Scheduling Problems in Next Generation Computing Networks

By Jia He Sun

A thesis submitted to the Graduate Program in School of Computing in conformity with the requirements for the Degree of Master of Science

Queen's University
Kingston, Ontario, Canada
November, 2022

Abstract

With the development of the novel Internet of Things (IoT) ecosystem, the digital support required by emerging applications is a highly growing area of concern. The current network paradigm, cloud computing, is proving unable to adequately support the ever increasing computational and storage needs of the billions of edge devices in use. In recent years, several solutions have been put forward including two complementary computing paradigms: Multi-access Edge Computing (MEC) and fog computing. MEC and fog computing are both considered augments to cloud computing since they both conceptualize supplementary server layers to existing networks.

This thesis addresses the emerging challenge of efficient task scheduling in both MEC and fog computing. Since traditional mathematical programming models are proven to be very difficult to solve optimally at large dimensions, another approach is needed to satisfy the time-sensitive needs of the dynamic environments of both MEC and fog computing. In this thesis, we propose several heuristics that are designed to provide good solutions while maintaining low complexity guarantees.

The first problem examined in this thesis is how to fully utilize parked vehicles (PVs) as computational resources in vehicular networks under the MEC framework. We formulate a multi-objective task offloading problem that minimizes both task delay and wireless channel load. Then, a stable matching based heuristic is proposed

and evaluated at various configurations of the vehicular environment.

The second problem that we examine in thesis is how to fairly allocate resources in fog computing. Fairness in resource allocation is a highly desired quality because it not only increases the quality of service (QoS) for users but also maximizes the resource utilization in the system. This thesis adopts the Dominant Resource Fairness (DRF) scheme and applies it to a multi-resource, multi-server, and heterogeneous task environment. Furthermore, four different types of tasks are considered: ordered/unordered, splittable/unsplittable. Finally, three different low complexity heuristics are proposed to maximize fairness between users under the DRF scheme.

Acknowledgments

Firstly, I would like to thank Queen's University's School of Computing for their financial support.

I would also like to thank my supervisors, Dr. Salimur Choudhury and Dr. Kai Salomaa, for their continued support and advice throughout the past two years.

In addition, I would like to thank Dr. Sameh Sorour for his constructive comments on Chapter 3.

Furthermore, to my examining committee (Dr. Thom Dean and Dr. Robin Dawes), thank you for your questions and helpful comments. Also, thanks to Dr. Burton Ma for chairing the defense.

Lastly, I would like to give my thanks to everyone who has given me love and support over the past two years. Notably, Lucia for her help with editing; Zoe for her help with diagrams; Vincent for literally being my support; and, of course, my family for always being there for me.

Statement Of Originality

The following work is my own and I hereby certify the intellectual content of this thesis is the product of my own work. All references and contributions of other individuals has been cited and sourced appropriately.

