerbate the first drawback, making the latency even more unpredictable and unacceptable.

The drawbacks of task offloading to remote computing systems motivate the recent development of edge computing [4] [5] [51] [57] [60], which uses network devices with large computing resources, such as cloudlets, computing-enabled routers or switches, and computing-enabled base stations at the edge of computing networks, to support IoT applications. The network devices that support edge computing are called edge computing devices, or edge nodes [4] [51]. In this dissertation, edge computing device and edge computing node will be used interchangeably, as there are with the same meaning.

Due to the proximity to IoT devices, edge computing networks can decrease the latency of IoT applications and make larger bandwidth available for interconnecting IoT devices compared to remote cloud computing systems. Usually, the distances between edge computing networks and IoT devices are negligible compared to the distances between remote cloud computing systems and IoT devices. Edge computing networks can be set up in an office building, in a community, or in a smart city. The network connections between edge computing networks and IoT devices are mostly Local Area Networks (LAN). In contrast, cloud computing data centers usually locate in another city, or in another state, which can be hundreds of miles away. The network connections between cloud computing systems and IoT devices are both LAN and WAN. By avoiding data transmission through WAN, edge computing can largely reduce the overhead of data transmission, which, in turn, largely reduces the latency for its connected IoT applications. Another benefit of edge computing is to reduce network congestion, as mentioned in the Cisco Global Cloud Index [6], over 75% of data generated by people, IoT devices cannot be digested by cloud computing systems. Edge computing networks can help to digest part of the 75% data, which will reduce the network congestion problem. In addition, edge computing networks can also relieve the power constraint of battery-powered IoT devices [52].

Yet, task offloading to edge computing networks has its own problems. The major problem is the unbalanced workload among edge computing devices, which may result in network congestion in edge computing networks. There are two reasons for this problem: One is that the computing resources of single edge computing devices are not always enough to process all the IoT tasks it received [9], especially for computation-heavy tasks, such as computer vision tasks, and machine learning tasks. The other is that the workloads

on different edge computing devices are naturally unbalanced. Edge computing devices connecting more IoT devices usually have more workload. In addition, edge computing devices are heterogeneous. Some edge computing devices are resource-rich, while some are resource-poor. Hence, it is common that some edge computing devices are overloaded, while nearby edge computing devices may have normal workloads, or even be idle. If the workload in edge computing networks is not well managed, the QoS of both IoT applications and edge computing networks cannot be guaranteed. Enabling resource sharing and task offloading among edge computing devices has been used for addressing this problem [18] [44] [47] [49] [50] [52] [77].

In this dissertation, an effective approach to distributing tasks within the edge computing network will be presented. The approach is to periodically distribute batches of tasks among edge computing devices, which are not limited to tasks' access edge computing devices so that the number of tasks that can be accommodated by the edge computing network can be maximized. Our approach considers the priorities of tasks as weighting factors in the optimization objective. The task distribution solution generated by our approach is ensured to satisfy all the accommodated tasks' QoS specifications required by their IoT applications. The QoS specifications refer to tasks’ completion deadlines and security requirements [46]. The benefits of our approach include balancing the workload among edge computing devices and reducing network congestion within edge computing networks.

1.2 Organization of Dissertation

The dissertation is organized as follows: the current state of the art related to this dissertation research will be presented in Chapter 2. Our task distribution approach will be presented in Chapter 3. The simulations results will be presented in Chapter 4. The conclusion of this dissertation work and future research directions will be presented in Chapter 5.

Chapter 2

CURRENT STATE OF ART

2.1 Edge Computing Use Cases

The concept of edge computing is first introduced in [5] [51]. “Fog computing” has also been used to refer to edge computing [5] [18] [25] [44] [47] [60] [75] [77]. There are many use cases of edge computing, such as smart cities [14] [31] [52] [56] [73] and intelligent healthcare systems [18] [52] [53] [59].

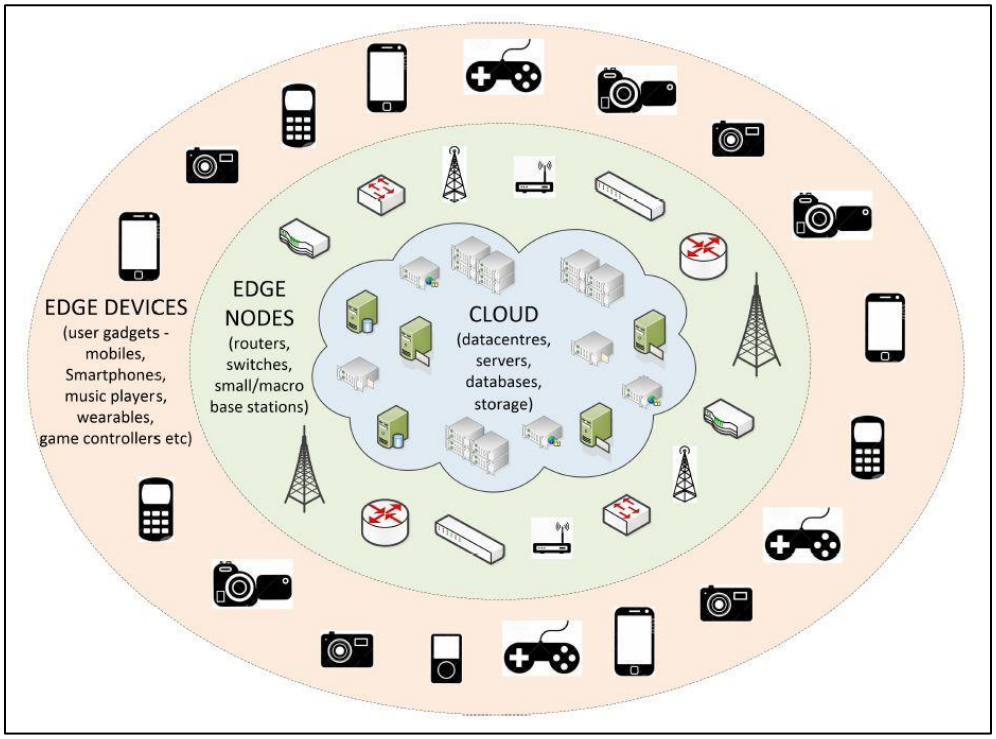

Figure 1 IoT devices, Edge Computing and Cloud Computing [61]

Figure 1 shows the three-level architecture of edge computing [61]: IoT devices as the bottom layer, edge computing servers as the middle layer and cloud data centers as the top layer. The IoT devices at the bottom layer can be smartphones, or sensors on smart cars, or augmented reality/virtual reality glasses, or connected surveillance camera. The edge computing servers can be network devices, such as network switches, routers, cellular base stations, or "Cloudlet" [51]. The cloud data center can be Amazon AWS or Microsoft Azure.

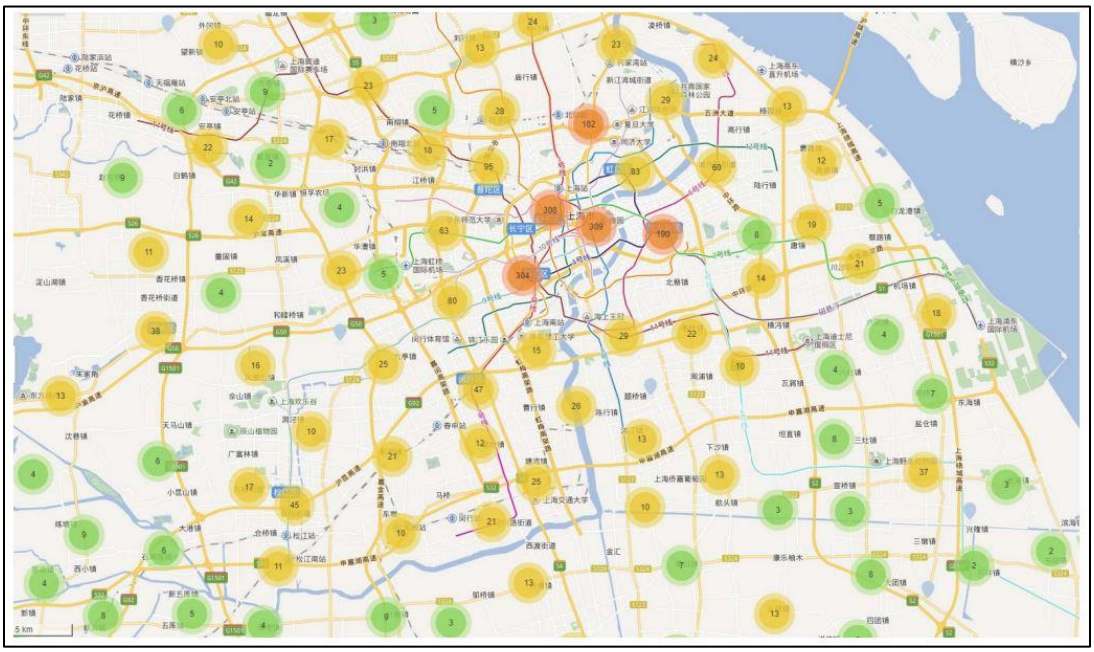

Figure 2 Distribution of 3233 Base Stations in Shanghai [31]

Li and Wang [31] introduce edge computing in a smart city scenario. Their edge computing servers are deployed in base stations in the city. Figure 2 shows the distribution of 3233 base stations in Shanghai and the authors propose the deployment scheme for edge computing servers among these base stations. This is a smart city use case where edge computing servers can be deployed in these base stations and IoT devices can connect to edge computing servers to run IoT applications.

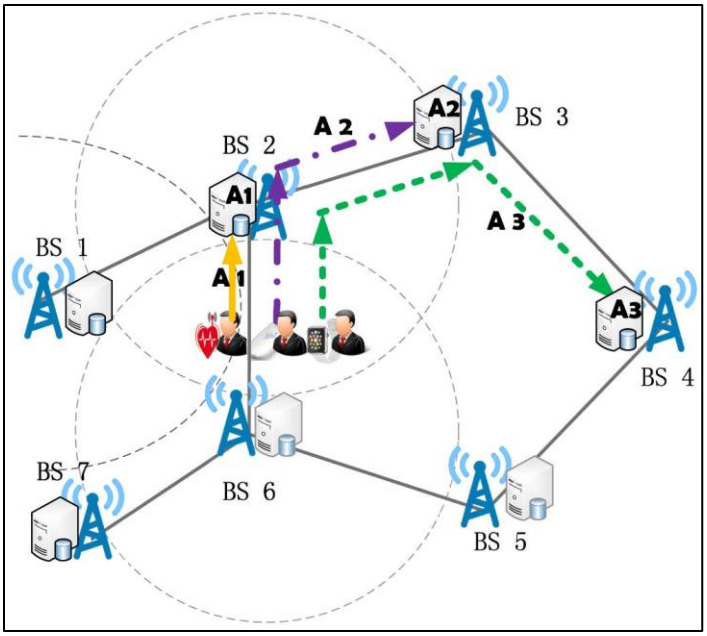

Figure 3 An overview of Fog Supported Medical Cyber-Physical Systems [18]

Gu, Zeng, Guo, Barnawi, and Xiang [18] introduce fog computing supported medical cyber-physical system (FC-MCPS), which is an edge computing enabled health care system. The FC-MCPS can be depicted in Figure 3. This work discusses the enabling of edge computing to the MCPS discussed in [30]. [30] defines the virtual medical device (VMD), which connects multiple medical devices, runs multiple clinic algorithms, and defines how the connected medical devices should interact with each other as specified in the clinic algorithms. VMDs can be deployed on edge computing servers, to get access to more resources to support more medical devices and more complex medical applications. As medical applications are mostly highly sensitive to latency, edge computing is a better choice than cloud computing in this scenario, as edge computing usually has much less latency compared to cloud computing [18] [51] [77].

2.2 Computation Offloading

The prerequisite of task distribution in edge computing networks is computation offloading. Hence, I will first discuss the researches on computation offloading for IoT devices to both cloud computing systems and edge computing networks. Computation offloading for IoT devices, which includes mobile devices, has attracted significant research interests since the booming development of IoT devices [27]. There have been researches on both how to make computation offloading decisions and how to offload computation.

Researches on how to make computation offloading decisions focus on quantifying and comparing the performance and energy consumption of conducting computation for both locally on IoT devices and offloaded to remote resourceful servers [2] [7] [19] [20] [21] [22] [32] [33] [34] [42] [45] [48] [62] [66] [67]. Researches on how to offload computation generally fall into two categories, static computation offloading and dynamic computation offloading [27]. Static computation offloading means that programs are partitioned during the program development phase. Static partition has lower overhead during execution but less accuracy compared to dynamic offloading. Dynamic offloading offloads computation during run-time, and has higher overhead, but more accuracy [19] [22] [24] [48] [66]. Dynamic offloading has higher overhead because it needs to monitor the run-time conditions, for example, the network bandwidths. There are also researches on what part(s) of the computation of an application should be offloaded. Different program partition algorithms are proposed in [7] [11] [22] [32] [33] [34] [42] [43] [58] [62] [67].

Current researches on computation offloading mainly use cloud computing systems as the offloading target. As edge computing networks is indeed a form of cloud computing system at the edge of computer networks, most of these computation offloading techniques are applicable to edge computing networks.

2.3 Task Distribution in Edge Computing

Task distribution in edge computing networks has been studied in [3] [17] [18] [36] [44] [49] [50] [65] [77]. These works are differentiated from each other in three major aspects, which are the system model, the optimization objective, and the perspective of their optimizations. In the following paragraphs, I will discuss current research works on task distribution in edge computing networks in these three aspects.

For the system model, it includes the service model for tasks and the infrastructure model for edge computing networks. Different works are usually based on differently built system models for edge computing networks and IoT tasks. Zeng, Gu, Guo, Cheng, and Yu [77] present a joint optimization of task scheduling and image placement to minimize the average task completion time for fog computing supported software-defined embedded system. First, this work considers the situation that the data of tasks is not generated and sent from IoT devices, but stored in storage servers. It also considers the I/O interruptions (e.g. page faults) of such storage servers. Hence, the data flow in this work is from storage servers to computation servers. Secondly, this work also assumes that all tasks will be completed either at devices sides or on edge computing nodes. Gu, Zeng, Guo, Barnawi, and Xiang [18] present a joint optimization of user association, virtual machine deployment,

and task distribution to minimize the cost of operation for fog computing supported medical cyber-physical system. This work considers the association of devices and access edge nodes as a variable for the optimization problem. Oueis, Strinati, and Barbarossa [44] present an approach to load distribution for small cell cloud computing and a joint optimization of computational and radio resources to minimize the power consumption for dynamically formed small cell cloud systems. This work assumes that the data size of each task is proportional to each task's workload. Chen, Jiao, Li, and Fu [9] present an approach to optimize the multi-user computation offloading in multi-channel wireless contention environments. This work considers the association of devices and access edge nodes as a variable for the optimization problem, which is similar as in [18]. Munoz, Iserte, and Vidal [41] present a framework to jointly optimize the computation and communication resources to analyze the tradeoffs between energy consumption and latency. This work allows task partition, so part of a task can be executed locally on IoT devices and the other part can be offloaded.

For the optimization objective, it can be minimizing the average task completion time, minimizing the sum of operation cost to complete tasks, or minimizing the energy consumption for IoT devices, etc. Zeng, Gu, Guo, Cheng, and Yu [77] aim to minimize the average task completion time with a joint optimization of task scheduling and storage image placement. Gu, Zeng, Guo, Barnawi, and Xiang [18] aim to minimize the cost of operation of tasks is based on this consideration, as optimizing the association may reduce the operation cost of task execution. Oueis, Strinati, and Barbarossa [44] aim to minimize the power consumption for each user. Sardellitti, Barbarossa, and Scutari [50] present a joint optimization approach to minimize the energy consumption at the mobile terminal

side. Barbarsossa, Sardellitti, and Lorenzo [3] present a similar joint optimization approach for mobile application offloading scenario with the same optimization objective. Both works optimize the computation resource allocation and communication resource allocation with consideration of latency constraints. Sardellitti, Scutari, and Barbarossa [49] present a joint optimization of radio and computational resources for mobile edge computing. First, this work does not take storage limitation as a constraint when formatting the optimization problem. Secondly, this work does not consider the compatibility between virtual machines and tasks. Sardellitti, Scutari, and Barbarossa [49] aim to minimize the overall users' energy consumption at the mobile terminal side. Wen, Zhang, and Luo [65] present an approach to optimize both the computation and communication, with the objective to minimize the energy consumption for IoT devices. Gedawy, Habak, Harras, and Hamdi [17] present an approach to optimize the task scheduling in edge femtocloud, which is defined as a cluster of heterogeneous mobile and IoT devices. The optimization objective of this work is to maximize the computational throughput of the edge femtocloud while maintaining the energy consumption constraints. Li and Wang [31] present an approach to optimize the edge servers' placements to reduce energy consumption and improve resources utilization of edge servers. Although this work does not involve task distribution, its approach to distributing edge servers is related to task distribution in edge computing networks. This work assumes that the edge server placement is a variable, which is a major difference from most existing researches. The perspective of the optimization objective to minimize the energy consumption is edge server, but not IoT device, which differentiates this work from other works aiming to optimize energy consumption [44] [49] [65]. Nimmagadda, Kumar, Lu, and Lee [42] compare the performance of executing tasks

onboard of a robot and the performance of executing the same tasks offloaded to a Linux server. According to their system setup, the server is a regular standalone Linux server at the network edge. Hence, this work belongs to the scenario of edge computing, but not cloud computing. As this work considers a scenario of one single edge computing node, it does not involve task distribution or load balancing of edge computing devices. The goal is to study the advantages and disadvantages of computation offloading edge devices. Zhao, Guo, Zhang, and Li [82] propose a multi-objective task scheduling algorithms, which was to minimize the overhead of task executions and minimize the number of failed tasks.

For the perspective of optimization, it can be the perspective of IoT devices, the perspective of IoT applications, or the perspective of edge computing networks. Gu, Zeng, Guo, Barnawi, and Xiang [18] have the perspective of IoT tasks as they aim to minimize the average task completion time. Oueis, Strinati, and Barbarossa [44] and Sardellitti, Scutari, and Barbarossa [49] have the perspective of IoT devices as they aim to minimize the power consumption for each IoT device. Li and Wang [31] have the perspective of edge computing servers as they aim to minimize the energy consumption of edge computing servers. Chen, Wang and Sheng [8] have the perspective of edge computing devices as they aim to achieve virtual energy sharing in edge computing networks through task reallocations.

In this dissertation research, for the system model, we consider both the performance requirement and the security requirement of tasks. When generating the task distribution solution, we not only generate the mappings between tasks and edge computing devices, but also the data routings of tasks in the edge computing network. Hence, both the

computing resources of edge computing devices and the network resources of the edge computing network will be considered. Compared to our work, existing efforts mainly consider either computing resource allocation and network provisioning separately [75]. For the optimization objective, this dissertation research aims to maximize task accommodation number with the consideration of tasks' priorities. Our task distribution solution guarantees the QoS requirements of tasks, which include task completion deadline and security requirement. For the perspective of optimization, our approach is from the perspective of edge computing networks.

Chapter 3

OUR TASK DISTRIBUTION APPROACH

3.1 Overview

In this chapter, our overall approach to task distribution in edge computing networks will be presented [55].

One assumption is made in our approach, which is: The tasks to be processed in edge computing networks do not have any dependency relationships between them. By dependency, it means information flows between tasks. This assumption is normally valid as the tasks are from multiple IoT applications and/or various IoT devices.

A task refers to a program from IoT applications, and it is static. An IoT application may contain multiple tasks. An IoT device may run multiple IoT applications. When the task is being executed in a virtual machine on edge computing devices, it could be one or multiple processes during the run-time.

As tasks to the edge computing network are continuously sent from IoT devices, our task distribution approach periodically applies our task distribution process, which takes inputs from tasks need to be processed and the edge computing network, and generates task distribution solution. Let σ be the time interval between two consecutive applications of task distribution processes. Because input tasks are dynamically changing in terms of both arrival rate and specifications (workload, data size, deadline, etc.), σ

should be dynamically adjusted to maintain acceptable performance of our task distribution process at each application. As the performance of our task distribution process could be measured by the rate of successful task distributions in the edge computing network, which is a value between 0 and 1, inclusive, ϭ should be adjusted based on the rate of successful task distributions. If the rate of successful task distributions falls below a preset threshold, for example, 0.6, ϭ should be reduced so that our task distribution process will be applied more frequently.

With our task distribution approach, the following two questions will be answered:

(1) Which edge node shall the tasks be distributed to? The assigned edge node should satisfy a task's security requirement and complete the task within its deadline. There are three factors should be taken into consideration when making this decision, the currently available resources of the edge node, the QoS requirements of the task, and the available network bandwidth resource between the task's IoT device and its execution edge node.

(2) How much network bandwidth should be reserved for transmitting a task from its access edge node to its execution edge node? If a task is to be executed locally, there is no need to reserve any bandwidth, as there is no data transmission in the edge computing network for the task. If a task is to be executed outside its access node, we need to decide how to transmit its data from its access node to its execution node, how to select the network routes as there are usually multiple routes between two nodes, and how to assign flow on each route. All these questions will be answered by our task distribution approach.