Glossary of Abbreviations

5G Fifth Generation. 3, 20

BS Base Station. 20, 22, 24

C-V2X Cellular Vehiclar to Everything. 20

CEEI Competitive Equilibrium from Equal Incomes. 17, 44, 45

CV Coefficient of Variation. 38

DDRF Distributed Dominant Resource Fairness. 46

DRF Dominant Resource Fairness. 6, 17, 42, 44–50, 52, 53, 60, 71, 77, 79

FCFS First Come First Server. 67–70, 73, 74

ILP Integer Linear Problems. 7–9, 23–26, 29, 36

IoT Internet of Things. 2, 10, 17

ITS Intelligent Transportation System. 10

LTE Long-Term Evolution. 19

MEC Multi-access Edge Computing. 2–4, 7, 9–12, 15, 78

PRR Packet Reception Ratio. 20

PV Parked Vehicle. 5, 13, 19–22

PVC Parked Vehicular Computing. 13, 19, 21, 40, 78, 79

QoS Quality of Service. 17, 20, 45

RS Roth-Shapley. 27–29, 31, 36, 39, 40

SJF Shortest Job First. 67–70, 73, 74, 76

VCC Vehicular Cloud Computing. 19

VEC Vehicular Edge Computing. 5, 7, 10, 12, 13, 19, 20, 24, 25

Contents

List of Tables

List of Figures

List of Equations

Chapter 1

Introduction

Cloud computing is at the forefront of computing paradigms by centralizing main computing and storage capabilities in a remote location that is accessible to all [1]. However, with the vision of the Internet of Things (IoT), we have realized that the remote network system is simply not enough to satisfy the intensive computational needs of the future that we envision [4] [5]. In recent years, distributed computing infrastructures have evolved at an rapid pace. With it, the sheer quantity of computational devices that comprise the edge layer of these large scale networks is astounding [6]. As a result, the vast amount of data produced by the edge devices as well as the computational needs required by the edge devices have become severe bottlenecks in large scale networks that rely on cloud computing. Several novel computing paradigms have been pushed forward as solutions to this problem including Multi-access Edge Computing (MEC) and fog computing. MEC and fog computing are both emerging complementary computing paradigms that aim to address current limitations in cloud computing [7].

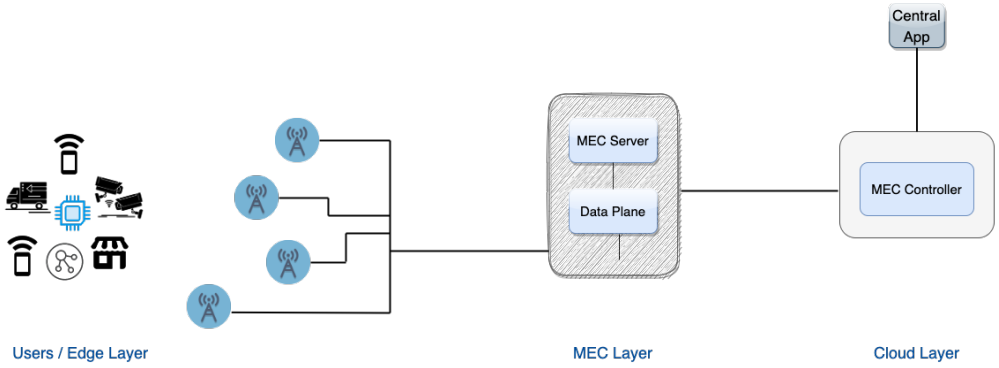

Figure 1.1: MEC Infrastructure

1.1 Multi-access Edge Computing

MEC, as an augment computing paradigm to cloud computing, extends cloud computing services to the edge of networks through the use of base stations. As new technological services emerge, the computational power required by edge devices are becoming a severe bottleneck in computing networks. To counteract this, MEC aims to provide highly responsive and low latency support to edge users by bringing computation and storage resources closer to them. MEC plays an important role in the developing Fifth Generation (5G) networks which support a variety of applications and services where ultra-low latency communication is required. Fig. 1.1 visualizes the infrastructure of a MEC network where there is an additional MEC layer between the users and the central cloud. This is especially important in vehicles as the vehicular environment requires incredibly low latency due to its dynamic nature [8]. Furthermore, since it is financially infeasible to mass install expensive computing hardware into every vehicle, edge computing has become the foremost solution in vehicular networks [9].