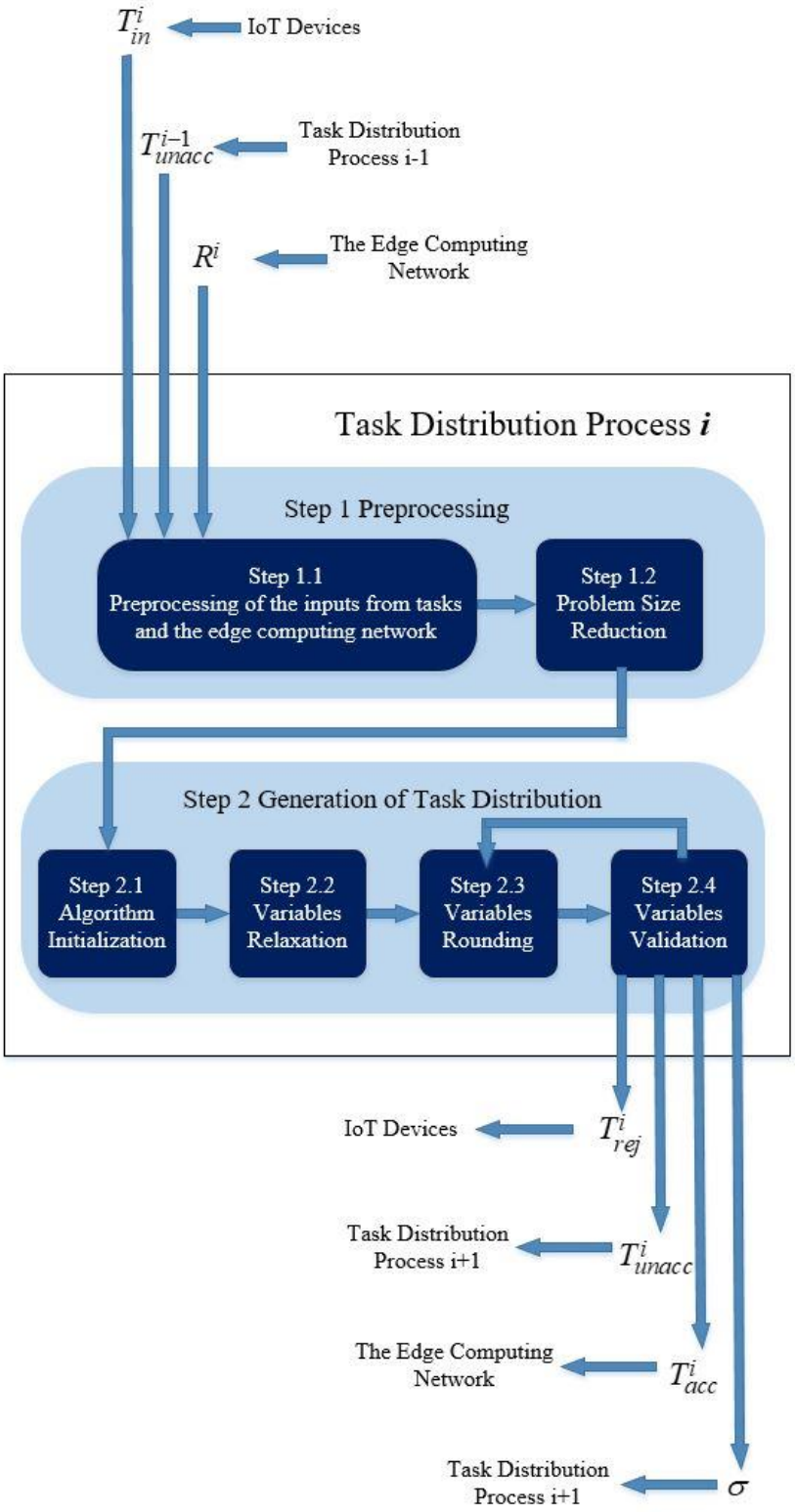

Figure 4 Our Task Distribution Process

3.2 The Task Distribution Process

Our task distribution process can be depicted in Figure 4 and summarized as follows.

Table 1

Notations for Task Distribution Process

T_{in}^{i}	Set of input tasks from IoT devices between task distribution processes $i-1$ and i
T_{acc}^{i}	Set of accommodated tasks in task distribution process i
T_{unacc}^{i-1}	Set of unaccommodated tasks in task distribution process $i-1$
T_{rej}^{i}	Set of rejected tasks in task distribution process i
R^{i}	Available resources of the edge computing network at the application of task distribution process i
ϵ	Time interval between two consecutive task distribution processes

There are three inputs for each task distribution process, except for the first task distribution process, which has two inputs and does not have Input 2, as there is no task distribution process before the first task distribution process. Input 1, denoted as T_{in}^{i}, includes all the tasks sent from IoT devices between task distribution process $i-1$ and i. Input 2, denoted as T_{unacc}^{i-1}, includes all the tasks that cannot be accommodated from the task distribution process $i-1$. Input 3, denoted as R^{i}, includes the available resources of the edge computing network when applying task distribution process i. All the information from these three inputs will be preprocessed in Step 1.

In Step 2, our proposed algorithm solves the task distribution problem and generate four outputs. Output 1, denoted as T_{rej}^{i}, includes the tasks that cannot be completed by the edge computing network. Rejected tasks are either too larger or have too tight requirement

to be completed by the edge computing network, and they will be sent back to IoT devices as the edge computing network is unable to complete these tasks. Output 2, denoted as T_{unacc}^{i}, includes tasks that cannot be completed during the task distribution process i. Different from tasks in Output 1, the tasks in output 2 can be completed by the edge computing network, but not in current task distribution process, hence, they will be sent to task distribution process $i + 1$ as its Input 2. An adjustment is required for these tasks that in task distribution process $i + 1$, the deadline requirements of these tasks will be reduced by the amount of current ϭ, which is the time interval between the task distribution processes i and $i + 1$. Output 3, denoted as T_{acc}^{i}, includes the accommodated tasks in task distribution process i. These tasks will be forwarded to their corresponding execution edge nodes. Output 4 is ϭ, which is the feedback for the adjustment of the time interval between task distribution processes i and $i + 1$. As discussed in Section 3.1 Overview, ϭ is dynamically adjusted based on the task accommodation rate of each task distribution process. For example, if the task accommodation rate of current process falls below a preset threshold, ϭ should be reduced by a preset factor, so that our task distribution process will be applied more frequently, and the performance of the edge computing network will be increased and the resource utilization will be more optimized. On the other hand, if the task accommodation rate of current task distribution process is above the preset threshold, there is no need to adjust ϭ.

Within a task distribution process, there are two steps. Step 1) is the preprocessing of the task distribution problem. Step 1.1) takes the QoS requirements of tasks and resource availability of the edge computing network to formulate the task distribution problem to a mixed-integer nonlinear program (MINLP). It then linearizes the MINLP, which can not

be solved in polynomial time. Step 1.2) further reduces the problem size by eliminating unfeasible task distributions. Step 2) uses our proposed algorithm to generate the task distribution solution. Step 2.1) initializes the data structures for executing the algorithm. Step 2.2) relaxes integer variables to continuous variables, which derives a linear program relaxation (LPR) of the MINLP. Step 2.3) solves the LPR and rounds solutions of relaxed continuous variables back to integers. Step 2.4) validates rounded solutions of variables. If the solution of a variable is not validated, it goes back to Step 2.3) and redo the rounding. Otherwise, the solution is validated, and the algorithm proceeds to process next task until it finishes all tasks.

Table 2

Constants of Edge Commuting Networks

$V = \{v_0, \cdots v_h, \cdots v_{\eta-1}\}$	Set of types of VM
sec_h	Lower bound of security strength that the VM type v_h has [72]
$D = \{d_0, \cdots, d_i, \cdots, d_{m-1}\}$	Set of edge computing devices
l_i^h	The number of available VM of type v_h on node d_i
r_i	Computing rate of VMs on node d_i
st_i	Available storage of node d_i
$E = \{e_0, \cdots, e_j, \cdots, e_{n-1}\}$	Set of edge links
$in(i)$	Set of incoming links of node d_i
$out(i)$	Set of outgoing links of node d_i
b_j	Available bandwidth of link e_j

3.2.1 Preprocessing the Task Distribution Problem

To preprocess the task distribution problem in edge computing networks, we first describe the edge computing network and the tasks. Then we generate the objective

function with various constraints of edge computing networks and QoS requirements of tasks.

3.2.1.1 Preprocessing Tasks and Edge Computing Networks

3.2.1.1.1 Infrastructure Model for Edge Computing Networks

An edge computing network is built on the traditional network infrastructure, where devices such as cellular base stations, access points, routers, and switches, are associated with certain amounts of computing and storage resources [4] [18] [69] [70] [77].

An edge computing network can be represented by a directed graph $G = (D, E)$, where $D = \{d_0, \cdots, d_i, \cdots, d_{m-1}\}$ is the set of edge nodes, and $E = \{e_0, \cdots, e_j, \cdots, e_{n-1}\}$ is the set of edge links.

The set of incoming links of edge node d_i is denoted by $in(i)$, and the set of outgoing links of edge node d_i is denoted by $out(i)$. Let b_j be the available bandwidth of link e_j of the edge computing network at the time of applying the task distribution process.

Each edge node runs a number of VMs, and each VM performs one task at a time. Different types of VMs are supported in our approach [51]. VMs can be different in two ways. The first difference is the operating system. For example, Windows and Linux are two different operating systems. The second difference is the version/release. For example, Windows 7 and Windows 10 are two different Windows versions. Another example is Ubuntu 16.04 and Ubuntu 18.04 are two different releases. There are compatibility constraints between tasks and different types of virtual machines. Assigning tasks to

incompatible virtual machines results in rejection of tasks. Hence, our task distribution process takes into consideration the compatibility constraints between tasks and different types of virtual machines. Let l_i^h be an integer representing the count of VM v_h available on edge node d_i at the time of applying our task distribution process. The length of the list is the number of the type of VM supported by the edge computing network.

Each VM on edge node v_i is granted with computing rate r_i. There are two major methods to quantify the computing rate in related optimization works. One method is to quantify computing rate as the number of instructions per second [3]. The other method is to quantify computing rate as CPU cycles per second [9] [38] [65]. In this work, there is no specification on which quantification method should be applied, as both work fine. Let st_i be the available storage of edge node v_i at the time of applying our task distribution process. For storage capacity, it does not need to be associated with each VM, as in most cases, the storage capacity for each VM is elastic, as long as there is available storage capacity on the edge node.

Let sec_h be the lower bound of the security strength that VM v_h has. There are two major methods to quantitatively measure security strength. One method is based on the parameter configurations of security mechanisms applied in computing systems. For example, in [72], the security parameter used to determine security strength are security functionality, security algorithm, key length, and protection percentage. The security functionality can be confidentiality, integrity, and non-repudiation. The security algorithm can be DES or AES for confidentiality. Key length is an important factor for security strength. Usually, longer key length is more secure but consumes more resources. Protection is also an important factor. Larger protection percentage is more

secure but consumes more resources [71] [72]. The other major method to quantify security strength is through trust establishment [16] [51] [54]. For example, in [54], a trust establishment tool "Trust-Sniffer" is introduced. The "Trust-Sniffer" establishes trust with three steps. It first uses a minimal trusted operating system to validate the host OS. Then, the trusted host OS is booted and validates applications. At last, the host OS only permits trusted applications to execute. In this work, there is no specification on which security quantification method should be applied, as both work fine.

Table 3

Constants of Tasks

$\mathrm{T} = \{\mathrm{t}_0, \cdots, \mathrm{t}_\mathrm{k}, \cdots, \mathrm{t}_{\psi-1}\}$	Set of tasks
a_k	Access node of task t_k
w_k	Computation workload of task t_k
d_k	Data size of task t_k
st'_k	Storage requirement of task t_k
sec'_k	Upper bound of the required security strength of task t_k
δ_k	Completion deadline of task t_k
b_{a_k}	Access bandwidth of task t_k
p_k	Priority of task t_k
l'^h_k	The binary indicator of compatibility between VM v_h and task t_k

3.2.1.1.2 Service Model for Tasks

Let $T = \{t_0, \cdots, t_k, \cdots, t_{\psi-1}\}$ be the set of tasks to be processed at the time of applying our task distribution process. A task is defined as $t_k = (a_k, w_k, d_k, st'_k, sec'_k, \delta_k, p_k, l'_k)$. a_k is the access node of t_k. w_k is the computation workload [27] of t_k, which can be quantified as the number of instructions [3], or the

number of CPU cycles [9] [38] [65]. d_k is the data size of t_k. st'_k is the storage requirement of t_k. sec'_k is the upper bound of the required security strength of t_k. δ_k is the completion deadline of t_k. p_k is the priority of t_k. Let l'^h_k be a binary indicator representing the compatibility of task t_k and VM v_h supported by the edge computing network. An indicator with a value of 1 means the task and VM are compatible, while a value of 0 means they are not compatible The length of the list is η, which is the number of the type of VM supported by the edge computing network.

As all the tasks are initially sent from IoT devices to their connected access nodes, we further denote b_{a_k} as the access bandwidth for t_k, which is the bandwidth between the IoT device that sends t_k and its access point a_k.

3.2.1.1.3 Constraints of Task Distribution

Before discussing these constraints, we need to identify the variables for task distribution due to the characteristics of the task distribution problem.

Let binary variable $x_{k,i}$ be the task distribution indicator. $x_{k,i} = 1$ if t_k is decided to be distributed to d_i, and $x_{k,i} = 0$ if $\mathrm{t_k}$ is decided not to be distributed to d_i. Let $f_{k,j}$, $0 \leq f_{k,j} \leq b_j$, be the required bandwidth on edge link e_j for the flow of task t_k to pass through e_j.

The constraints of the task distribution problem are given below:

C-1: Task assignment constraint:

In this work, each task is atomic and independent. A task t_k can be assigned to at most one edge node and must be executed and completed on the assigned edge node. Hence,

the summation of all the task distribution variable of task t_k should be either 0 or 1. A summation equal to 0 means that task t_k is successfully distributed. A summation equal to 1 means that task t_k is not distributed. That is,

$$\sum_{i=0}^{m-1} x_{k,\,i} \leq 1\,. \tag{1}$$

C-2: Node storage constraint:

To execute a task, edge nodes must receive the task data from the task's IoT device. An edge node d_i must have sufficient storage to store the data of all the tasks distributed to d_i. Hence, the summation of the storage requirement of all the tasks distributed to edge node d_i should be no larger than the storage capacity of d_i. That is,

$$\sum_{k=0}^{\psi-1} x_{k,\,i} st'_k \leq st_i\ . \tag{2}$$

C-3: Security constraint:

One of the two QoS requirements of tasks we consider is the security requirement of tasks. To be capable of completing task t_k, the VMs on each node d_i must be sufficiently secure in order to satisfy the specified security requirement of t_k. The methods to quantify security strength is discussed above in this section. The security constraint is formulated as,

$$x_{k,i} sec'_k \leq sec_i\,. \tag{3}$$

C-4: VM availability constraint:

The compatibility of a task and a type of VM is discussed above in this section. To be capable of completing task t_k, edge node d_i must have VMs that are compatible with tasks t_k available at the application of the task distribution process. That is,

$$x_{k,i} l'^{h}_{k} \leq l^{h}_{i} \, . \quad (4)$$

C-5: Task completion time constraint:

This is the second one of the two QoS requirements of tasks we consider in this dissertation research. It means that each accommodated task t_k must be completed within the deadline specified by its IoT application.

The completion time of t_k on an edge node d_i consists of two components: t_k's data transmission time between t_k's IoT device and its execution node d_i, and the execution time of t_k on its execution node d_i. The initial setup time for establishing the connection between IoT devices and edge nodes, the time of sending the IoT applications' programs from IoT devices to edge nodes, and the time for nodes to send computation results back to IoT devices are negligible compared to the task execution time and data transmission time [27]. So, all these time factors are not considered here. In addition, for the network delay, the processing delay, the queuing delay, and the propagation delay are also not considered in this work and only transmission delay is considered [29].

The first component consists of the t_k's data transmission time between t_k's IoT device and its access node a_k, and the data transmission time between t_k's access node a_k and its execution node d_i. Recall that we use b_{a_k} to denote the bandwidth between t_k's IoT device and its access node a_k. Let $b_{d_i}^{a_k}$ be the bandwidth for task t_k to transmit its data from its access node a_k to its execution node d_i, then the bandwidth between the IoT device and the execution node d_i is

$$min\left\{b_{a_k}, b_{d_i}^{a_k}\right\}.$$

The data transmission time between t_k's IoT device and the execution node, denoted as $\tau_k^{tr}(d_i)$, is then given by

$$\frac{d_k}{min\left\{b_{a_k}, b_{d_i}^{a_k}\right\}}.$$

The second component of the task completion time of t_k, is the execution time of t_k on d_i, denoted as $\tau_k^{ex}(d_i)$, is given by

$$\frac{w_k}{r_i}.$$

The constraint on the completion time for each task t_k, denoted as τ_k, is the sum of both is data transmission time $\tau_k^{tr}(d_i)$ and task execution time $\tau_k^{ex}(d_i)$. To meet t_k completion deadline, the completion time of t_k should be no larger than t_k's deadline δ_k. That is,

$$\begin{aligned}
\tau_k &= \sum_{i=0}^{m-1} \tau_k(d_i) x_{k,i} \\
&= \sum_{i=0}^{m-1} [\tau_k^{tr}(d_i) + \tau_k^{ex}(d_i)] x_{k,i} \\
&= \sum_{i=0}^{m-1} \left[\frac{d_k}{min\left\{b_{a_k}, b_{d_i}^{a_k}\right\}} + \frac{w_k}{r_i} \right] x_{k,i} \\
&\leq \delta_k. \qquad (5)
\end{aligned}$$

C-6: Link bandwidth sharing constraints:

Note that in constraint C-5, the bandwidth allocation for each task between its access edge node and execution node has yet been decided. Due to the path diversity and the dynamic bandwidth availability in the edge computing network, we use the network flow abstraction to capture the network sharing among different tasks [10]. The flows in the edge computing network are the data transmitted from tasks' access edge nodes to their execution edge nodes. A flow in a flow network satisfies two properties, which are (1)

capacity constraint for each link and (2) flow conversation for each node, except the source node and the destination node.

Hence, there are two constraints for edge link bandwidth sharing by multiple tasks in the edge computing network. The first constraint C-6.1 is derived from the flow conservation property. It concerns the difference between the total bandwidth of all the incoming and all the outgoing links used by the flow of task t_k of each node in the edge computing network. The second constraint C-6.2 is derived from the capacity constraint property. It concerns the link capacity constraint of each link in the edge computing network, which comes from the second property of network flow.

C-6.1: Link bandwidth conservation constraint:

For an edge node d_i, which is neither the access node, nor the execution node, of t_k, the sum of the required bandwidth on all the incoming links to d_i for the flow of t_k must be the same as the sum of the required bandwidth on all the outgoing links from d_i for the flow of t_k.

For an edge node d_i, which is the execution node of t_k, the sum of the required bandwidth for the flow of t_k on all the incoming links to d_i must be greater than the sum of the required bandwidth for the flow of t_k on all the outgoing links from d_i by $b_{d_i}^{a_k}$, which is the bandwidth for task t_k to transmit its data from its access node $a_k$to its execution node d_i. For the access node a_k, which is the source node for task t_k's data flow, the sum of the required bandwidth for the flow of t_k on all the incoming links to a_k must be smaller than the sum of the required bandwidth for the flow of t_k on all the outgoing links from a_k by $b_{d_i}^{a_k}$. As the sum of flows through the source node and the destination node is zero, there is

no need to add the flow conservation of the source node as a constraint, as it is represented by the flow conservation of the destination node.

Hence, the flow conservation constraint is expressed as follows,

$$\sum_{e_j \in in(i)} f_{k,j} - \sum_{e_j \in out(i)} f_{k,j} = b_{d_i}^{a_k} \cdot x_{k,i} \,. \tag{6}$$

C-6.2: Link bandwidth constraint:

For each link e_j, the sum of the required bandwidth for the flows of all the tasks to pass through e_j must be no greater than the capacity of link e_j. That is,

$$\sum_{k=0}^{\psi-1} f_{k,j} \leq b_j \,. \tag{7}$$

3.2.1.1.4 The Task Distribution Program

Note that due to Constraint (1), some tasks may not be distributed to any edge node for execution. In practice, not all tasks can be accommodated for processing in the edge computing network because of the limited resources in edge computing network [9].

The objective of the task distribution generated by our approach is to maximize the number of tasks that can be accommodated in the edge computing network, with the consideration of tasks priorities. To consider the priorities of tasks, we add them as weighting factors in our objective function. Without the consideration of resource consumption, completing a task with a higher priority means better performance for the edge computing network than completing a task with a lower priority. However, in certain cases, a task with a higher priority consumes more computing resources and network bandwidth than a task with a lower priority. From the perspective of the edge computing

network, we want to increase the performance of the entire system, not only one task. Hence, in the objective function, we introduce the priorities as weighting factors of each task. Our problem formulation considers resource utilization while doing optimization. Through the optimization process, a trade-off will be made between priority weighted accommodation number and resource utilization. The task distribution problem can be formulated as follows:

$$max \sum_{k=0}^{\psi-1} \sum_{i=0}^{m-1} p_k x_{k,i} \,, \quad s.t. (1) - (7). \tag{8}$$

Note that (8) is a mixed-integer nonlinear program (MINLP), and is *NP*-hard because of the following reason: Consider a special case of the task distribution program, which assumes that all VMs in the edge computing network finish their assigned tasks instantly, all nodes have the same storage available, satisfy the maximum security strength, and all edge links have infinite bandwidth, is equivalent to the decision version of Bin Packing problem, which is *NP*-hard [15]. Because this special case is *NP*-hard, (8) is *NP*-hard. Due to the *NP*-hard nature of (8), it is impossible to generate the optimal solution for (8) in polynomial time, unless $P = NP$ [15].

In order to achieve the goal of our task distribution approach, we will use a mixed-integer linear program (MILP) by simplifying Constraint (5) and (6) in (8).

3.2.1.1.5 Linearization of the Task Distribution Program

The nonlinearity of (8) is due to Constraints (5) and (6), which are both nonlinear because they involve both $x_{k,i}$ and $b_{d_i}^{a_k}$. By assuming that the execution of all accommodated tasks will be completed exactly at their respective deadlines, $b_{d_i}^{a_k}$ can be

replaced by a constant $\hat{b}_{d_i}^{a_k}$, which will be defined in (11). This assumption does not increase any limitations on the services provided by the edge computing network since each accommodated task still satisfies its completion time duration requirement. The details of the linearization are explained as follows:

Given an execution node v_i, Constraint (5) now becomes

$$\frac{d_k}{min\left\{b_{a_k},\, b_{d_i}^{a_k}\right\}} + \frac{w_k}{r_i} \leq \delta_k \,. \tag{9}$$

Therefore, we can solve (9), and have the following inequality:

$$min\left\{b_{a_k},\, b_{d_i}^{a_k}\right\} \geq \hat{b}_{d_i}^{a_k} \,, \tag{10}$$

Where

$$\hat{b}_{d_i}^{a_k} = \frac{d_k}{\delta_k - \frac{w_k}{r_i}} \,. \tag{11}$$

In other words, if $b_{a_k} < \hat{b}_{d_i}^{a_k}$ or $\hat{b}_{d_i}^{a_k} < 0$, then node d_i cannot be assigned as t_k's execution node, because Constraint (5) is not satisfied. On the other hand, if $0 < \hat{b}_{d_i}^{a_k} \leq b_{a_k}$, then Constraint (5) is equivalent to the following

$$b_{d_i}^{a_k} \geq \hat{b}_{d_i}^{a_k} \,. \tag{12}$$

Note that letting equality hold for (12), which means that all tasks are completed exactly on their respective deadlines, has no impact on the optimal solution set, since bandwidth allocation may only reduce on each link with a lower bandwidth requirement of $\hat{b}_{d_i}^{a_k}$. Hence the variable $b_{d_i}^{a_k}$is now resolved to a constant number $\hat{b}_{d_i}^{a_k}$ in Constraint (5).

To sum up, the linearization consists of two components: first, for each task t_k, remove all potential execution nodes $d_i \in D$ where $b_{a_k} < \hat{b}_{d_i}^{a_k}$ or $\hat{b}_{d_i}^{a_k} < 0$; second, replace Constraints (5) and (6) with the following:

$$\sum_{e_j \in in(i)} f_{k,j} - \sum_{e_j \in out(i)} f_{k,j} = \hat{b}_{d_i}^{a_k} \cdot x_{k,i} \ , \quad \text{for } \forall d_i \in D \backslash \{a_k\}. \tag{13}$$

The resultant program is now an MILP:

$$max \sum_{k=0}^{\psi-1} \sum_{i=0}^{m-1} p_k x_{k,i} \ , \quad s.t. \ (1) - (4), (7), (13). \tag{14}$$

3.2.1.2 Problem Size Reduction

We first reduce the size of Program (14) by removing unnecessary $x_{k,i}$ variables by setting unfeasible $x_{k,i}$ variable to a fixed value 0. Define $D_k = \{d_i \in D | sec_i < sec_k'\}$ as the set of candidate execution nodes for task t_k who satisfy the security constraint.

We can then restrain the set of task assignment variables to only contain $x_{k,i}$ where $d_i \in D_k$, and remove Constraints (3).

Secondly, we reduce the size of Program (14) by removing tasks that cannot be completed by the edge computing network.

Generate a min flow program for the data routing in the edge computing network by replacing the objective function in the LPR of (14) with a new objective function, which is to minimize the sum of the flows of all tasks in the edge computing network. That is,

$$min \sum_{k=0}^{\psi-1} \sum_{j=0}^{n-1} f_{k,j} \,, \qquad (15)$$

$$s.t. (1) - (4), (7), (13).$$

For each input task t_k, we solve the min-flow Program (15) with the assumption that t_k is the only input task. If Program (15) does not have a feasible solution, it means t_k cannot be completed by the edge computing network. On the other hand, if Program (15) has a feasible solution, it means t_k can be completed by the edge computing network and it will be considered in Step 2 of the task distribution process. All tasks that cannot be completed by the edge computing network will be added to a rejected task set T_{rej}.

3.2.2 Generation of Task Distribution Solution

We generate the task distribution by solving the MILP (14), which is the Step 2 of our overall task distribution process. It has four sub-steps, which will be presented as follows.

Step 2.1) Initiate this sub-step by setting both be the set of accommodated tasks, T_{acc}, and the set of failed tasks, T_{unacc}, as empty sets.

Step 2.2) Relax the MILP (14) by replacing the binary variables $x_{k,i} \in \{0, 1\}$ with new continuous variables $\bar{x}_{k,i} \in [0, 1]$, that is each $\bar{x}_{k,i}$ has a value between 0 and 1, inclusive. The resultant program with relaxed variables $\bar{x}_{k,i}$ is called a linear program relaxation (LPR) of the original program (14).

Step 2.3) Solve the LPR of (14) over T ∪ T_{acc}, and obtain the values of the relaxed variables $\bar{x}_{k,i}$. Find the variable $\bar{x}_{k,i}$ with the largest value, round off its value to 1, and all other variables $\bar{x}_{k,i'}$ $(d_i' \neq d_i)$ for the same task t_k to 0. Then go to Step 2.4).

Step 2.4) Update T_{acc} to $T_{acc} \cup t_k$. Validate the rounding in Step 2.3) by checking if (15) with the updated accommodated task set, has any feasible solution. If there is a feasible solution, the rounding of the values of $\bar{x}_{k,i}$ and $\bar{x}_{k,i'}$in Step 2.3) is validated, and move t_k from T to T_{acc}. In addition, put the tasks with all their variables $\bar{x}_{k,i}$ marked as failed the validation to T_{unacc}. Then go to Step 2.3) if T is not empty. On the other hand, if there is no feasible solution for (15), the rounding of the values of $\bar{x}_{k,i}$ and $\bar{x}_{k,i'}$ in Step 2.3) will be undone, variable $\bar{x}_{k,i}$ will be marked as failed the validation. Then go back to Step 2.3) to round off the variable with the next largest value. The solution of last min flow program that has a feasible solution is used to update network bandwidth resources.

As mentioned in Section 3.1, ϭ, which is the time interval between two consecutive task distribution processes, should be adjusted dynamically to maintain a satisfactory performance of the edge computing network. The input of this adjustment of ϭ is generated in Step 2.4), based on the task accommodation rate. At the end of Step 2.4), all tasks are labeled as accommodated, unaccommodated, or rejected. Hence, the task accommodation rate can be computed, which is a value between 0 and 1, inclusive. If the rate of successfully distributed tasks falls below a preset threshold, for example, 0.6, ϭ should be reduced so that our task distribution process will be applied more frequently.

This section is implemented by Algorithm 1. Algorithm 1 solves a sequence of LP-relaxations and the corresponding min flow data routing sub-problems (15) to obtain a feasible solution over the relaxed versions.

Line 1 initializes the accommodated and unaccommodated sets T_{acc} and T_{unacc} to empty. Line 2–20 is the outer loop over the set of all input tasks except rejected tasks in Step 1.2 Problem Size Reduction. In each iteration of the outer loop (except the last one), exactly one task is added to T_{acc}. To do this, first, we solve the LP-relaxation at the beginning of each iteration in Line 3. Let X be the set of (relaxed) task assignment variables optimized in this iteration's LP (Line 4).

The inner iteration (Line 5–16) then iterates over all variables in X, in descending order of their optimized values. In each inner iteration, the currently picked variable $\bar{x}_{k,i}$ will have its value set to 1, and all other variables $\bar{x}_{k,i}$ for the same task set to 0 (Line 6). We then solve the min flow data routing sub-problem (15) over the subset of tasks defined by $T_{acc} \cup t_k$ (all successfully scheduled tasks and the current task), with all task assignment variables fixed (for tasks in T_{acc}, it is based on the node assigned in previous iterations), in Line 7. Note that the min flow data routing sub-problem with fixed task assignment is also an LP, hence can be solved efficiently. If the min flow data routing sub-problem returns a feasible solution, it means that the current task assignment of $T_{acc} \cup t_k$ can be satisfied by the network. We then fix the task assignment for t_k in Line 9, move t_k from T to $\mathrm{T_{acc}}$ in Line 10, and terminate the inner loop. Otherwise, the current task assignment of $T_{acc} \cup t_k$ is infeasible. We then undo the assignment of variables in Line 13, remove this variable $x_{k,i}$ from X in Line 14, and proceed to the next largest variable until

all variables are iterated in the inner loop. After the inner loop, the algorithm checks whether if there are any task t_k that cannot be satisfied with the currently scheduled tasks T_{acc}: if all variables $x_{k,i}$ has been iterated in the inner loop for task t_k and no feasible solution is found, we then move the task to the unaccommodated set T_{unacc}in Line 18.

After all tasks' distributions have either been successfully decided (moved to T_{acc} or T_{unacc}), the algorithm terminates and returns the accommodated and unaccommodated task sets, the per-successful task assignment, and the corresponding min flow data routing solutions $\{f_{k,j}\}$. Note that the min flow data routing solution is obtained from the last inner iteration that is feasible, which essentially includes all tasks that finally result in T_{acc}.

Algorithm 1 terminates in polynomial time. To see this, observe that in each outer iteration (Line 2-20), either a task is added to T_{acc} at Line 10, or all the rest tasks in T fail the validation and be moved to T_{unacc} at Line 18. The size of T is reduced by at least 1 in each outer iteration. Hence the outer loop terminates in at most ψ iterations (ψ is the number of tasks initially in T). The inner loop (Line 5-16) iterates over all task distribution variables, which is at most of the size $O(\psi m)$, where m is the number of nodes in D. Since LPs in each iteration can be solved in polynomial time [74], the algorithm terminates in polynomial time as well.

Algorithm 1: Task Distribution and Data Routing (TDDR)

Input: The edge computing network $G = (D, E)$, set of tasks to be processed T

Output: Set of accommodated tasks T_{acc}, set of unaccommodated tasks T_{unacc}, task distribution $\{x_{k,i}\}$, flow of tasks $\{f_{k,j}\}$

1 $T_{acc} \leftarrow \emptyset, T_{unacc} \leftarrow \emptyset$

2 **while** $T \neq \emptyset$ **do**

3 Solve LPR of (14) and obtain $\{x_{k,i}\}$ and $\{f_{k,j}\}$

4 $X \leftarrow \{\bar{x}_{k,i} | t_k \in T, d_i \in D\}$;

5 **for** *each* $\bar{x}_{k,i} \in X$ *in descending order of value* **do**

6 Assign $\bar{x}_{k,i} \leftarrow 1$, and $\bar{x}_{k,i'} \leftarrow 0$ for $\forall d_i' \neq d_i$;

7 Solve a min-flow program (15) for $T_{acc} \cup t_k$ with fixed task assignment;

8 **if** *the min-flow program has a feasible solution*

9 **then**

10 Fix task assignment for t_k;

11 $T_{acc} \leftarrow T_{acc} \cup t_k$, $T \leftarrow T \setminus \{t_k\}$;

12 **break**;

13 **else**

14 Un-assign $\bar{x}_{k,i}$ and $\bar{x}_{k,i'}$ values in Line 6;

15 **end**

16 **end**

17 **for** *each* $t_k \in T$ *where* $\bar{x}_{k,i} \notin X$ *for* $\forall d_i \in D$ **do**

18 $T_{unacc} \leftarrow T_{unacc} \cup t_k$, $T \leftarrow T \setminus \{t_k\}$;

19 **end**

20 **end**

21 **return** T_{acc}, T_{unacc}, $\{x_{k,i}\}$, $\{f_{k,j}\}$

Chapter 4

EVALUATIONS

4.1 Overview

In this section, we will present the evaluation of our approach to task distribution in the edge computing network. The simulation results show the improvement of our approach on increasing the accommodation number of tasks the edge computing network, with consideration of priorities, compared to the four comparison approaches. The four comparison approaches are the local execution approach (Local) and random distribution approach (Random), the greedy distribution approach based on fastest completion time (Greedy-Fastest), and the greedy distribution approach based on slowest completion time (Greedy-Slowest). The two greedy based approaches are based [18] and [77].

The Local approach does not involve task distribution in the edge computing network. In the Local approach, each task is executed locally on its access node. For all the input tasks, we first sort them according to the priorities from high to low. We use merge sort based sorting as merge sort is a stable sorting algorithm. Hence, for the tasks with the same priorities, they are computed based on the order of arrivals. For each task in the sorted order, we solve the task with the min-flow data routing Program (15). If it does not have any feasible solution, this task is moved to the unaccommodated set. If it has a feasible solution, this task is moved to the accommodates set and we use the solution of Program (15) to update the resources of the edge computing network, which includes, the storage capacity of the edge node, the VM availability of the edge node and the bandwidth of the

network links. This step is to reserve resources for the accommodated tasks. Then we move to the next task until finish all tasks.

The Random approach randomly selects an edge node for each task as its execution node. We first sort all the tasks based on the same sorting rules in the Local approach. For each task, we randomly select an edge node as its execution node. Then we solve the min flow data routing Program (15). If it has a feasible solution, we update the resources of the edge computing network and move this task to the accommodated set. If it does not have any feasible solution, we randomly select the next edge node and solve Program (15). If all the edge nodes have been tested and none of them has a feasible solution of Program (15), this task is moved to the unaccommodated set.

The Greedy-Fastest approach and the Greedy-Slowest approach perform all steps the same as the Random approach, exception the selection of execution node for each task. The Greedy-Fastest selects an eligible node for each task that could complete the task fastest. If the fastest node could not complete a task within the deadline, the task will be moved to the unaccommodated set. Otherwise, the task will be moved to accommodated set and we update resources of the edge computing network. The Greedy-Slowest approach selects an eligible node for each task that could complete the task slowest but within the deadline. The slowest and acceptable completion time for a task is the deadline. Hence, if an edge node can complete a task on the deadline, its completion time is the slowest and acceptable, and this edge node will be the execution node.

Greedy based task distribution algorithms have been proposed and implemented for the purpose of comparison. Zeng, Gu, Guo, Cheng, and Yu [77] implemented two greedy

based comparison algorithms. One is server greedy algorithm, which greedily distributes tasks to resourceful servers until servers reach saturation. The other one is client greedy algorithm, which greedily distributed tasks to clients until clients reach saturation. Gu, Zeng, Guo, Barnawi, and Xiang [18] implemented a greedy based algorithm for user association and execution edge node selection. As mentioned in Chapter 2 Current State of Art, the system model in this work regards user association as a variable. Their greedy algorithm first greedily associate users to a base station that has the smallest uploading cost. It then selects a base station that has the smallest incremental cost as the task's execution node. The incremental cost includes VM deployment cost and inter base stations communication cost.

4.2 Simulations Setup

We run all the simulations on a Linux virtual machine on a Dell PC. The processor of the Dell PC is Intel(R) Core(TM) i7-3770 with 4 cores. The base frequency is 3.40 GHz. The RAM of the Dell PC is 8.00 GB. The operating system of the Dell PC is Windows 10 Enterprise, version 1709, OS build 16299.611 and 64-bit. For the Linux virtual machine, it is installed on the Dell PC through Oracle VM VirtualBox of version 5.2.12. There are three cores allocated to this VM out of the 4 cores the Dell PC has. The RAM allocated to this VM is 4.00 GB. The operating system of the Linux VM is Ubuntu 16.04.1 and 64-bit.

We use IBM CPLEX as the engine to solve the linear programs. The version of IBM CPLEX we used is 12.7.0. The program of linear program solver is written in C++.

Our approach and the four approaches used for comparison are written in Java. The Java version on the Ubuntu virtual machine is 1.8.0_181.

We use Waxman model [64] to generate network topologies for edge computing networks. The major benefit of applying Waxman model to generate the edge computing network topology is that Waxman model is simple to implement while its generated topologies satisfy the requirements of edge computing networks. Beside Waxman model, there are two major types of network topology models [37], hierarchical topology model, and power-law based model.

To the best of my knowledge, there are no requirements regarding the hierarchical structure and node degree for edge computing network. In the edge computing network, all edge computing devices can connect endpoint IoT devices, hence, the edge computing network is flat, and no hierarchical structures are required. Hence, hierarchical topology generators, such as Tiers [13] and the Transit-Stub [76] are not feasible for our research. Power-law based topology generators, such as Inet2 [26], the BRITE [40], and PLRG [1], define the degree distribution of network nodes according to certain power laws. Indeed, Waxman model is equivalent to a special case of BRITE 1.0 [39]. For the edge computing networks in our simulations, there are no requirements on the node degree, hence, power-law based topology generators are not necessary.

To conclude, Waxman model is sufficient to generate random network topologies for our simulations with the least effort to implement.

The formulation of Waxman model is:

$$p(u,v) = \alpha \cdot \exp\left(-d(u,v)/(\beta \cdot L)\right),$$

where $p(u,v)$ is the probability of the connection of nodes u and v. There are two adjustable parameters of Waxman model, which are α and β, both in the range of (0, 1]. A larger value of α yields to more links. And a larger value of β yields to a larger ratio of long links relative to shorter links. We further process the resultant network to make sure that all the edge nodes are interconnected. Hence, the resultant edge computing network is a directed graph.

The comparison metric in our simulations is the accommodated number of tasks, the accommodated rate of tasks, the weighted accommodated number of tasks, the bandwidth utilization of the edge computing network, the average number of links used by each task, the storage utilization of the edge computing network, and the VM utilization of the edge computing network.

To investigate the effects of different factors on the performance of task distribution process, we conduct three simulations with different settings. The first set of simulation is varying the number of tasks, which is to investigate the effect of the total workload. The second set of simulation is varying the data size of tasks, which is also to investigate the effect of the total workload. The last set of simulation is varying the connectivity of the edge computing network, which is to investigate the effect of network capacity and the degree of resources sharing between edge nodes. To make our result more reliable, for each of the three simulations, we run each simulation 10 instances and get the average of each comparison metric. In addition to showing the average of each metric, we also show the standard deviation of each metric in each plot. The standard deviations show the amount

of variation of each set of data. We observe that our approach has acceptable standard deviations.

4.3 Simulation with Varying Number of Tasks

To better simulate the real edge computing, we have two set of edge nodes. One set contains resource-rich nodes and the other set contains recourse-poor nodes.

For each recourse-rich node, the storage capacity is normally distributed in the range of [1000, 5000] MB The upper bound of security strength provided by VMs is normally distributed in the range of [0.4, 1]. (The quantification of security strength is discussed in Section 3.2.1 Preprocessing the task distribution problem). The computing rate of VMs on each resource-rich edge node is normally distributed in range the range of [2, 4] (The quantification of workload is discussed in Section 3.2.1 Preprocessing the task distribution problem). The number of each type of VM on each resource-rich edge node is normally distributed in the range of [20, 30].

For each recourse-poor node, the storage capacity is normally distributed in the range of [100, 500] MB. The upper bound of security strength provided by VMs is normally distributed in the range of [0.4, 0.9]. The computing rate of VMs on each resource-poor edge node is normally distributed in the range the range of [1, 2]. The number of each type of VM on each resource-rich edge node is normally distributed in the range of [5, 15].

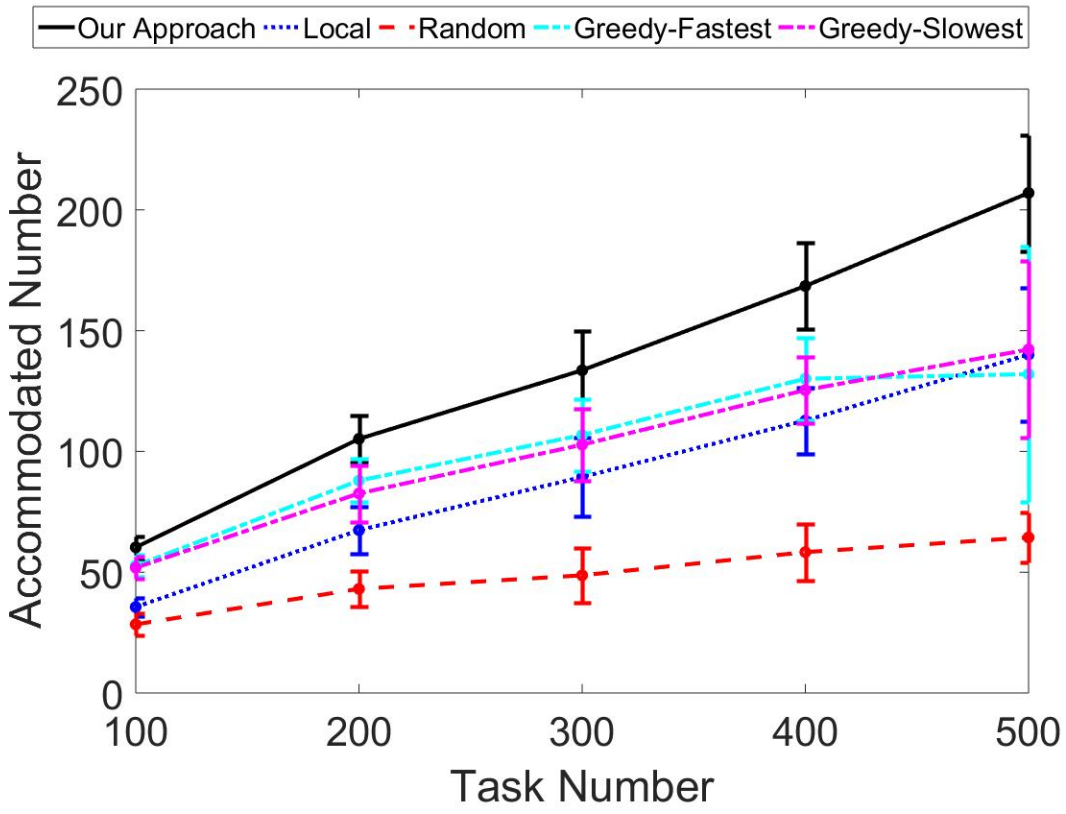

Figure 5 Accommodated Number for Simulation with Varying Task Number

There are 3 different types of VMs in the edge computing network. The access bandwidth of each edge node is normally distributed in the range of [100, 200] Mbps. The bandwidth of each edge link is normally distributed in the range of [50, 250] Mbps.