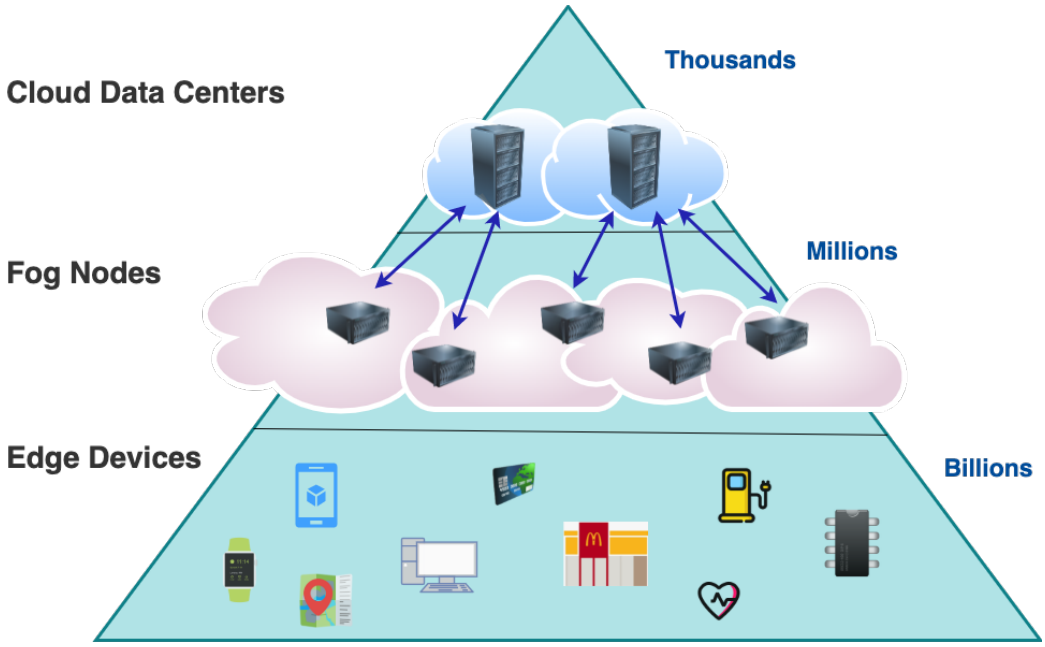

Figure 1.2: Fog Computing Infrastructure

1.2 Fog Computing

Fog computing is another complementary computing paradigm that is aimed to handle the ever increasing computational needs of our networks. As displayed in Fig. 1.2, fog computing is different from MEC as it introduces an intermediary layer between the cloud layer and the edge layer named the fog layer. This fog layer is comprised of many fog nodes that provide data processing and analysis for the edge devices with reduced latency and increased quality as well as data filtering and load management for the cloud layer. Overall, the implementation of the fog layer aims to provide better service to the end devices while reducing the burden on the cloud layer by managing data analysis and computational requests at an intermediate level.

To accomplish this, part of the fog layer's job is to receive and process computational service requests put forward by edge devices. In this infrastructure, computational service requests can be observed by several fog nodes. Each request is comprised of a set of tasks and each fog node is comprised of a set of resources. Consequently, the task scheduling problem is one of the main focuses of current research in the fog computing field and is the focus of Chapter 4 [10].

1.3 Thesis Outline

The remainder of this thesis is organized as follows.

Chapter 2 "Background" provides a detailed account of fundamental concepts used in this thesis.

Chapter 3 "Multi-Objective Task Assignment Solution for Parked Vehicular Computing" examines a problem in the current development of Vehicular Edge Computing (VEC) which is the high cost of installing enough edge servers to compute all offloaded tasks at peak hours. However, it has been observed that Parked Vehicle (PV)s are a rich reserve of underutilized computing resources and their incorporation into the VEC network could lead to a solution to the aforementioned problem. Chapter 3 proposes a task offloading system with an assumed parking time estimation mechanism. Then, a novel formulation of the task offloading problem is presented. Finally, a matching based heuristic is proposed and evaluated at various configurations of the VEC environment.

Chapter 4 "An Online Fair Resource Allocation Solution for Fog Computing" examines a fundamental problem of fog computing which is how to allocate the computing resources of fog nodes when scheduling tasks that arrive in an online

manner. Other than task completion speed metrics, fairness of resource allocation between competing users is also an important metric to consider. One such metric is Dominant Resource Fairness (DRF), a fairness scheme that guarantees four key qualities: incentivized sharing, strategy-proof, Pareto-efficiency, and envy free. This paper examines the multi-resource, multi-server, and heterogenous task resource allocation problem from a DRF perspective. Four different types of tasks are considered: ordered/unordered, splittable/unsplittable. Three low complexity heuristics are proposed to maximize fairness between users. Results show that the proposed heuristics are at least comparable to three baseline scheduling algorithms in terms of task completion speed while achieving higher fairness between users.