The settings of the tasks in this simulation are as follows. The data size of each task is normally distributed in the range of [1, 200] MB. We assume that the storage requirement of each task is the same as the task's data size. The lower bound of security strength required by each task is normally distributed in the range of [0.2, 1] (The quantification of security strength is discussed in Section 3.2.1 Preprocessing the task distribution problem). The deadline of each task is normally distributed in the range of [10, 20] seconds. The

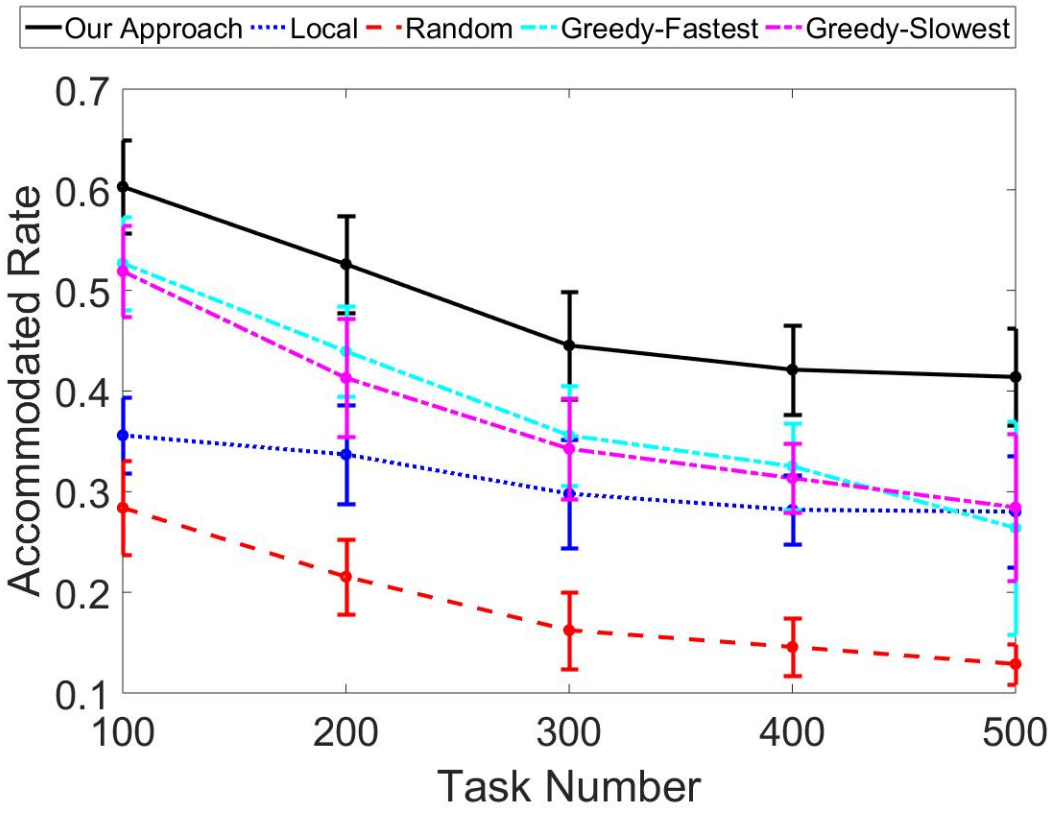

Figure 6 Accommodated Rate for Simulation with Varying Task Number

workload of each task is normally distributed in the range of [1, 30] (The quantification of workload is discussed in Section 3.2.1 Preprocessing the task distribution problem). The access edge node of each task is uniformly distributed among all the edge nodes. The priority of each task is uniformly distributed in the range of [90, 100]. For each task, its compatible VM is uniformly set among all the types of VMs.

In this simulation, to generate the edge computing network, we set the number of edge nodes to 8, and both α and β in the Waxman model to 0.9. We change the number of tasks to be processed from 100 to 500, with a step of 100, to examine the effect of the number of tasks.

Figure 5 shows the accommodated numbers of all the five approaches. It shows that on average, our approach achieves 54.4% more accommodated number than the Local approach, 168.2% more accommodated number than the Random approach, 29.1% more accommodated number than the Greedy-Fastest approach and 31.0% more accommodated number than the Greedy-Slowest approach. The coefficients of variation of our approach for different task number (from 100 to 500 with the step of 100) are 7.7%, 9.1%, 12.1%, 11.0%, and 11.6%.

When the number of tasks is small, the differences of accommodated numbers are small as the edge computing network resources, which include both edge nodes resources and edge network bandwidth resources, are enough for tasks' computation and transmission. As the number of tasks gets larger, the edge computing network resources become scarcer for tasks completion. As our approach performs even better when the number of tasks is larger, it proves that our approach is efficient for resource management. The performance of the Random approach is worse than the Local approach, hence, we conclude that it is better not distributing tasks, rather than distributing tasks randomly. Greedy-Fastest performs slightly better and more stable than Greedy-Slowest. Figure 5 also show the standard deviation

Figure 6 shows the average accommodated rate of all the five approaches. It shows that on average, our approach achieves 54.4% more accommodated rate than the Local approach, 168.2% more accommodated rate than the Random approach, 29.1% more accommodated rate than the Greedy-Fastest approach and 31.0% more accommodated rate than the Greedy-Slowest approach.

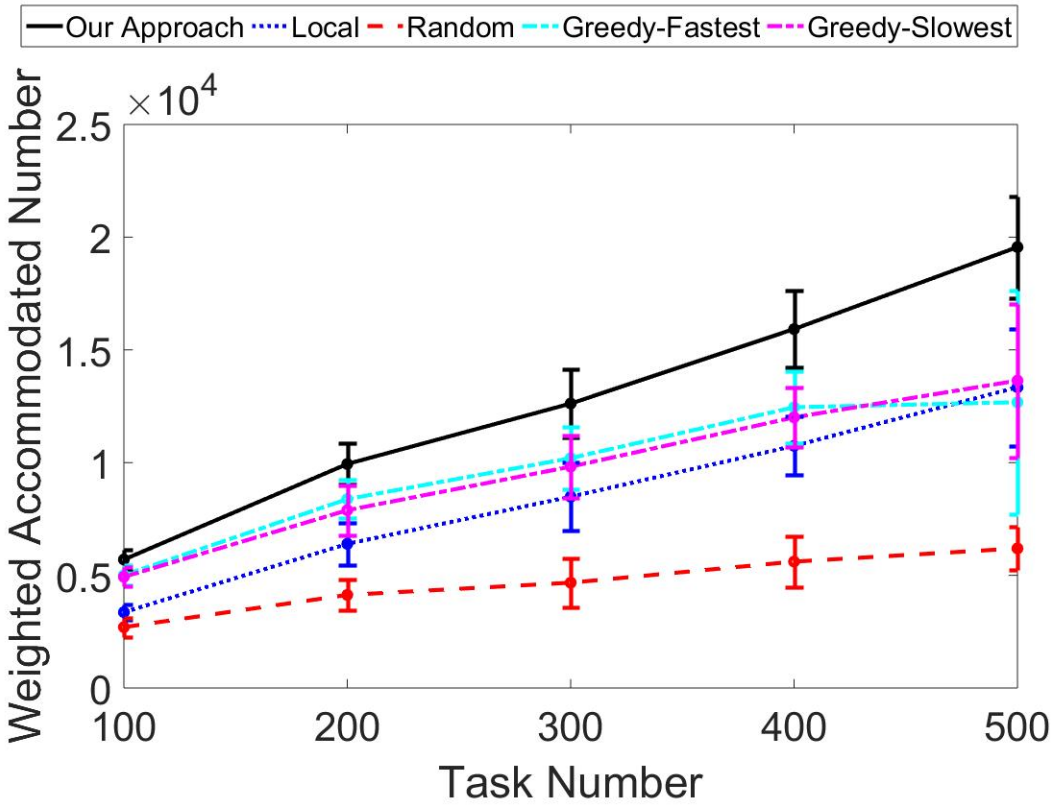

Figure 7 Weighted Accommodated Number for Simulation with Varying Task Number

The statistics in Figure 6 are the same to those in Figure 5. However, the plot shows a different view of the comparison between the performances of different approaches. The interpretation of this figure is very similar to the previous figure as they are dependent. We find that as the number of tasks gets larger, the accommodated rate gets smaller. This is because of the limitation of resources of the edge computing network. The highest accommodated rate of all the five approaches is when the number of tasks is 50 with our approach. Hence, this simulation is run in the condition that the edge computing network resources are not enough for tasks. We find that our approach always has the highest accommodated rate.

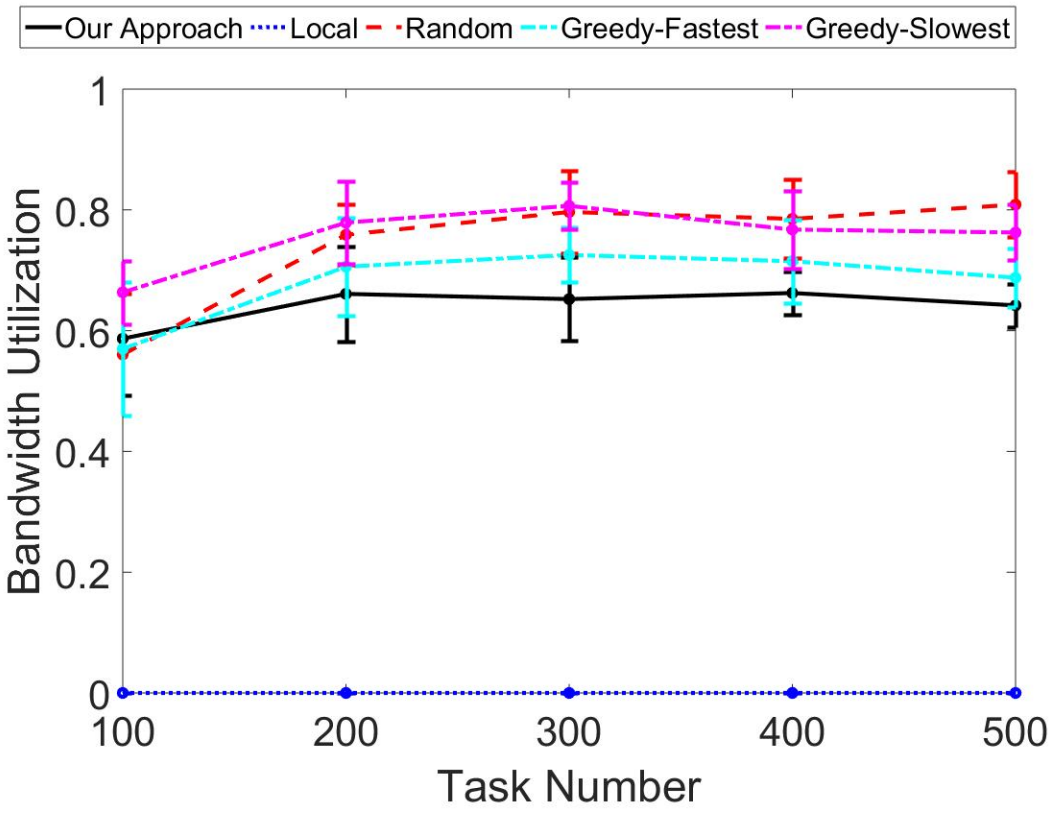

Figure 8 Bandwidth Utilization for Simulation with Varying Task Number

Figure 7 shows the weighted accommodated number of all the five approaches. The weighted accommodated number has each task's priority as the weight factor on each task's accommodation. It shows that on average, our approach achieves 53.7% more weighted accommodated number than the Local approach, and 164.4% more weighted accommodated number than the Random approach, 27.7% more weighted accommodated number than the Greedy-Fastest approach, and 29.2% more weighted accommodated number than the Greedy-Slowest approach.

As the number of tasks increases, our approach performs even better than other comparison approaches. This is because (1) the optimization objective of our approach is

to maximize the weighted accommodated number, (2) our approach is a joint optimization of computing resources and network bandwidth provisioning, so our approach could manage the tradeoff between weighted accommodated number and resource utilization much better than other approaches.

The difference between Figure 5 and Figure 7 is small and it comes from the priority of tasks. As weighted accommodation number is adjusted from the accommodation number using the priorities of tasks, Figure 5 and Figure 7 have a minor difference. Figure 7 also shows that the Random approach performs much worse than Local approach. Hence, we conclude that with a less efficient task distribution approach, the performance will be even worse than not doing any computation offloading. This proves that a good selection of task distribution approach is very important.

Figure 8 shows the average bandwidth utilization of all the five approaches. It shows that on average, the Random approach uses 15.4% more bandwidth capacity than our approach, the Greedy-Fastest approach uses 6.0% more bandwidth capacity than our approach, the Greedy-Slowest approach uses 17.9% more bandwidth capacity than our approach.

The bandwidth utilization is an important indicator of the efficiency of network resource management of different approaches. For the same tasks, the selection of different execution nodes results in different data routing for the task. Even for the same task and the same execution node, the selection of data routing can still be different. Hence, the selection of execution nodes and the data routings between tasks' access nodes and execution nodes are very important to the efficiency of bandwidth utilization.

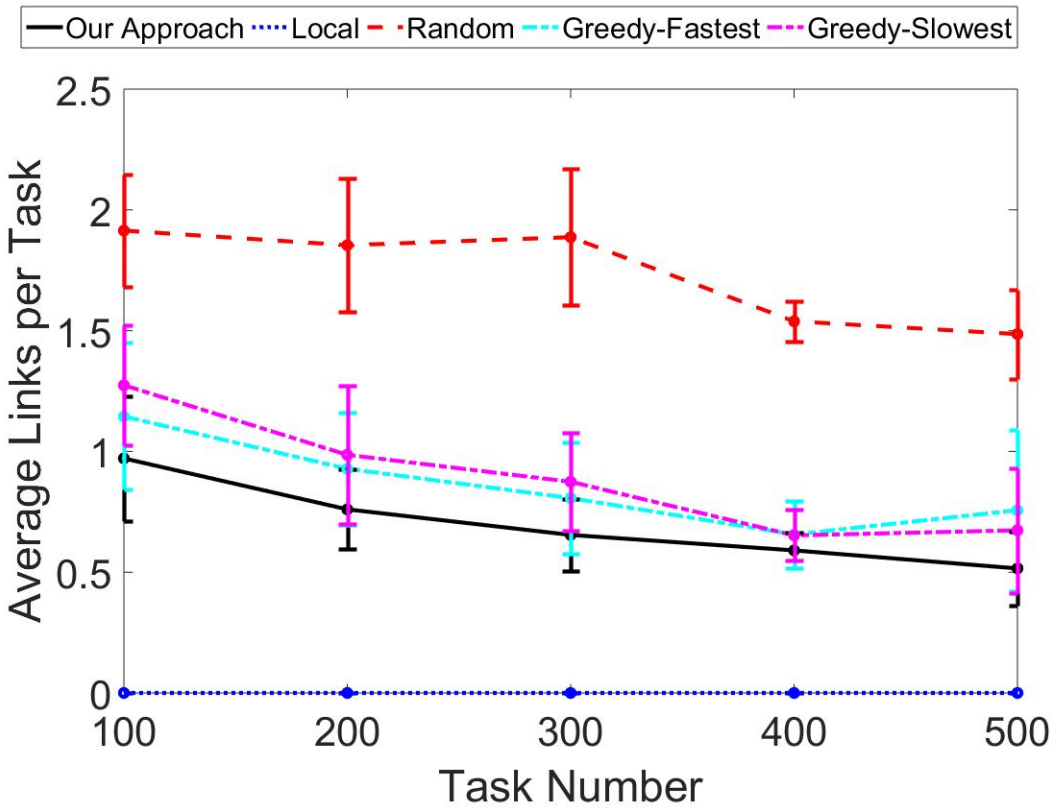

Figure 9 Average Links per Task for Simulation with Varying Task Number

From the previous three figures, we conclude that our approach outperforms all other four comparison approaches. Hence, even if the bandwidth utilization is the same, our approach is better in network resource management. In this Figure, we find that our approach uses the smallest bandwidth resources, which proves that our approach performs the best in terms of network resource management. Although the Random approach performs when the number of tasks is small, it has the worst performance overall. This is because random distribution doesn't have any management over the network bandwidth. In Figure 8, we observe that both when the number of tasks is very small and very larger, the bandwidth utilization is small. For the first situation when the number of tasks is small, low bandwidth utilization is because of the small workload. For the second situation when

the number of tasks is large, low bandwidth utilization is because of low accommodated rate, which could be found in Figure 6.

Figure 9 shows the average links per task for each approach. It shows that on average, the Random approach uses 155.7% more links per task than our approach, the Greedy-Fastest uses 24.2% more links per task than our approach, and Greedy-Slowest uses 27.1% more links per task than the Greedy-Slowest approach.

The number of average links per task is similar to the bandwidth utilization in the way that for the same task and/or the same execution node, the average links per task can be different because of the data routings in the edge computing network.

For Local approach, it does not involve task distribution among edge computing devices, so its average links per task are always zero. For the Random approach, its performance is the worst, as it doesn't have any management over task distribution and routing selection. For all the other three approaches, there are managements over the task distribution and routing selection. Hence, their performances are much better than the performance of the Random approach. For average links per task, the smaller the value is, the less overhead of the network has. For each task, if only one link is needed, the task is only transmitted once. However, if multiple links are needed, multiple amounts of data get transmitted, which will enlarge the overhead of the edge computing network. For a task, two factors make a difference in the value of average links per task. One is the selection of the task's destination node. Different approaches may select different destination edge nodes for the same task. The second factor is the selection of routes for data routing for the task. For the same task and the same destination edge computing node, there can be

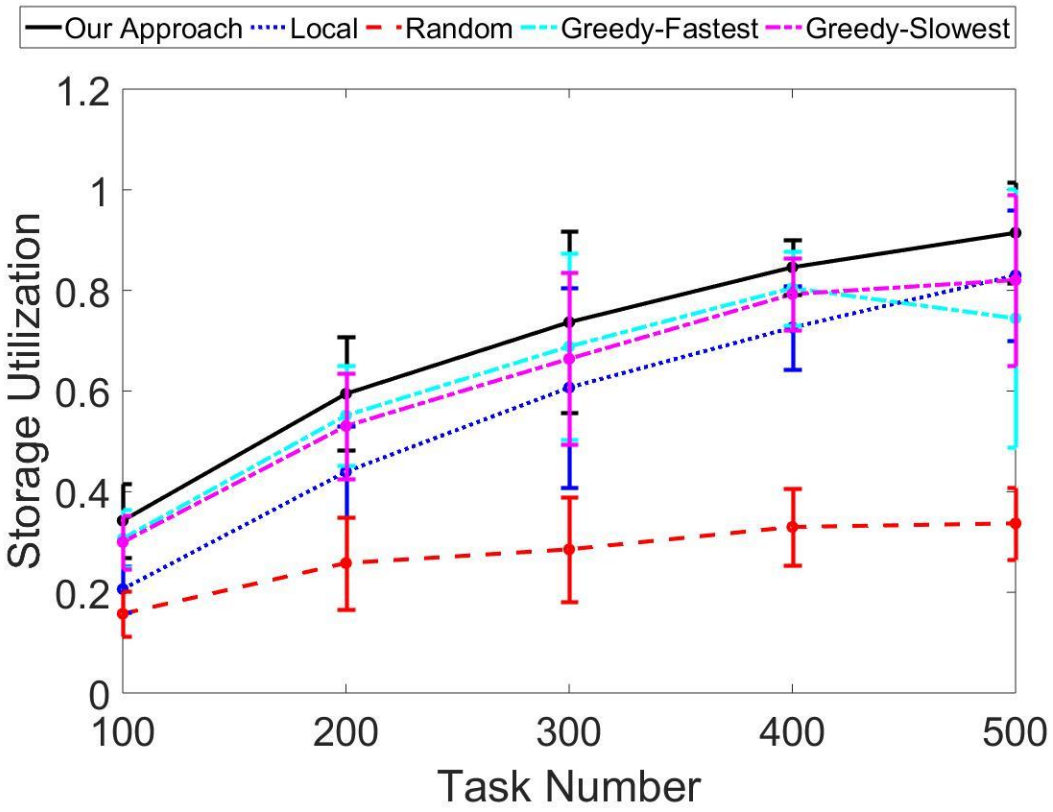

Figure 10 Storage Utilization for Simulation with Varying Task Number

different routes to transmit data from the task's access node to its destination node. Figure 9 shows that our approach has the best performance on these two factors. In addition, as the number of tasks gets larger, the average links per tasks get smaller. This is because as the number of tasks gets larger, network resources get more limited for long distance data transmission and tasks tend to be executed closer to their access nodes.

Figure 10 shows the storage utilization of all the five approaches. It shows that on average, our approach achieves 29.9% more storage than the Local approach, 146.7% more storage than the Random approach, 10.9% more storage than the Greedy-Fastest approach and 11.1% more storage than the Greedy-Slowest approach.

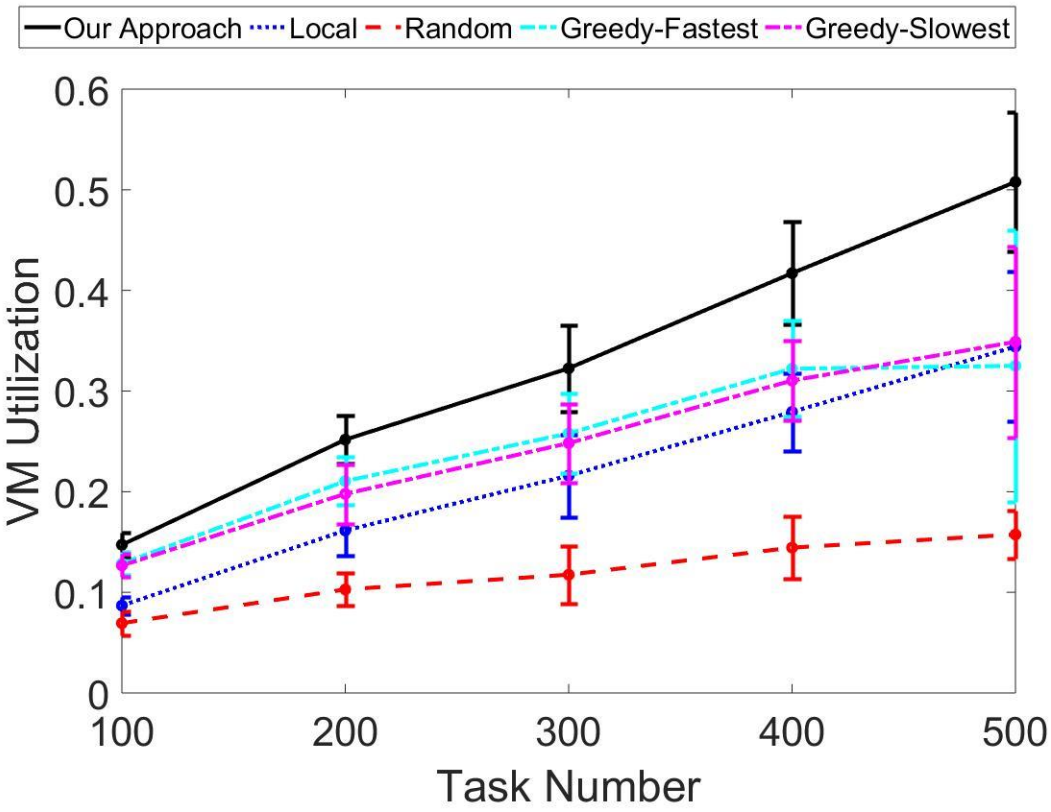

Figure 11 VM Utilization for Simulation with Varying Task Number

Storage utilization is not like bandwidth utilization. For each task, its required bandwidth resource is not fixed. The amount of the bandwidth required to finish a task depends on the route between its access edge node and execution edge node. However, for storage, for each task, it is fixed. In this dissertation research, without loss of generality, we use a task's data size as its required storage. For the Random approach, we observe that after 200 tasks, its plot becomes flat, which means saturation for storage. This limitation can be overcome by task distribution. It is clear that other four task distribution approaches do not have this limitation.

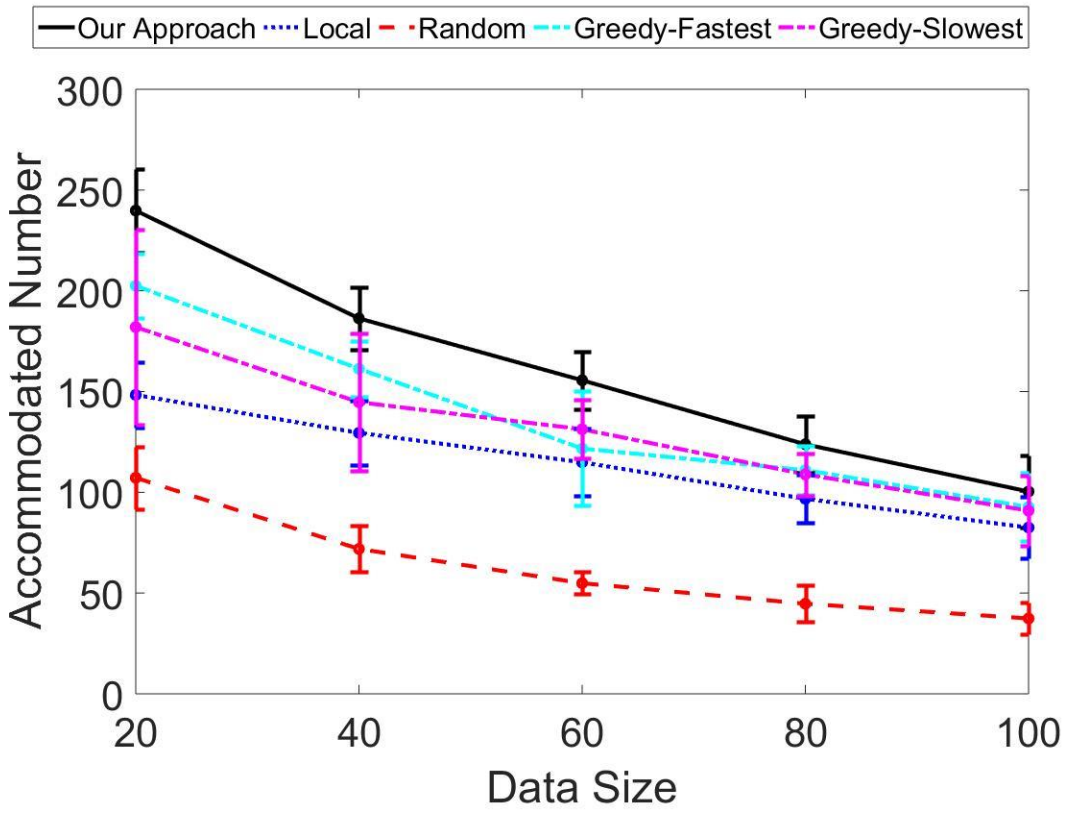

Figure 12 Accommodated Number for Simulation with Varying Data Size

Figure 11 shows the VM utilization of all the five approaches. It shows that on average, our approach utilizes 54.4% more VMs than the Local approach and 168.6% more VMs than the Random approach, 28.9% more VMs than the Greedy-Fastest approach and 30.7% more VMs than the Greedy-Slowest approach.

The utilization of VM is similar to the utilization of storage: if a task is fixed, its VM utilization and storage utilization are fixed. This is different from the bandwidth utilization, which depends on the data size of tasks, the selection of execution nodes and the data routing between the task's access edge node and execution edge node.

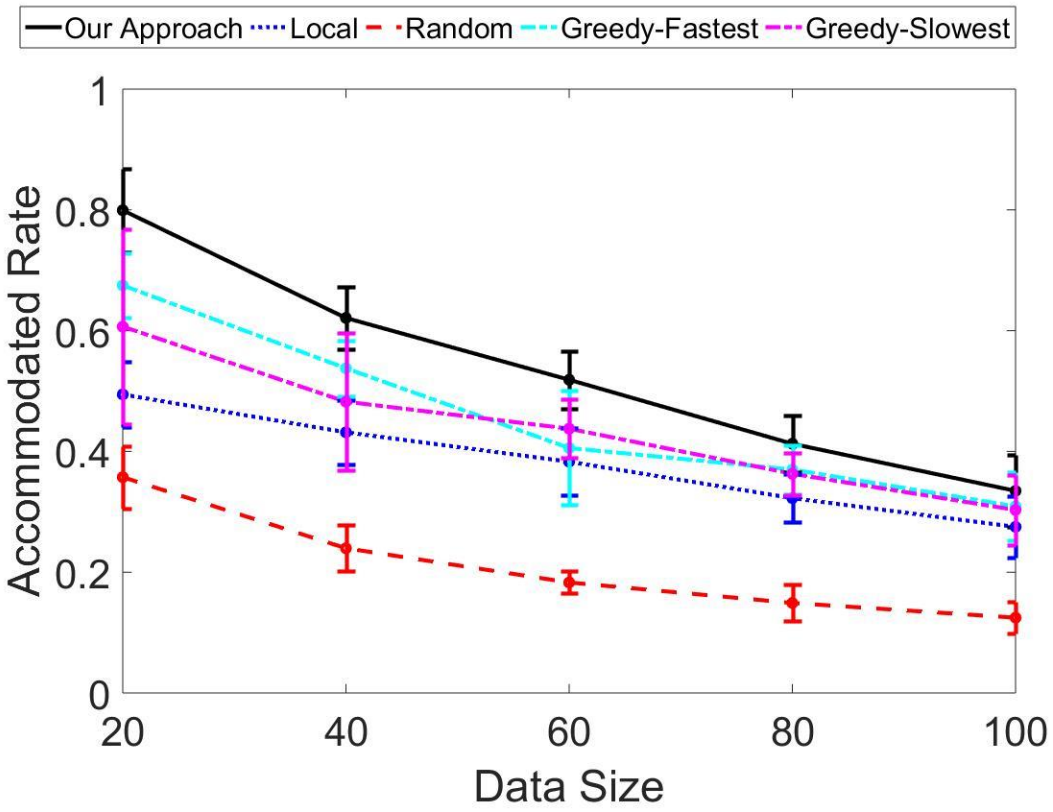

Figure 13 Accommodated Rate for Simulation with Varying Data Size

4.4 Simulation with Varying Data Size of Tasks

Same to the settings in Section 4.3 Simulation with Varying Task Number, in this set of simulation, we also have two set of edge nodes. One set contains resource-rich nodes and the other set contains recourse-poor nodes. The settings of the edge computing network and the tasks are the same to those in Section 4.3 Simulation with Varying Task Number, except the data size of tasks and the task number. We set the number of tasks to be processed to 300, and change the data size of tasks from 20 MB to 100 MB, with a step of 20 MB, to examine the effect of the data size of tasks.

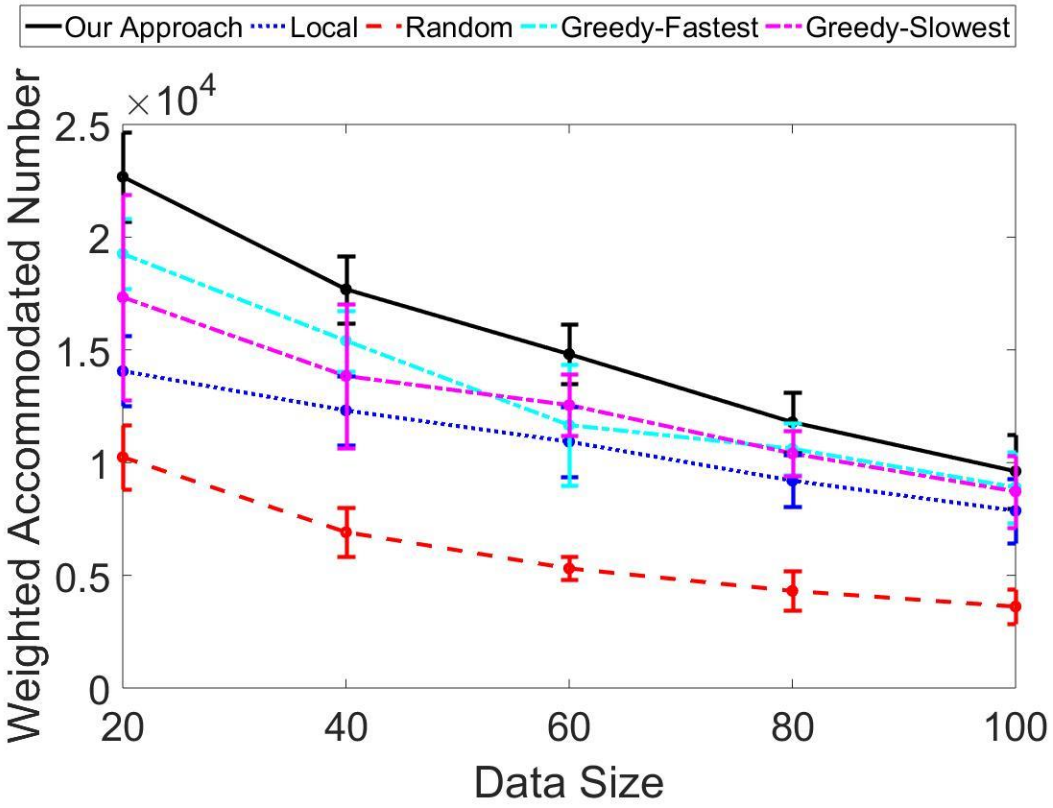

Figure 14 Weighted Accommodation Number for Simulation with Varying Data Size

Figure 12 shows the accommodation number of all the five approaches. It shows that on average, our approach achieves 38.1% more accommodation number than the Local approach, 162.3% more accommodation number than the Random approach, 16.3% more accommodation number than the Greedy-Fastest approach and 20.6% more accommodation number than the Greedy-Slowest approach. The coefficients of variation of our approach for different data size (from 20MB to 100 MB with the step of 20MB) are 8.7%, 8.4%, 9.2%, 12.0%, and 17.7%.

As the number of total tasks is 300, no approach accommodates all tasks. This means the edge computing network resources, which include both edge nodes resources and edge network bandwidth resources, are scarce for tasks' computation and transmission.

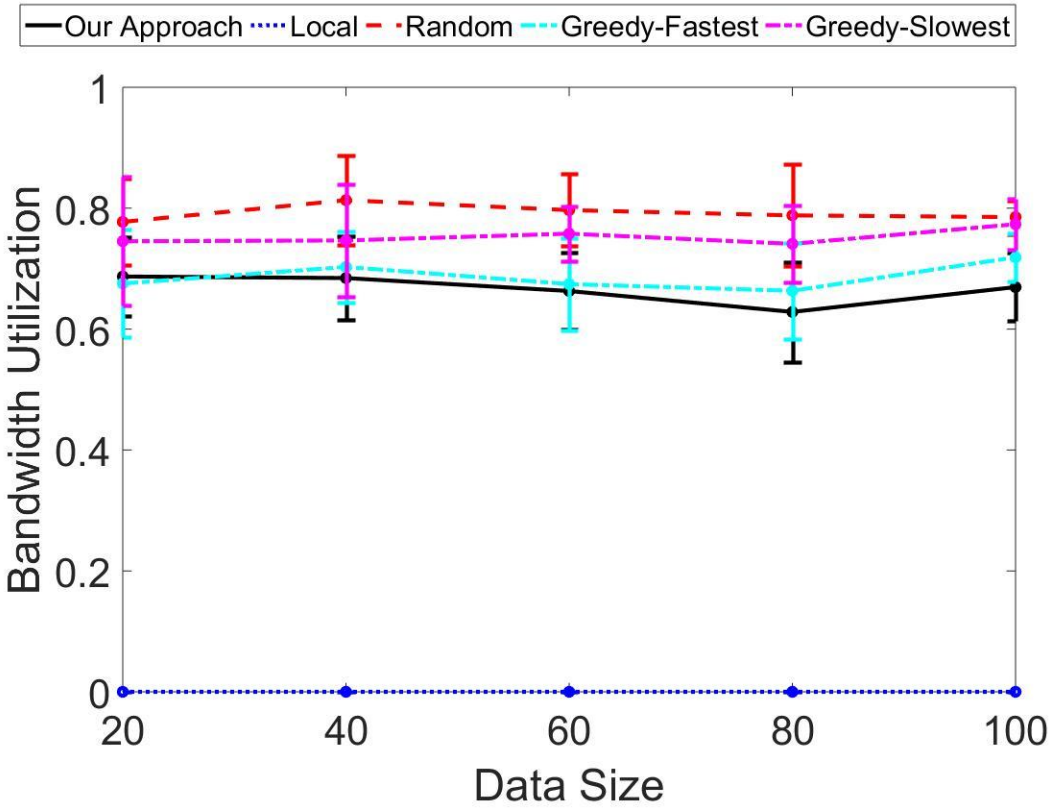

Figure 15 Bandwidth Utilization for Simulation with Varying Data Size

Our approach always performs better than other comparison approaches. The performance of the Random approach is always worse than the Local approach, hence, we conclude that it is better not distributing tasks, rather than distributing tasks randomly.

Figure 13 has the same statistics as Figure 12, as in this set of simulations, the number of tasks is fixed.

Figure 14 shows the weighted accommodated number of all the five approaches. The weighted accommodated number has each task's priority as the weight factor on each task's accommodation. It shows that on average, our approach achieves 38.2% more weighted accommodated number than the Local approach, and 159.3% more weighted

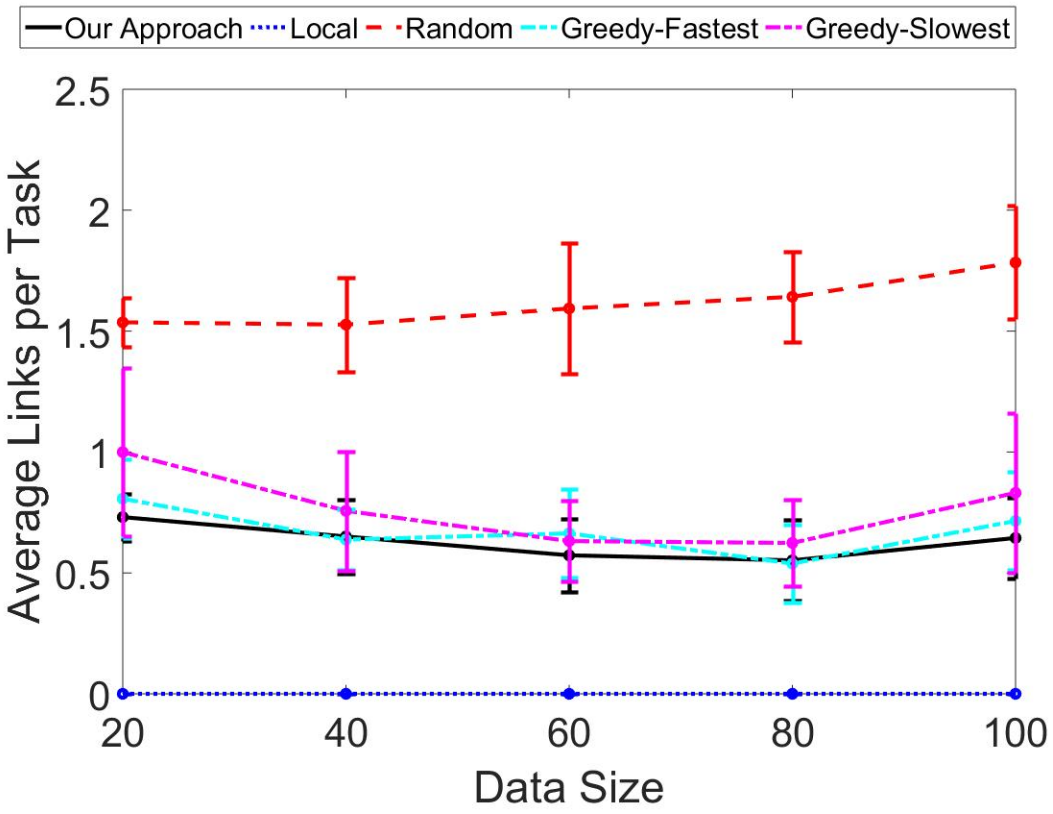

Figure 16 Average Links per Tasks for Simulation with Varying Data Size

accommodated number than the Random approach, 15.7% more weighted accommodated number than the Greedy-Fastest approach, and 20.0% more weighted accommodated number than the Greedy-Slowest approach.

Figure 15 shows the average bandwidth utilization of all the five approaches. It shows that on average, the Random approach uses 19.0% more bandwidth capacity than our approach, the Greedy-Fastest approach uses 3.1% more bandwidth capacity than our approach, the Greedy-Slowest approach uses 13.1% more bandwidth capacity than our approach.