Chapter 5 "Conclusion" concludes the thesis and provides current limitations as well as future research directions.

Chapter 2

Background

This chapter provides the necessary background for this thesis, including overviews on the following topics: mathematical optimization, Multi-access Edge Computing (MEC), Vehicular Edge Computing (VEC), and fog computing.

2.1 Mathematical Optimization

Mathematical optimization is a branch of applied mathematics that is present in many different areas such as inventory control, transportation, scheduling, networks, finance, economics, etc. It is typically used to make decisions by analyzing models of physical situations [11]. In general terms, an optimization problem is finding the best solution under some set of constraints. The problems studied in this thesis are both considered optimization problems, or more specifically, Integer Linear Problems (ILP). The general canonical form of an ILP is comprised of three components: objective function(s), variable(s), and constraint(s) [11]. An ILP is typically represented in the following form:

$$\underset{x}{\text{minimize}} \quad \sum c^T x \tag{2.0.1a}$$

subject to

$$Ax \geq b, \tag{2.0.1b}$$

$$x \in \{0,1\}^n \tag{2.0.1c}$$

The objective function that we are trying to minimize is 2.0.1a, subject to constraints 2.0.1b and 2.0.1c. Constraint 2.0.1b is a general form of a constraint that depends on the problem. Constraint 2.0.1c restricts the solution variable x to be only integers and is in every ILP. Any solution that satisfies all the constraints is a feasible solution. The set of all such solutions is called the feasible set. Finding the optimal solution is finding a solution x from the feasible set that produces the minimum objective value based on 2.0.1a.

2.1.1 Scheduling

There are many different types of problems under the umbrella term optimization problems. In this thesis, we are concerned with a specific type of ILP called *scheduling problems*. Scheduling problems solve for the optimal schedule under various machine environments and jobs characteristics [12]. Let the set of jobs be $J = \{J_1, ..., J_n\}$ and the set of machines be $M = \{M_1, ..., M_m\}$. A generic scheduling problem would ask for a mapping from jobs J to machines M, subject to feasibly constraints and optimization objectives [13]. There can be very many different constraints in a scheduling

problem. Some common ones are: jobs must be completed in a certain order (precedence), jobs must be completed within a specific time frame (due dates), and jobs are only available to be executed after a certain amount of time (release dates). There are also very many different metrics that can be used as objective functions in scheduling problems. Some common ones are: maximum completion time (makespan), number of jobs that are completed after their due date (tardiness), and waiting time of jobs (downtime).

2.1.2 Optimization Algorithms

The ILP model is widely studied in both academia and industry. As such, there exist several commercial solvers that can be used to solve ILPs optimally, such as Gurobi, CPLEX, and OSL [14] [15] [16]. However, even using these software, ILPs are incredibly difficult to solve at large dimensions [11].

In terms of scheduling algorithms, they can be mainly classified into the follow two categories: static scheduling and dynamic scheduling [17]. Both have their own advantages and limitations. For example, dynamic scheduling algorithms typically have higher performance than static algorithms but also have a lot more overhead. Fig. 2.1 displays a more detailed taxonomy of typical task scheduling algorithms.

2.2 Mult-access Edge Computing

With significant advances in recent technology, computational power must meet new demands. As a result, MEC, or previously known as Mobile Edge Computing, is an emerging networking paradigm that has received a surge in interest from both academia and the industry. MEC aims to push powerful computing and storage