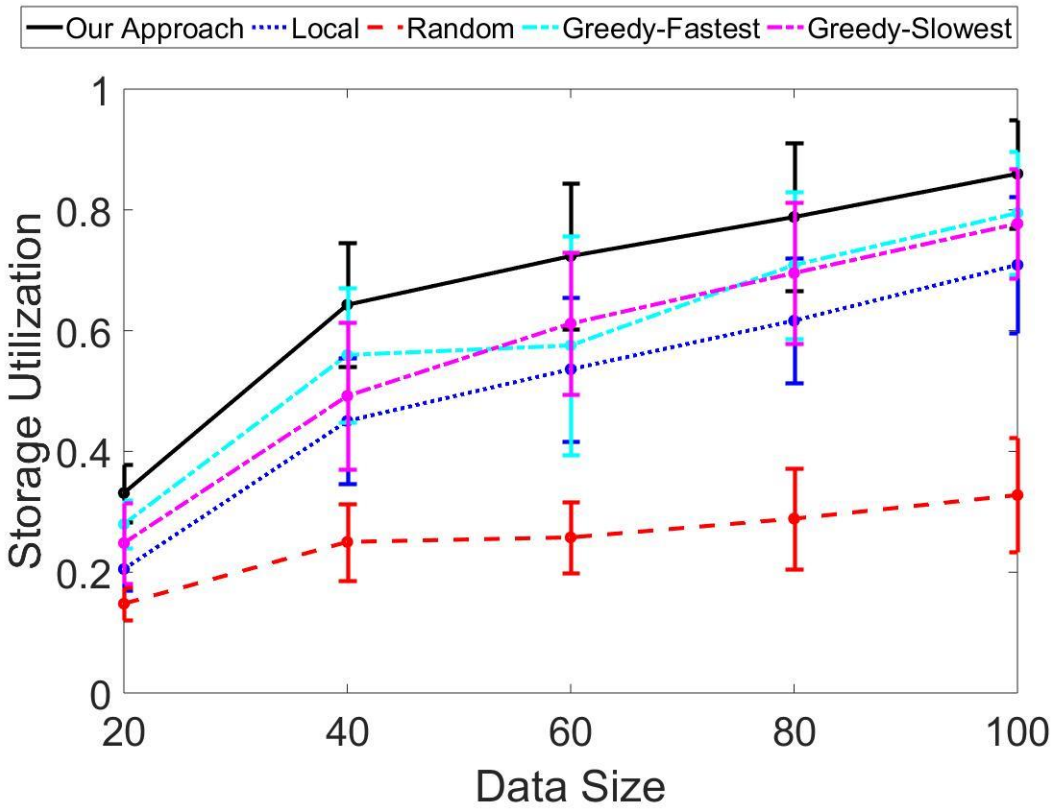

Figure 17 Storage Utilization for Simulation with Varying Data Size

In Figure 15, we observe that both when the size of data is very small and very large, the bandwidth utilization becomes large. For the first situation when the size of data is small, high bandwidth utilization is because of the high accommodated rate. As data size is small, the overhead of network transmission is small, hence, more tasks are accommodated and more network resources are used. For the second situation when the size of data is large, high bandwidth utilization is because of the large workload of tasks. Although the accommodated rate is low when the data size is large, the total data size needs to be transmitted increases as the data size of single tasks increases.

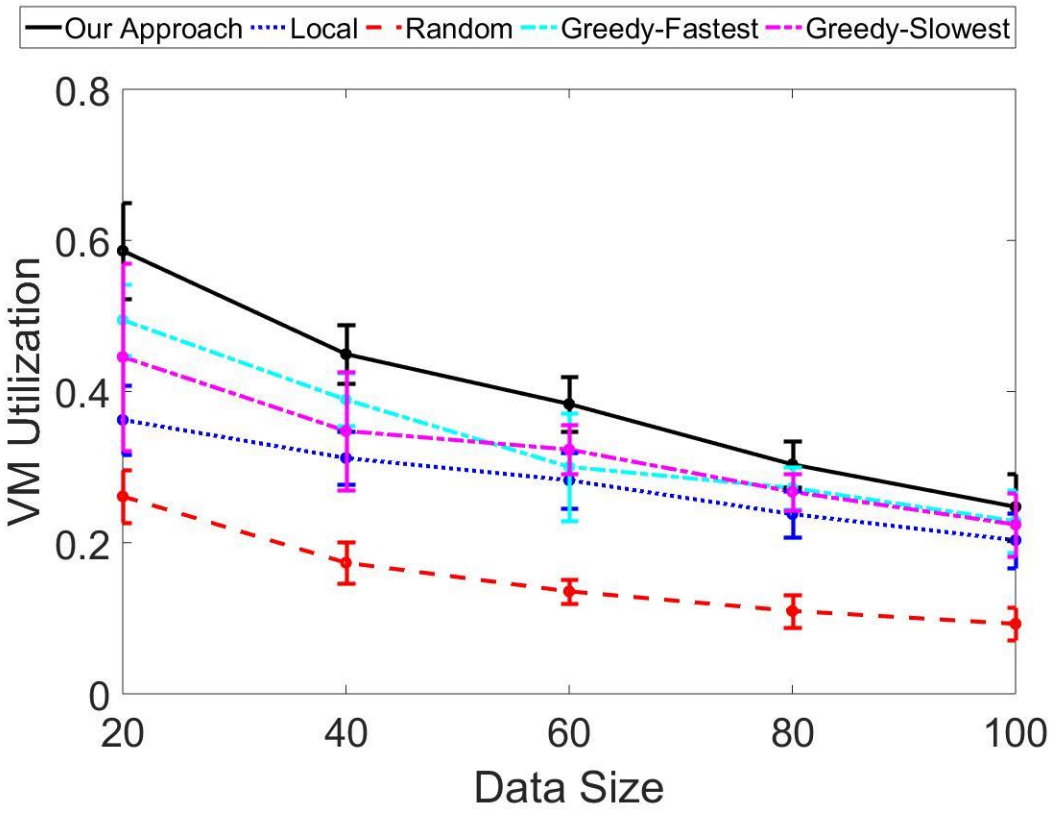

Figure 18 VM Utilization for Simulation with Varying Data Size

Figure 16 shows the average links per task for each approach. It shows that on average, the Random approach uses 159.5% more links per task than our approach, the Greedy-Fastest uses 6.6% more links per task than our approach, and Greedy-Slowest uses 21.1% more links per task than the Greedy-Slowest approach.

Figure 17 shows the storage utilization of all the five approaches. It shows that on average, our approach achieves 37.6% more storage than the Local approach, 159.4% more storage than the Random approach, 15.7% more storage than the Greedy-Fastest approach and 21.3% more storage than the Greedy-Slowest approach.

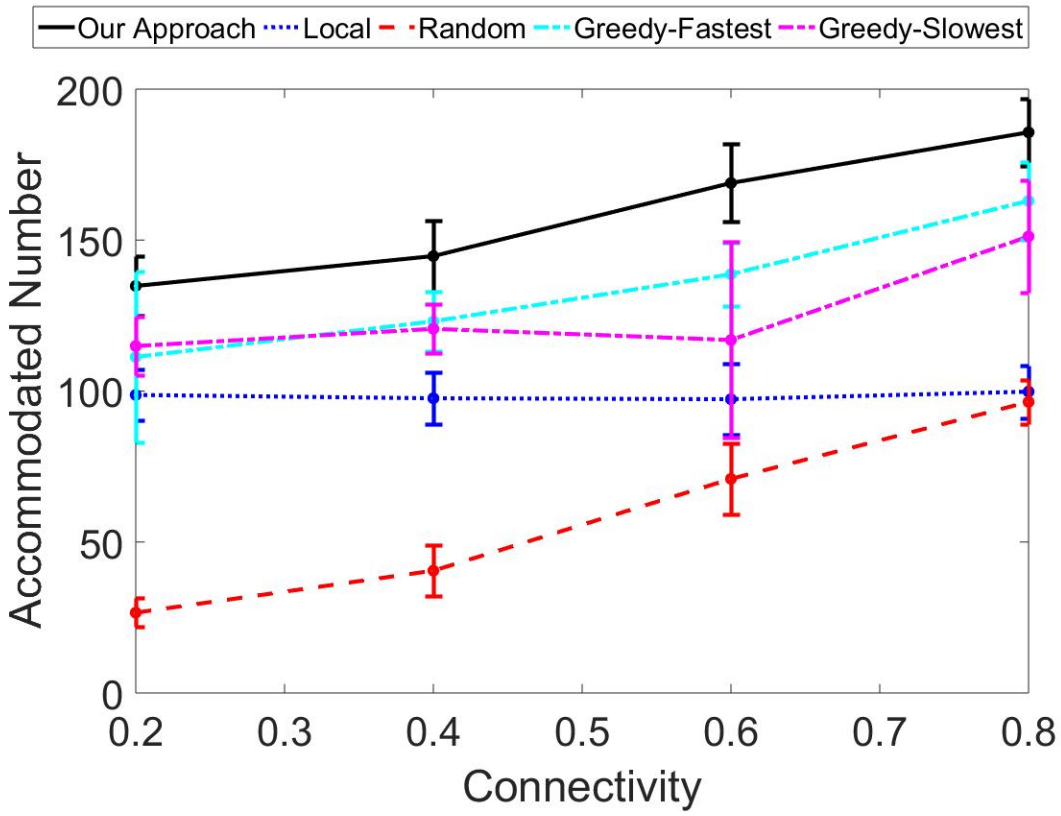

Figure 19 Accommodated Number for Simulation with Varying Network Connectivity

Figure 18 shows the VM utilization of all the five approaches. It shows that on average, our approach utilizes 38.2% more VMs than the Local approach and 162.1% more VMs than the Random approach, 16.3% more VMs than the Greedy-Fastest approach and 20.7% more VMs than the Greedy-Slowest approach.

4.5 Simulation with Varying Connectivity of the Edge Computing Network

Same to the settings in Section 4.3 Simulation with Varying Task Number, in this set of simulation, we also have two set of edge nodes. One set contains resource-rich nodes and the other set contains recourse-poor nodes. The settings of the edge computing network

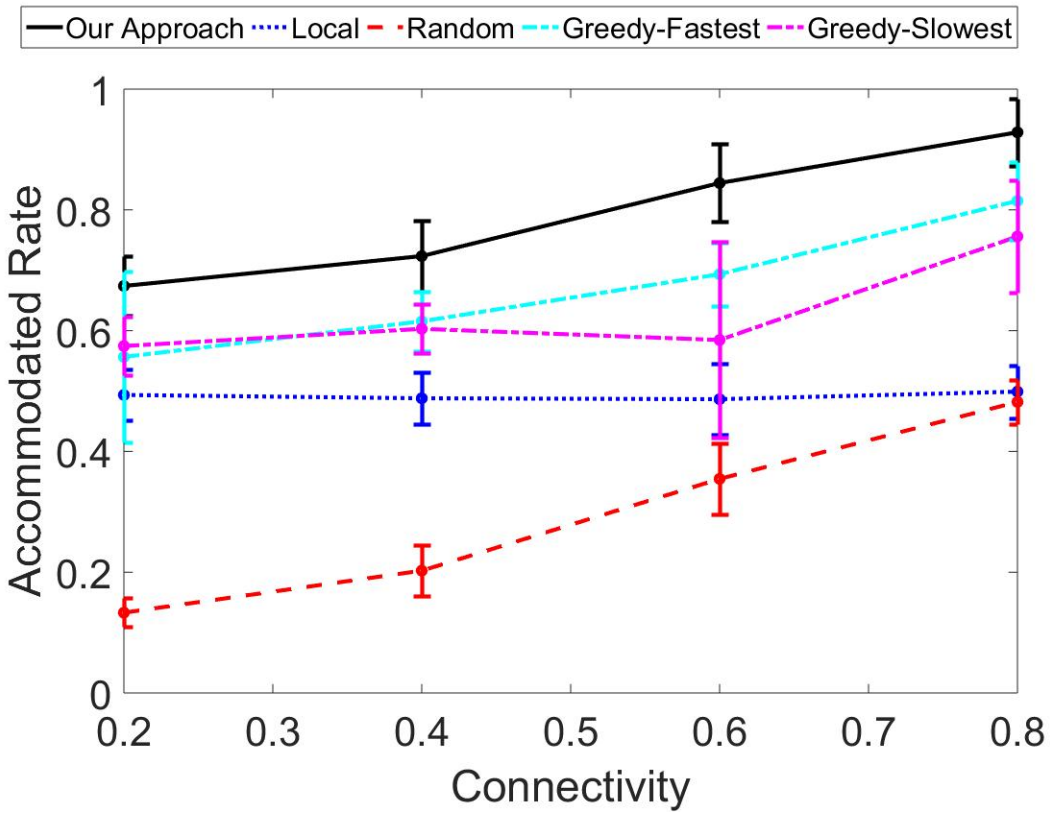

Figure 20 Accommodated Rate for Simulation with Varying Network Connectivity

and the tasks are the same to those in Section 4.3 Simulation with Varying Task Number, except the network connectivity and the task number. In this simulation, we set the number of edge nodes to 15. We set the number of tasks to be processed to 200, the data size of each task to 50 MB and change the two parameters of Waxman model (α and β) from 0.2 to 0.8, with a step of 0.2, to examine the effect of the connectivity of the edge computing network.

Figure 19 shows the accommodated number of all the five approaches. It shows that on average, our approach achieves 61.1% more accommodated number than the Local approach, 223.7% more accommodated number than the Random approach, 18.6% more

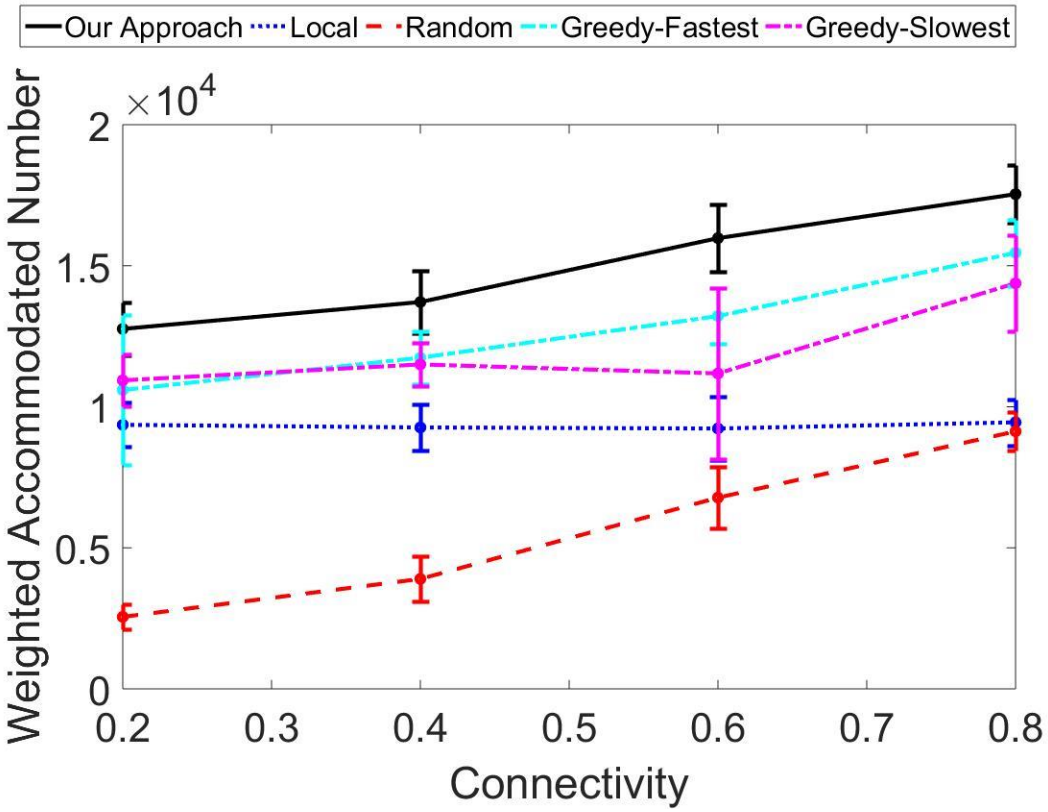

Figure 21 Weighted Accommodated Number for Simulation with Varying Network Connectivity

accommodated number than the Greedy-Fastest approach and 26.2% more accommodated number than the Greedy-Slowest approach. The coefficients of variation of our approach for different value of the two parameters of the Waxman model (from 0.2 to 0.8 with the step of 0.2) are 7.4%, 8.1%, 7.6%, and 5.9%.

First of all, as the Local approach doesn't involve task distribution, the difference in network connectivity doesn't have any effects on the performance of the Local approach. In Figure 16, the plot of the Local approach is a very flat line. All the other four approaches reply on the network to distribution tasks, hence, we see a clear trend that as the network connectivity gets better, the performances of all the other four approaches get better. The

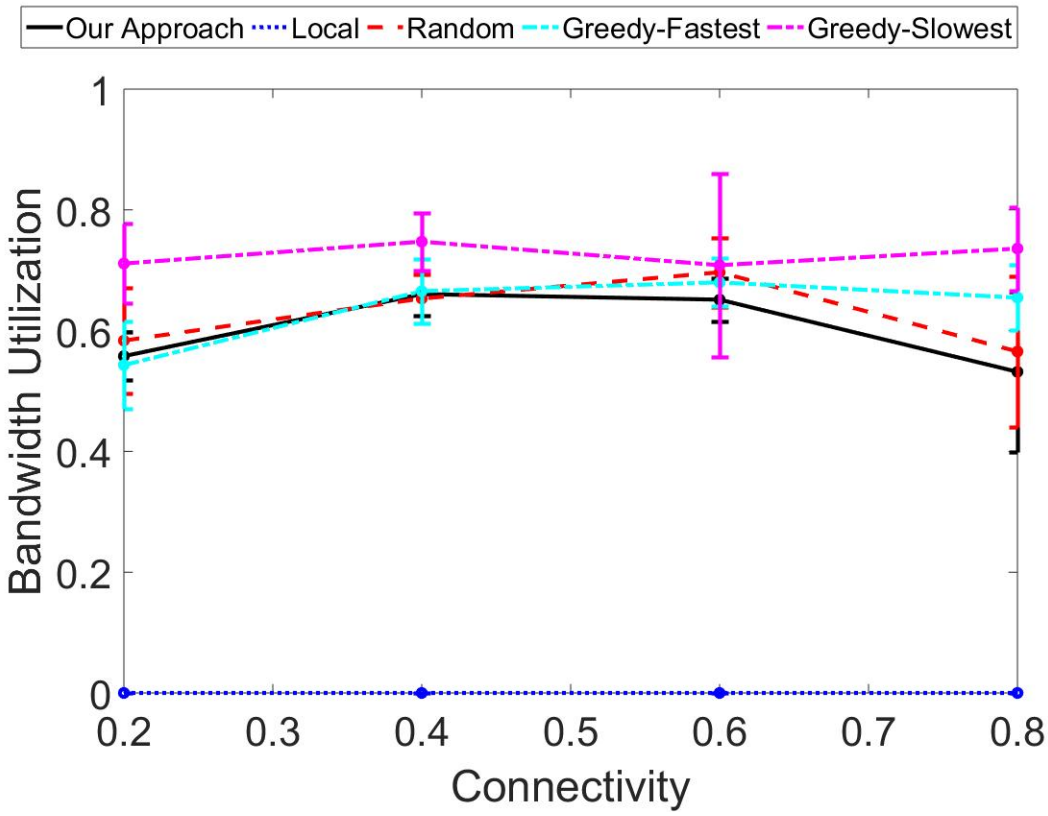

Figure 22 Bandwidth Utilization for Simulation with Varying Network Connectivity

random approach always performs worse than the Local approach. Hence, we conclude that it is better not to distribute tasks, rather than randomly distributing tasks. The performance of Greed-Fastest becomes slightly better than that of Greedy-Slowest, hence, we conclude that Greedy-Fastest relies more on the connectivity of the network. Greedy-Fastest approach always selects the fastest execution node for each task, hence, it is reasonable that the Greedy-Fastest approach consumes more bandwidth to expedite the transmission time of task completion. Hence, as the connectivity of the network grows, more bandwidth resources could be provided, and the performance of Greedy-Fastest gets better. Our approach always performs better than all other approaches.

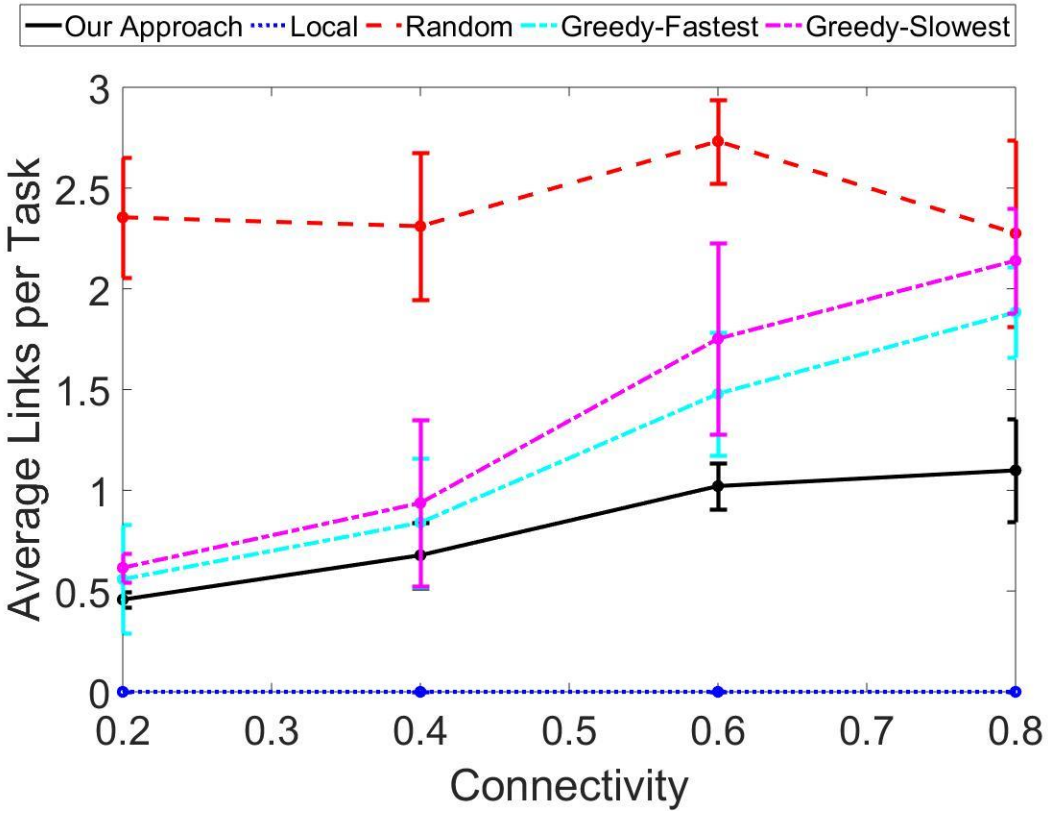

Figure 23 Average Links per Task for Simulation with Varying Network Connectivity

Figure 20 has the same statistics as Figure 19, as in this set of simulations, the number of tasks is fixed.

Figure 21 shows the weighted accommodated number of all the five approaches. The weighted accommodated number has each task's priority as the weight factor on each task's accommodation. It shows that on average, our approach achieves 60.8% more weighted accommodated number than the Local approach, and 220.0% more weighted accommodated number than the Random approach, 17.9% more weighted accommodated number than the Greedy-Fastest approach, and 25.2% more weighted accommodated number than the Greedy-Slowest approach.

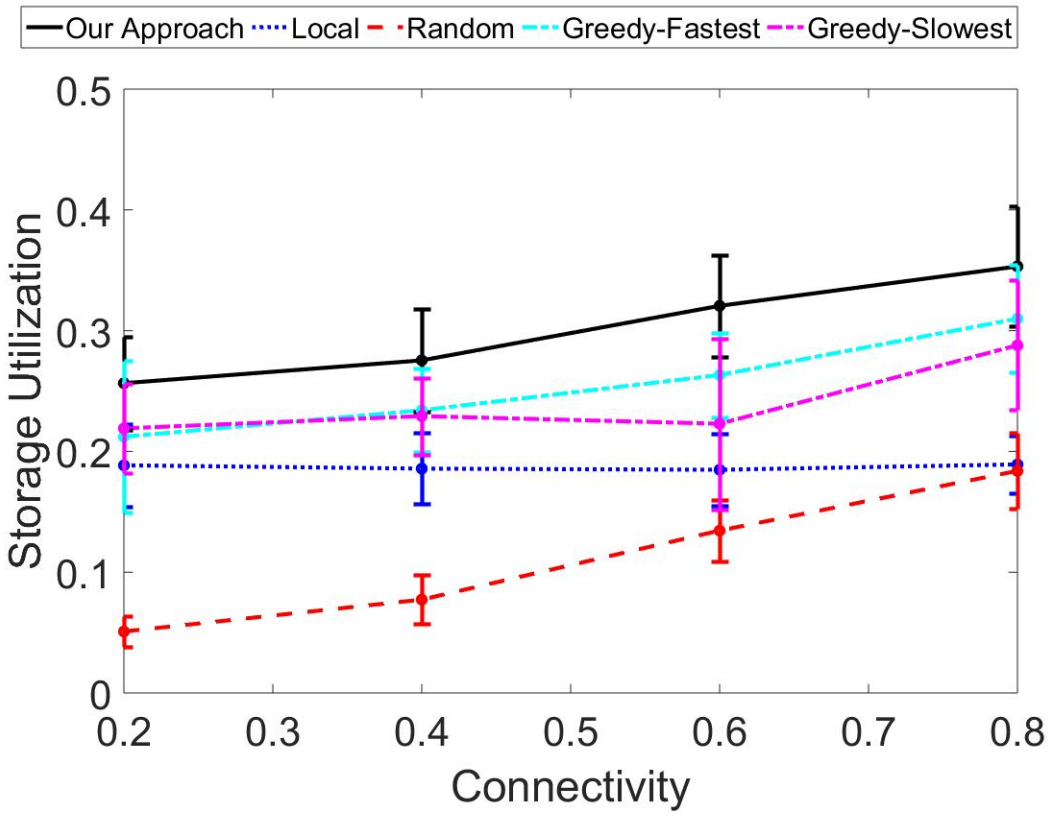

Figure 24 Storage Utilization for Simulation with Varying Network Connectivity

Figure 22 shows the average bandwidth utilization of all the five approaches. It shows that on average, the Random approach uses 4.2% more bandwidth capacity than our approach, the Greedy-Fastest approach uses 6.4% more bandwidth capacity than our approach, the Greedy-Slowest approach uses 22.0% more bandwidth capacity than our approach.

In Figure 22, we observe that both when the network connectivity is very low and very high, the bandwidth utilization is small. For the first situation when the network connectivity is low, the resource sharing within the edge computing network is difficult as there are few links available and the total network capacity is low. For the second situation

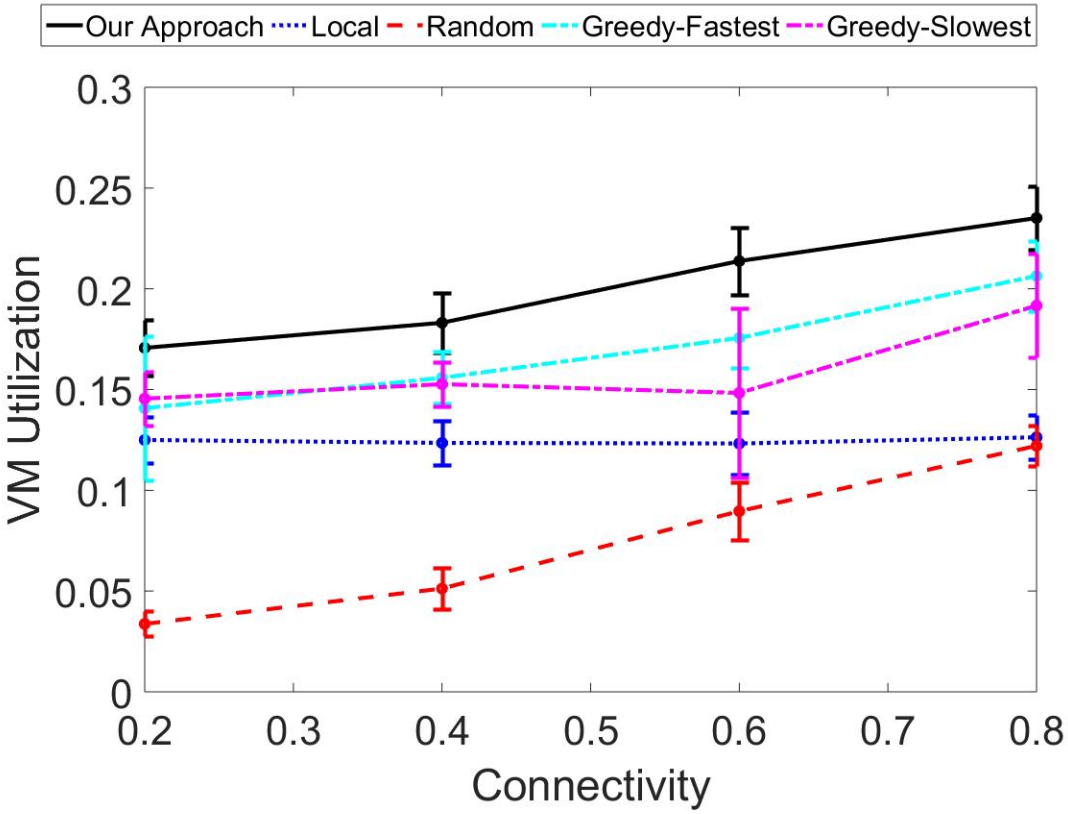

Figure 25 VM Utilization for Simulation with Varying Network Connectivity

when the network connectivity is high, low bandwidth utilization is because of large network capacity. When the Waxman parameters reach 0.8, there are 80 links between the 15 edge nodes, which forms a well-connected network.

Figure 23 shows the average links per task for each approach. It shows that on average, the Random approach uses 232.5% more links per task than our approach, the Greedy-Fastest uses 40.6% more links per task than our approach, and Greedy-Slowest uses 60.0% more links per task than the Greedy-Slowest approach.

As the network connectivity gets better, the number of average links per task gets larger. This is because more network capacity makes it easier for data transmission.

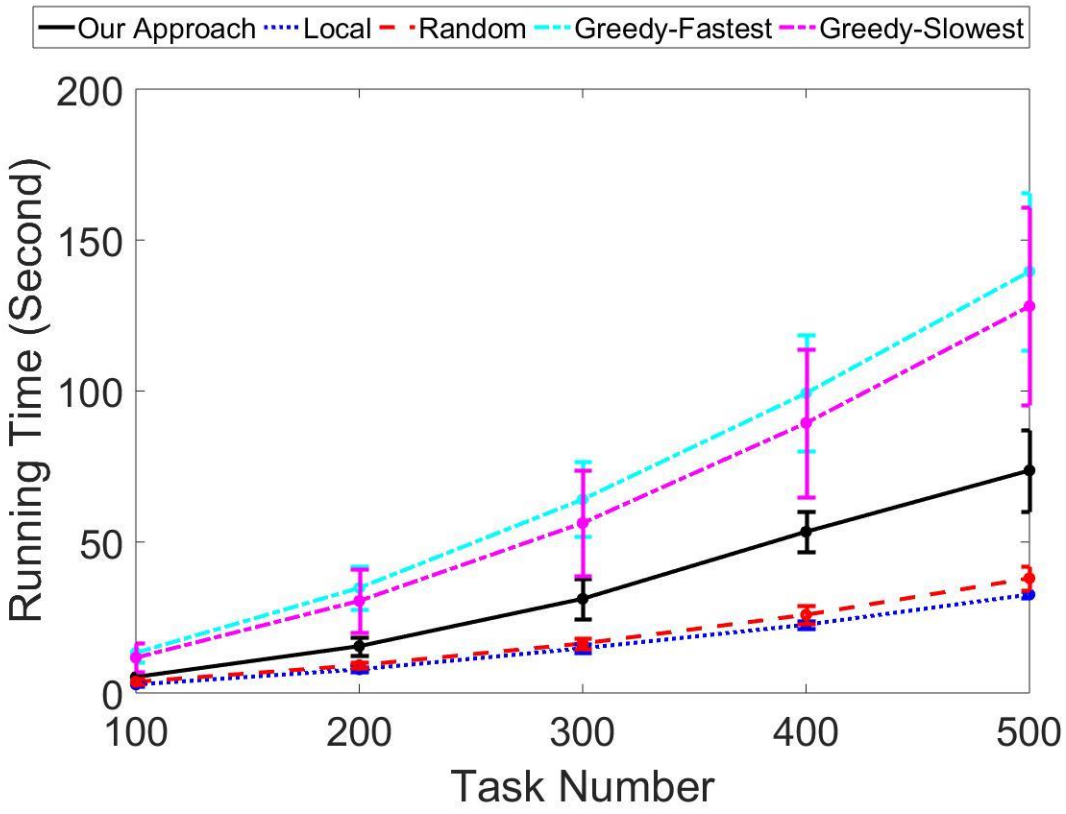

Figure 26 Running Time for Simulation with Varying Task Number

Figure 24 shows the storage utilization of all the five approaches. It shows that on average, our approach achieves 61.1% more storage than the Local approach, 222.7% more storage than the Random approach, 18.6% more storage than the Greedy-Fastest approach and 25.9% more storage than the Greedy-Slowest approach.

Figure 25 shows the VM utilization of all the five approaches. It shows that on average, our approach utilizes 61.1% more VMs than the Local approach and 223.7% more VMs than the Random approach, 18.6% more VMs than the Greedy-Fastest approach and 26.0% more VMs than the Greedy-Slowest approach.

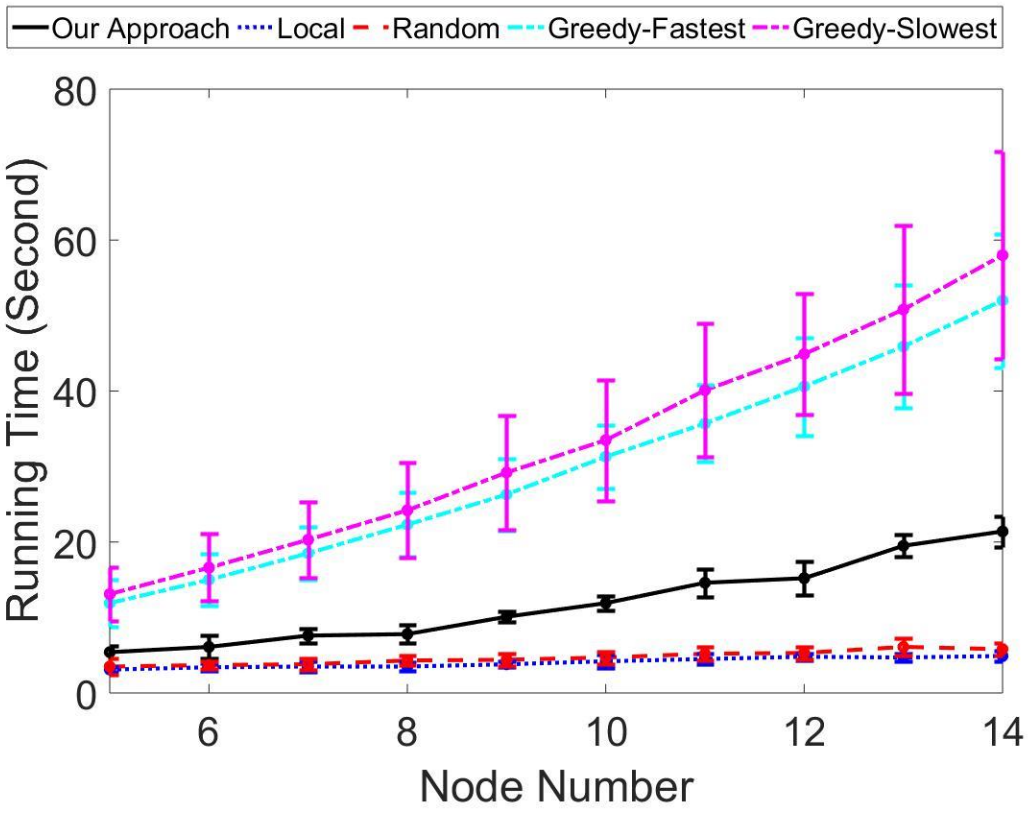

Figure 27 Running Time for Simulation with Varying Node Number

4.6 Running Time Performance

According to the time complexity analysis of our approach in Section 3.2.2 Generation of Task Distribution Solution, the running time of our approach is related to (1) the task number, (2) the node number and (3) the size of the program to be solved using the IBM CPLEX linear programming solver. The third component (the size of the program), is related to three factors, the number of tasks, the number of edge nodes, and the number of edge links. Hence, in this section, we show the running time performance of our

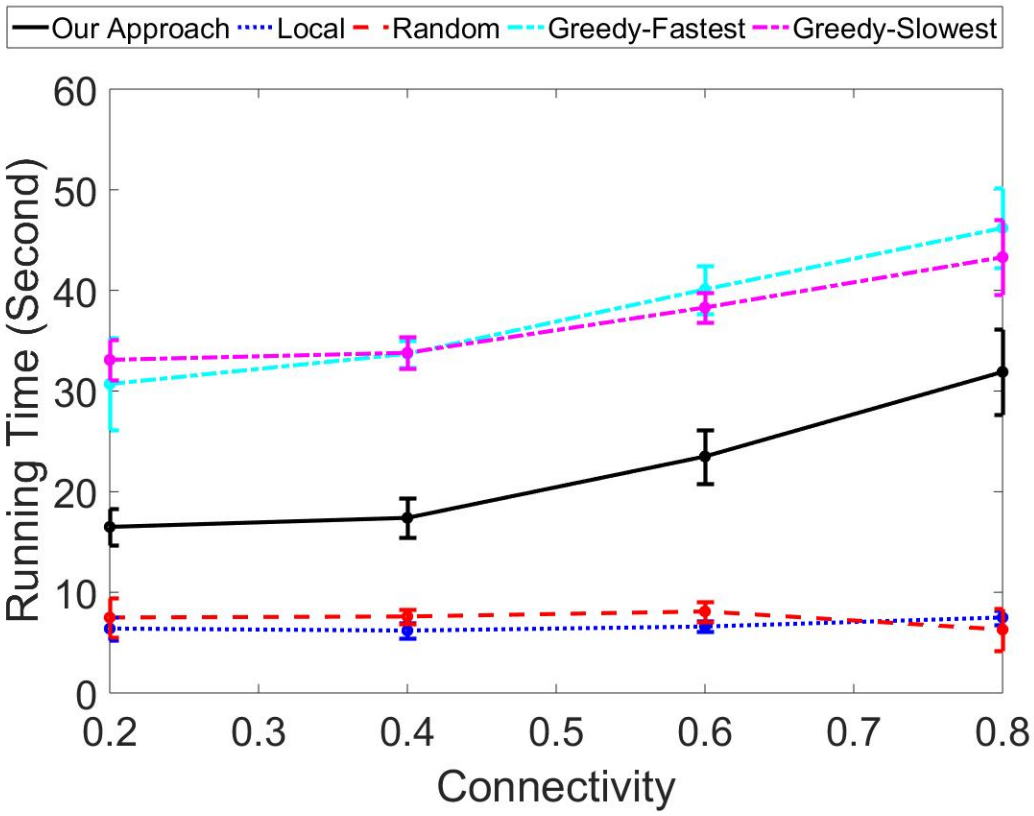

Figure 28 Running Time for Simulation with Varying Network Connectivity

approach with varying number of tasks, varying number of edge nodes and varying network connectivity.

Figure 26 shows the running time of our approach with varying task number. The node number is set to 5. The two parameters in the Waxman model are set to 0.2. The task number is set from 100 to 500, with a step of 100. It shows that as the task number increases, the running time also increases. This is because as the task number increases, the number of task distribution variables increases proportionally. It shows that on average, the Local approach runs 52.6% faster than our approach, and the Random approach runs 43.6% faster than our approach. Our approach runs 52.1% faster than the Greedy-Fastest approach and

46.2% faster than the Greedy-Slowest approach. The coefficients of variation of our approach for different task number (from 100 to 500 with the step of 100) are 17.9%, 20.0%, 21.7%, 12.3%, and 18.3%.

Figure 27 shows the running time of our approach with varying edge node number. The task number is set to 100. The two parameters in the Waxman model are set to 0.2. The node number is set from 5 to 15, with a step of 1. It shows that on average, the Local approach runs 61.4% faster than our approach, and the Random approach runs 55.8% faster than our approach. Our approach runs 60.0% faster than the Greedy-Fastest approach and 63.7% faster than the Greedy-Slowest approach. It shows that as the node number increases, the running time also increase. This is because as the node number increases, both the numbers of task distribution variables and flow variables increase proportionally. The coefficients of variation of our approach for different node number (from 1 to 14 with the step of 1) are 15.6%, 25.0%, 12.7%, 15.8%, 7.3%, 8.4%, 12.6%, 14.5%, 7.4%, and 9.7%.

Figure 28 shows the running time of our approach with varying connectivity of the network, which reflects the varying link number in the network. The task number is set to 100 and node number is set to 10. The two parameters in the Waxman model are set from 0.2 to 0.8, with a step of 0.2. It shows that on average, the Local approach runs 68.5% faster than our approach, and the Random approach runs 64.2% faster than our approach. Our approach runs 41.7% faster than the Greedy-Fastest approach and 40.9% faster than the Greedy-Slowest approach. The coefficients of variation of our approach for different values of the two parameters of Waxman model (from 0.2 to 0.8 with the step of 0.2) are 11.2%, 11.2%, 11.4%, and 13.4%.

For the running time performance, the Local and Random approaches run fasters than other three approaches, which is because they have much fewer iterations of linear programming in Step 2) of the task distribution process. Our approach performs better than the two greedy based approaches. The reason is that through the optimization of Algorithm 1, our approach can (1) reduce the number of variables that are infeasible to the objective function, and (2) reduce the number of iterations of linear programming, through the sequential rounding in Step 2) of the task distribution process.

As the goal of our approach is to maximize the number of accommodated tasks, in this dissertation research, different approaches are evaluated by the number of accommodated tasks, not the running time performance.

4.7 Summary of the Simulations

We find that in all the three simulations, our approach outperforms both the baseline approaches, in terms of the number of accommodated tasks. We also find that in the following circumstances, the benefit of our approach is even larger: (1) the overall workload in the edge computing network is high, as shown in Figure 5, when the number of tasks is large, and as shown in Figure 12, when data size of tasks is large; (2) the connectivity of edge network is high, as shown in Figure 19.

These simulation results show that our approach has a much better efficiency in terms of bandwidth utilization and network flows routing. For a task, the network bandwidth resource it requires is not like the parameter of storage or virtual machines, which are fixed. For each task, the bandwidth resource reservation is dynamic, which

depends on the selection of execution nodes and the data routings between the task's access node and its execution node. Through the figures, we find that our approach is more efficient in (1) selecting the execution nodes and (2) routing data from the task's access node and execution node to reduce unnecessary data transmission in the network. This advantage of our approach brings two benefits to the whole system, which are faster execution of tasks and less network bandwidth resources utilization.