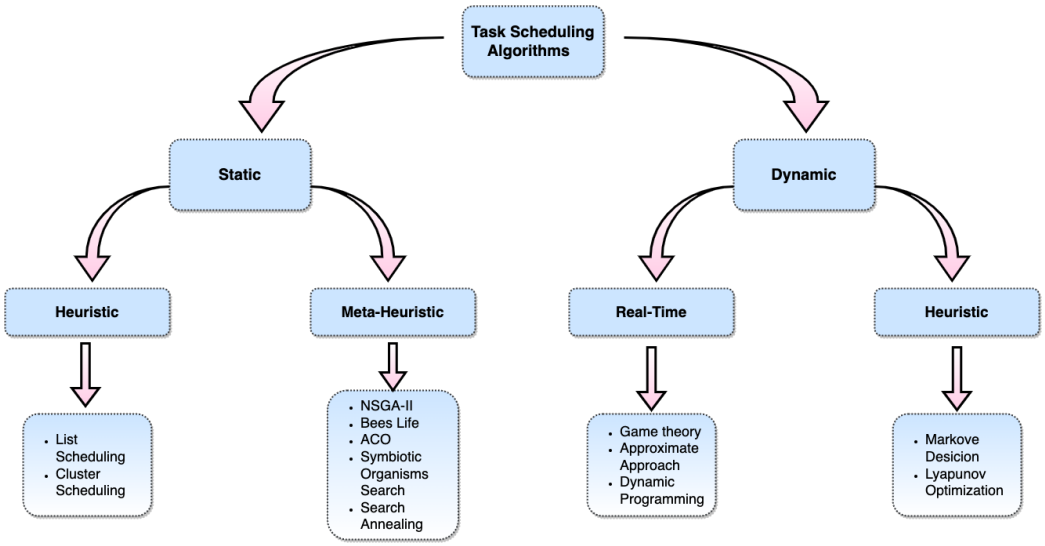

Figure 2.1: Taxonomy of Scheduling Algorithms

capabilities from remote cloud servers to up close edge servers [18]. It is applicable in many different promising technologies such as Internet of Things (IoT), virtual reality, and smart vehicular networks [19]. Recall that Fig. 1.1 shows the typical network infrastructure in MEC.

2.3 Vehicular Edge Computing

VEC, a subfield of MEC, has been introduced to specifically increase the computing capacity of vehicular networks, an essential component for the development of Intelligent Transportation System (ITS). VEC aims to address the lack of computational resources on board vehicles, as it is more financially feasible than the mass installation of hardware in every vehicle and more stable than using cloud computing resources. By using edge servers, vehicles can reliably offload their computational tasks, alleviating the heavy burden placed on the vehicles' internal hardware. Fig. 2.2 illustrates the structure of the overall VEC system. There are three layers included

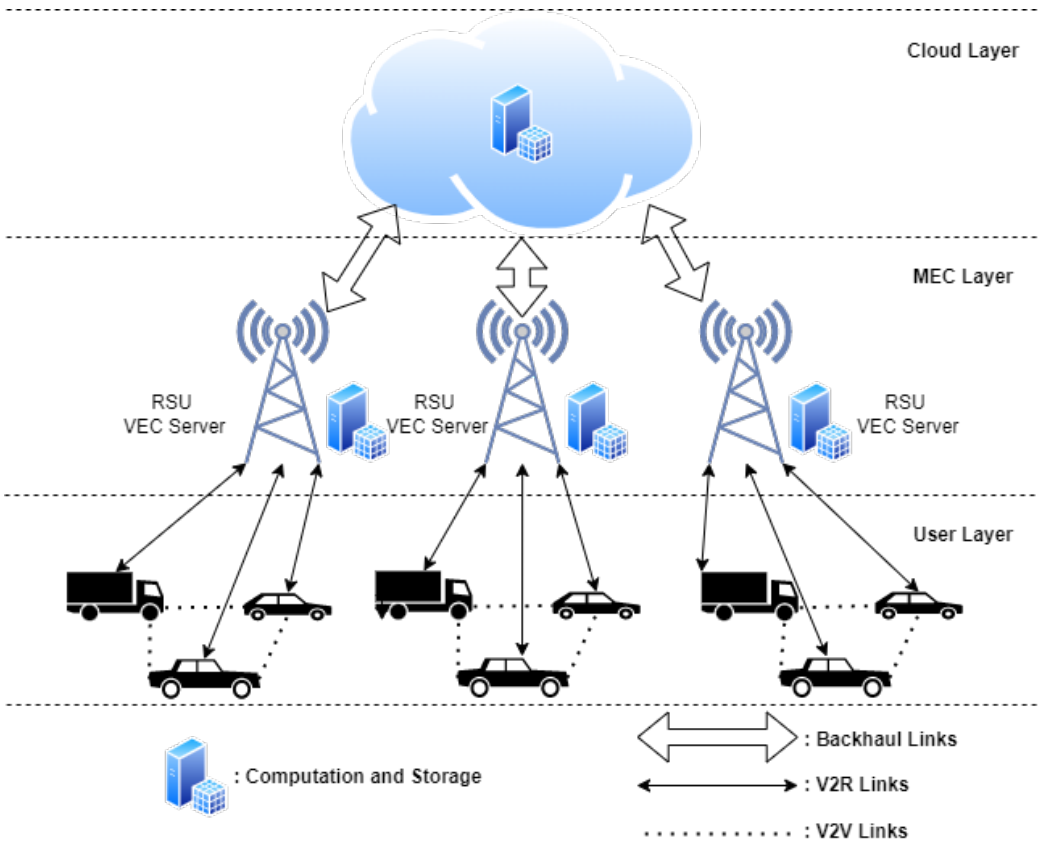

Figure 2.2: Vehicular Edge Computing (VEC) Architecture

in this model:

- The cloud layer (cloud computing servers) provides a global managerial view of the entire network system by providing centralized control.
- The MEC layer (roadside units) comprises the edge servers that are the main computational resources. They are responsible for receiving information and tasks from the vehicles and, following a specified algorithm, they will process the information and complete the tasks before sending the information back to the vehicles.

- The user layer (vehicular devices) is mainly composed of vehicles. While they are able to communicate with the MEC layer to offload their tasks for processing, they are also able to communicate with other vehicles and can utilize other vehicles' computing/storage power for their own tasks.

Table 2.1: Comparison table: Vehicular Edge Computing vs Vehicular Cloud Computing [1]

Features	**VEC**	**VCC**
Location	At user's proximity	Remote location
Latency	Low	High
Mobility Support	High	Limited
Decision making	Local	Remote
Communication	Real time	Constraints in bandwidth
Storage capacity	Limited	Highly scalable
Context awareness	Yes	No
Device heterogeneity	Highly supported	Limited support
Computing capability	Medium	High
Cost of development	Low	High

As indicated in Table 2.1, VEC adopts all of the benefits of MEC and integrates them within the existing vehicular network infrastructure. By moving computational resources closer to the vehicles, it addresses the vehicular network's exponentially growing need for low latency and high bandwidth. Furthermore, VEC exceeds the traditional MEC network with its dynamic nature. The mobility of vehicles reflects the need for a network with an infrastructure that could support frequent movement and rapidly changing channel environments, needs that are readily addressed by VEC [1].

A current problem in VEC is the high cost of installing enough edge servers to

compute all offloaded tasks at peak hours. However, we have observed that Parked Vehicle (PV)s are a rich reserve of underutilized computing resources [20]. Thus, their incorporation into the VEC network could lead to a solution to the aforementioned problem. Research in this area is called Parked Vehicular Computing (PVC) and is the focus of Chapter 3.

Currently, there are five main technical issues of VEC that are currently being researched: latency, scheduling, resource management, privacy and security, offloading. The problem we examine in this thesis is under the offloading category.

2.3.1 Offloading

As mentioned previously, the edge servers will not have a large enough computational capacity to handle every single vehicle's request. Furthermore, it is not financially realistic to implement enough roadside edge servers to be able to have the same computational capacity as a cloud server. Thus, a computational offloading model which allows tasks to be offloaded to another edge server or vehicle was proposed. The main topics in offloading can be categorized as follows: mobility awareness, energy efficiency, incentive strategy, server-based offloading, and cooperative offloading [9].