The following is the summary of the comparisons between all the five approaches:

(1) Local:
   a. Does not consume network bandwidth
   b. Has average performance in terms of both task accommodation number and weighted task accommodation number
   c. Has average performance in terms of resource utilization

(2) Random:
   a. Has the worst performance in terms of both task accommodation number and weighted task accommodation number
   b. Consumes the most amount of bandwidth, and the most average links for each task, which are the bottleneck of Random as its performance is limited by the network resources.

(3) Greedy-Fastest
   a. Preforms better than greedy-slowest when resources are enough, or the total workload is small, as the Greedy-Fastest approach tends to accommodate maximum resource to tasks

(4) Greedy-Slowest

a. Performs better than Greedy-Fastest overall, as Greedy-Slowest approach tends to finish tasks at their deadlines, which consumes less energy than the Greedy-Fastest approach.

b. Performs well when resources are limited or the total workload is large

(5) Our approach:

a. Has the best performance in terms of both task accommodation number and weighted task accommodation number

b. Has the best performance in terms of resource utilization

c. Has acceptable variation, which can be observed in Figure 5, 12, and 19.

The reason that our approach has the best performance with the least network resource utilization is our joint optimization of both task distribution and data routing. The optimization of task distribution results in high performance in terms of the number of accommodated tasks. The optimization of data routing results in the low network resource utilization.

Chapter 5

CONCLUSION AND FUTURE RESEARCH

5.1 Conclusion

In this dissertation research, our approach to task distribution in edge computing networks for IoT applications is presented. The objective of our approach is to maximize the number of tasks that can be accommodated in the edge computing network with the consideration of the priorities of tasks. Our approach guarantees to satisfy all accommodated tasks' QoS requirements, which include the completion deadline and security requirement. Two research problems have been addressed. (1) How to distribute a set of tasks to a set of interconnected edge computing devices for the goal of maximizing the number of accommodated tasks with the consideration of the priorities of tasks. (2) How to efficiently arrange data routing in the edge computing network to transmit the data of the accommodated tasks from their access edge computing devices to their assigned execution edge computing devices.

In Chapter 3, we presented our approach to distributing tasks within the edge computing network. The problem was formulated as a joint optimization of task distribution and data routing. The problem is originally formulated to a mixed integer non-linear program (MINLP), which is NP-hard. We first linearized the MINLP with the requirement that all accommodated tasks are completed exactly on the deadline. Then we reduced the problem size by removing unfeasible variables using the security requirements of tasks. At last, we applied our proposed TDDR algorithm to sequentially relax, round and

validate the solutions of variables until we get the final task distribution and data routing solution. In Chapter 4, we conducted three sets of simulations on our approach and four other comparison approaches. The four comparison approaches are local execution approach, random distribution approach, and two greedy-based distribution approaches. The three different sets of simulations were designed to investigate different factors on the performance of our approach. They are simulations with varying number of tasks, simulations with varying data size of tasks, and simulations with varying connectivity of networks. The simulation results showed that our approach can greatly improve the performance while utilizing less network bandwidth resources compared to the four comparison approaches. The factors being examined are the workload of tasks, the data size of tasks and the connectivity of edge computing networks.

5.2 Future Research Directions

There are two future research directions, which are (1) exploring other optimization objectives, (2) including dependencies among tasks.

In this dissertation research, the optimization objective is to maximize the number of tasks that can be accommodated in the edge computing network with the consideration of the priorities of tasks. Other different objectives of optimization can be explored. For example, minimizing average/shortest/longest tasks' completion time, minimizing energy consumption of IoT devices/edge computing devices, and minimizing the operational cost of edge computing networks, etc. The priorities are included in our objective function as weighting factors. Another way to incorporate the priorities of tasks is through the

consideration of utility of the edge computing network [78] [79] [80] [81]. The utility here defines the revenue for the edge computing network to provide services to IoT applications. For example, the utility can be defined as the price paid by each task owner to have its request instantiated and executed by the edge computing network. In the view of the edge computing network, the priority of each task can be represented by the utility that the task provides to the edge computing network. Hence, for the objective function, it can be established with the consideration of utility, for example, maximizing the utility of the edge computing network. In addition, more comprehensive utility functions can be considered. For example, the utility of the edge computing network will not only consider the income from tasks but also the payment for using network resources. As the income is related to priority and the payment related to task workload, this comprehensive utility function will consider both priorities and workload of tasks.

Most IoT tasks are not related as there are naturally from different IoT devices and IoT applications, hence, in this dissertation research, we have one assumption that the tasks in our approach are not dependent with each other, which is practical. As IoT applications grow more and more complex, some IoT application may involve the collaboration of multiple IoT devices, which brings the dependencies between tasks from multiple IoT devices. For example, the virtual medical devices [30] connect multiple medical devices and run multiple clinic algorithms. Hence, for these kinds of IoT tasks, the preprocessing steps should be revised to represent the constraints of such a dependency. The data flow between tasks should also be considered when modeling the network bandwidth provisioning.

REFERENCES

[1] Aiello, William, Fan Chung, and Linyuan Lu. "A random graph model for massive graphs." Proceedings of the thirty-second annual ACM symposium on Theory of computing. Acm, 2000.

[2] Balan, Rajesh Krishna. "Powerful change part 2: reducing the power demands of mobile devices." IEEE Pervasive Computing 3.2 (2004): 71-73.

[3] Barbarossa, Sergio, Stefania Sardellitti, and Paolo Di Lorenzo. "Joint allocation of computation and communication resources in multiuser mobile cloud computing." Signal Processing Advances in Wireless Communications (SPAWC), 2013 IEEE 14th Workshop on. IEEE, 2013.

[4] Beck, Michael Till, Martin Werner, Sebastian Feld, and S. Schimper. "Mobile edge computing: A taxonomy." Proc. of the Sixth International Conference on Advances in Future Internet. Citeseer, 2014.

[5] Bonomi, Flavio, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. "Fog computing and its role in the internet of things." Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM, 2012.

[6] "Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper." N.p., 1 Feb. 2018. Web. 13 Sep. 2018. <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html>.

[7] Chen, Guangyu, B-T. Kang, Mahmut Kandemir, Narayanan Vijaykrishnan, Mary Jane Irwin, and Rajarathnam Chandramouli. "Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices." IEEE Transactions on Parallel and Distributed Systems 15.9 (2004): 795-809.

[8] Chen, Ruitao, Xianbin Wang and Shuran Sheng. "Flexible Virtual Energy Sharing by Distributed Task Reallocation in IoT Edge Networks", 2018 IEEE Conference on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, Congress on Cybermatics, 2018.

[9] Chen, Xu, Lei Jiao, Wenzhong Li, and Xiaoming Fu. "Efficient multi-user computation offloading for mobile-edge cloud computing." IEEE/ACM Transactions on Networking 5 (2016): 2795-2808.

[10] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms. MIT press, 2009.

[11] Chu, Hao-hua, Henry Song, Candy Wong, Shoji Kurakake, and Masaji Katagiri. "Roam, a seamless application framework." Journal of Systems and Software 69.3 (2004): 209-226.

[12] Dinh, Hoang T., Chonho Lee, Dusit Niyato, and Ping Wang. "A survey of mobile cloud computing: architecture, applications, and approaches." Wireless communications and mobile computing 13.18 (2013): 1587-1611.

[13] Doar, Matthew B. "A better model for generating test networks." Global Telecommunications Conference, 1996. GLOBECOM'96.'Communications: The Key to Global Prosperity. IEEE, 1996.

[14] Garcia Lopez, Pedro, Pedro, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. "Edge-centric computing: Vision and challenges." ACM SIGCOMM Computer Communication Review 45.5 (2015): 37-42.

[15] Garey, Michael R., and David S. Johnson. Computers and intractability. Vol. 29. New York: wh freeman, 2002.

[16] Garriss, Scott, Rámon Cáceres, Stefan Berger, Reiner Sailer, Leendert van Doorn, and Xiaolan Zhang. "Trustworthy and personalized computing on public kiosks." Proceedings of the 6th international conference on Mobile systems, applications, and services. ACM, 2008.

[17] Gedawy, Hend, Karim Habak, Khaled Harras, and Mouri Hamdi. “An Energy-Aware IoT Femtocloud System”, Edge Computing (EDGE), 2018 IEEE International Conference on. IEEE, 2018.

[18] Gu, Lin, et al. "Cost efficient resource management in fog computing supported medical cyber-physical system." IEEE Transactions on Emerging Topics in Computing 5.1 (2017): 108-119.

[19] Gu, Xiaohui, Deze Zeng, Song Guo, Ahmed Barnawi, and Yong Xiang. "Adaptive offloading inference for delivering applications in pervasive computing environments." null. IEEE, 2003.

[20] Gurun, Selim, and Chandra Krintz. "Addressing the energy crisis in mobile computing with developing power aware software." Memory 8.64MB (2003): 512MB.

[21] Gurun, Selim, Chandra Krintz, and Rich Wolski. "NWSLite: a light-weight prediction utility for mobile devices." Proceedings of the 2nd international conference on Mobile systems, applications, and services. ACM, 2004.

[22] Hong, Yu-Ju, Karthik Kumar, and Yung-Hsiang Lu. "Energy efficient content-based image retrieval for mobile systems." Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on. IEEE, 2009.

[23] Hu, Wenlu, Ying Gao, Kiryong Ha, Junjue Wang, Brandon Amos, Zhuo Chen, Padmanabhan Pillai, and Mahadev Satyanarayanan. "Quantifying the impact of edge computing on mobile applications." Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems. ACM, 2016.

[24] Huerta-Canepa, Gonzalo, and Dongman Lee. "An adaptable application offloading scheme based on application behavior." Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on. IEEE, 2008.

[25] Jalali, Fatemeh, Kerry Hinton, Robert Ayre, Tansu Alpcan, and Rodney S. Tucker. "Fog computing may help to save energy in cloud computing." IEEE Journal on Selected Areas in Communications 34.5 (2016): 1728-1739.

[26] Jin, Cheng, Qian Chen, and Sugih Jamin. "Inet: Internet topology generator." (2000).

[27] Kumar, Karthik, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. "A survey of computation offloading for mobile systems." Mobile Networks and Applications 18.1 (2013): 129-140.

[28] Kumar, Karthik, and Yung-Hsiang Lu. "Cloud computing for mobile users: Can offloading computation save energy?." Computer 43.4 (2010): 51-56.

[29] Kurose, James F. Computer networking: A top-down approach featuring the internet, 3/E. Pearson Education India, 2005.

[30] Lee, Insup, Oleg Sokolsky, Sanjian Chen, John Hatcliff, Eunkyoung Jee, BaekGyu Kim, Andrew King, Margaret Mullen-Fortino, Soojin Park, Alexander Roederer, and Krishna K.Venkatasubramanian. "Challenges and research directions in medical cyber–physical systems." Proceedings of the IEEE 100.1 (2012): 75-90.

[31] Li, Yuanzhe, Shangguang Wang. “An energy-aware Edge Server Placement Algorithm in Mobile Edge Computing”, Edge Computing (EDGE), 2018 IEEE International Conference on. IEEE, 2018.

[32] Li, Zhiyuan, Cheng Wang, and Rong Xu. "Computation offloading to save energy on handheld devices: a partition scheme." Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems. ACM, 2001.

[33] Li, Zhiyuan, Cheng Wang, and Rong Xu. "Task allocation for distributed multimedia processing on wirelessly networked handheld devices." Parallel and Distributed

Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM. IEEE, 2001.

[34] Li, Zhiyuan, and Rong Xu. "Energy impact of secure computation on a handheld device." Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on. IEEE, 2002.

[35] Liang, Hongbin, Tianyi Xing, Lin X. Cai, Dijiang Huang, Daiyuan Peng, and Yan Liu. "Adaptive computing resource allocation for mobile cloud computing." International Journal of Distributed Sensor Networks 9.4 (2013): 181426.

[36] Liu, Juan, Yuyi Mao, Jun Zhang, and Khaled B. Letaief. "Delay-optimal computation task scheduling for mobile-edge computing systems." Information Theory (ISIT), 2016 IEEE International Symposium on. IEEE, 2016.

[37] Magoni, Damien, and Jean-Jacques Pansiot. "Analysis and comparison of Internet topology generators." International Conference on Research in Networking. Springer, Berlin, Heidelberg, 2002.

[38] Mao, Yuyi, Jun Zhang, and Khaled B. Letaief. "Dynamic computation offloading for mobile-edge computing with energy harvesting devices." IEEE Journal on Selected Areas in Communications 34.12 (2016): 3590-3605.

[39] Medina Alberto. “Available Topology Generators.” N.p., December 4, 2001. Web. September 30, 2018. <https://www.cs.bu.edu/brite/user_manual/node3.html>

[40] Medina, Alberto, Ibrahim Matta, and John Byers. "On the origin of power laws in Internet topologies." ACM SIGCOMM computer communication review 30.2 (2000): 18-28.

[41] Munoz, Olga, Antonio Pascual-Iserte, and Josep Vidal. "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading." IEEE Transactions on Vehicular Technology 64.10 (2015): 4738-4755.

[42] Nimmagadda, Yamini, Karthik Kumar, Yung-Hsiang Lu, and CS George Lee. "Real-time moving object recognition and tracking using computation offloading." Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on. IEEE, 2010.

[43] Ou, Shumao, Kun Yang, Antonio Liotta, and Liang Hu. "Performance analysis of offloading systems in mobile wireless environments." Communications, 2007. ICC'07. IEEE International Conference on. IEEE, 2007.

[44] Oueis, Jessica, Emilio Calvanese Strinati, and Sergio Barbarossa. "The fog balancing: Load distribution for small cell cloud computing." Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st. IEEE, 2015.

[45] O'Hara, Keith J., Ripal Nathuji, Himanshu Raj, Karsten Schwan, and Tucker Balch. "Autopower: Toward energy-aware software systems for distributed mobile robots." Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on. IEEE, 2006.

[46] Rahimi, M. Reza, Jian Ren, Chi Harold Liu, Athanasios V. Vasilakos, and Nalini Venkatasubramanian. "Mobile cloud computing: A survey, state of art and future directions." Mobile Networks and Applications 19.2 (2014): 133-143.

[47] Roman, Rodrigo, Javier Lopez, and Masahiro Mambo. "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges." Future Generation Computer Systems 78 (2018): 680-698.

[48] Rong, Peng, and Massoud Pedram. "Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach." Proceedings of the 40th annual Design Automation Conference. ACM, 2003.

[49] Sardellitti, Stefania, Gesualdo Scutari, and Sergio Barbarossa. "Joint optimization of radio and computational resources for multicell mobile-edge computing." IEEE Transactions on Signal and Information Processing over Networks 1.2 (2015): 89-103.

[50] Sardellitti, Stefania, Sergio Barbarossa, and Gesualdo Scutari. "Distributed mobile cloud computing: Joint optimization of radio and computational resources." Globecom Workshops (GC Wkshps), 2014. IEEE, 2014.

[51] Satyanarayanan, Mahadev, et al. "The case for vm-based cloudlets in mobile computing." IEEE pervasive Computing (2009).

[52] Shi, Weisong, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. "Edge computing: Vision and challenges." IEEE Internet of Things Journal 3.5 (2016): 637-646.

[53] Sun, Xiang, and Nirwan Ansari. "EdgeIoT: Mobile edge computing for the Internet of Things." IEEE Communications Magazine 54.12 (2016): 22-29.

[54] Surie, Ajay, Adrian Perrig, Mahadev Satyanarayanan, and David J. Farber. "Rapid trust establishment for pervasive personal computing." IEEE Pervasive Computing 6.4 (2007).

[55] Song, Yaozhong, Stephen S. Yau, Ruozhou Yu, Xiang Zhang, and Guoliang Xue. "An Approach to QoS-based Task Distribution in Edge Computing Networks for IoT Applications." Edge Computing (EDGE), 2017 IEEE International Conference on. IEEE, 2017. (Best Student Paper Award)

[56] Taleb, Tarik, Sunny Dutta, Adlen Ksentini, Muddesar Iqbal, and Hannu Flinck. "Mobile edge computing potential in making cities smarter." IEEE Communications Magazine 55.3 (2017): 38-43.

[57] Thyagaturu, Akhilesh S., Yousef Dashti, and Martin Reisslein. "SDN-based smart gateways (Sm-GWs) for multi-operator small cell network management." IEEE Transactions on Network and Service Management 13.4 (2016): 740-753.

[58] Tilevich, Eli, and Yannis Smaragdakis. "J-orchestra: Automatic java application partitioning." European conference on object-oriented programming. Springer, Berlin, Heidelberg, 2002.

[59] Tran, Tuyen X., Abolfazl Hajisami, Parul Pandey, and Dario Pompili. "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges." IEEE Communications Magazine 55.4 (2017): 54-61.

[60] Vaquero, Luis M., and Luis Rodero-Merino. "Finding your way in the fog: Towards a comprehensive definition of fog computing." ACM SIGCOMM Computer Communication Review 44.5 (2014): 27-32.

[61] Varghese, Blesson, et al. "Challenges and opportunities in edge computing." arXiv preprint arXiv:1609.01967 (2016).

[62] Wang, Cheng, and Zhiyuan Li. "Parametric analysis for adaptive computation offloading." ACM SIGPLAN Notices. Vol. 39. No. 6. ACM, 2004.

[63] Wang, Yuan, and Partha Dasgupta. "Designing an adaptive lighting control system for smart buildings and homes." Networking, Sensing and Control (ICNSC), 2015 IEEE 12th International Conference on. IEEE, 2015.

[64] Waxman, Bernard M. "Routing of multipoint connections." IEEE journal on selected areas in communications 6.9 (1988): 1617-1622.

[65] Wen, Yonggang, Weiwen Zhang, and Haiyun Luo. "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones." INFOCOM, 2012 Proceedings IEEE. IEEE, 2012.

[66] Wolski, Rich, Selim Gurun, Chandra Krintz, and Dan Nurmi. "Using bandwidth data to make computation offloading decisions." Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on. IEEE, 2008.

[67] Xian, Changjiu, Yung-Hsiang Lu, and Zhiyuan Li. "Adaptive computation offloading for energy conservation on battery-powered systems." Parallel and Distributed Systems, 2007 International Conference on. Vol. 2. IEEE, 2007.

[68] Xing, Tianyi, Dijiang Huang, Shingo Ata, and Deep Medhi. "MobiCloud: a geo-distributed mobile cloud computing platform." Proceedings of the 8th International Conference on Network and Service Management. International Federation for Information Processing, 2012.

[69] Yau, Stephen S., and Dazhi Huang. "Distributed monitoring and adaptation of multiple qos in service-based systems." 2011 35th IEEE Annual Computer Software and Applications Conference Workshops. IEEE, 2011.

[70] Yau, Stephen S., and Ho G. An. "Adaptive resource allocation for service-based systems." Proceedings of the First Asia-Pacific Symposium on Internetware. ACM, 2009.

[71] Yau, Stephen S., and Ho G. An. "Protection of users' data confidentiality in cloud computing." Proceedings of the Second Asia-Pacific Symposium on Internetware. ACM, 2010.

[72] Yau, Stephen S., Yin Yin, and Ho An. "An adaptive approach to optimizing tradeoff between service performance and security in service-based systems." International Journal of Web Services Research (IJWSR) 8.2 (2011): 74-91.

[73] Yi, Shanhe, Cheng Li, and Qun Li. "A survey of fog computing: concepts, applications and issues." Proceedings of the 2015 workshop on mobile big data. ACM, 2015.

[74] Ye, Yinyu. "An O (n3L) potential reduction algorithm for linear programming." Mathematical programming 50.1-3 (1991): 239-258.

[75] Yu, Ruozhou. "Application Provisioning in Fog Computing-enabled Internet-of-Things: A Network Perspective.".

[76] Zegura, Ellen W., Kenneth L. Calvert, and Michael J. Donahoo. "A quantitative comparison of graph-based models for Internet topology." IEEE/ACM Transactions on Networking (TON) 5.6 (1997): 770-783.

[77] Zeng, Deze, Lin Gu, Song Guo, Zixue Cheng, and Shui Yu. "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system." IEEE Transactions on Computers 65.12 (2016): 3702-3712.

[78] Zhang, Xiang, Guoliang Xue, Ruozhou Yu, Dejun Yang, and Jian Tang. "Truthful incentive mechanisms for crowdsourcing." Computer Communications (INFOCOM), 2015 IEEE Conference on. IEEE, 2015.

[79] Zhang, Xiang, Guoliang Xue, Ruozhou Yu, Dejun and Yang. "You better be honest: Discouraging free-riding and false-reporting in mobile crowdsourcing." Global Communications Conference (GLOBECOM), 2014 IEEE. IEEE, 2014.

[80] Zhang, Xiang, Guoliang Xue, Ruozhou Yu, Dejun and Yang. "Keep your promise: Mechanism design against free-riding and false-reporting in crowdsourcing." IEEE Internet of Things Journal 2.6 (2015): 562-572.

[81] Zhang, Xiang, Guoliang Xue, Ruozhou Yu, and Dejun Yang. "Robust incentive tree design for mobile crowdsensing." Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on. IEEE, 2017.

[82] Zhao, Xingguang, Xing Guo, Yiwen Zhang, and Wei Li. “A Parallel-batch Multi-objective Job Scheduling Algorithm in Edge Computing”, 2018 IEEE Conference on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, Congress on Cybermatics, 2018.