2.3.2 Stable Matching Problem

The stable matching problem, also known as the stable marriage problem, is a model often used to solve task offloading problems. An instance of the stable matching problem consists of two disjoint sets of equal size, the men and the women. Each person has their own strictly ordered list of preferences that includes every member of the opposite sex. For example, man n prefers woman m to woman m' if

woman m precedes woman m' on man n's preference list. A matching is a bijection from the elements of one set to the elements of the other. A matching is called *stable* if there are no two people of the opposite sex who would both rather have each other than their currently matched partners (a more formal definition of stability is given in Chapter 4). The goal of this problem is to find a stable matching between the men and the women.

A variant of the stable matching problem is the hospital-resident problem. In the stable matching problem, each man can only be matched with one woman and vice versa, hence, the solution is a bijection between the men and the women. However, in the hospital-resident problem, each hospital can be matched with many residents. Then, a solution to the hospital-resident problem is a many to one matching, where each resident is matched to one hospital but each hospital can be matched to multiple residents. This is also sometimes called the college acceptance problem.

In 1962, David Gale and Lloyd Shapley presented an $O(n^2)$ algorithm that solves both the stable matching problem and the hospital-resident problem [21]. It is aptly named the Gale-Shapley algorithm. They further proved that, in the Gale-Shapley algorithm, for any equal number of men and women, it is always possible to find a solution to the stable matching problem that is stable. Later, Alvin Roth and Lloyd Shapley would apply this theory in the field of economics and win a Nobel prize for their work on stable allocation theory and market design [22]. Today, the algorithm used to solve the stable matching problem (and its variants) has several names including: "the Extended Gale-Shapley algorithm", "the Capacitated Gale-Shapley algorithm", "the Roth-Shapley algorithm", and "the deferred acceptance algorithm".

2.4 Fog Computing

In recent years, distributed computing infrastructures have evolved at a rapid pace. With it, the sheer quantity of computational devices that comprise the edge layer of these large scale networks is astounding [23]. As a result, the vast amount of data produced by the edge devices as well as the computational needs required by the edge devices have become severe bottlenecks in large scale networks that rely on cloud computing. Fog computing is one of several emerging complementary computing paradigms that aim to address current limitations in cloud computing [7]. Recall Fig. 1.2 which displays the infrastructure of fog computing. The concept of fog computing may sound similar to the previously discussed MEC, but there are key differences. Within the fog layer, there exists a complex hierarchy of many intermediate sublayers between the edge devices and the central cloud [24]. Edge computing consists of many sublayers located near the edge of the network [25]. Finally, MEC is a single layer of nodes located at the edge of the network that provides ultra low latency service to edge users [26]. In this sense, fog computing is considered a superset of edge computing which is a superset of MEC. Fig. 2.3 displays this concept.

2.4.1 Resource Allocation

A fundamental problem in fog computing is how to allocate the vast amount of resources provided by the many fog nodes. Whenever a user submits tasks to the fog layer, it must be given a certain amount of resources to be completed. How many resources and on which fog node are key questions to consider.

Refer back to Fig. 2.1, resource allocation and task scheduling algorithms can be

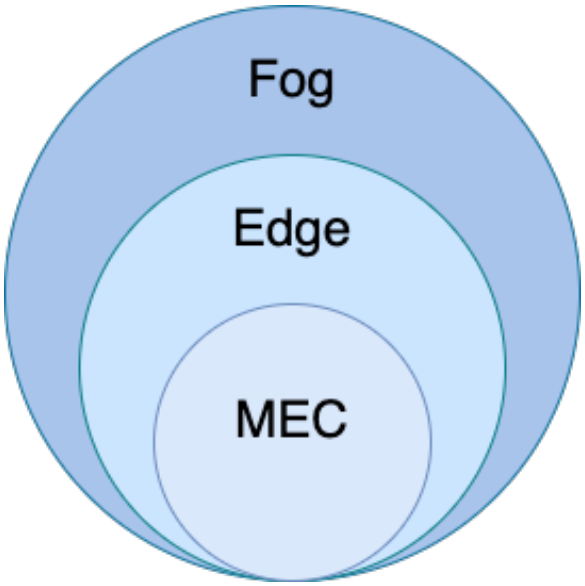

Figure 2.3: Fog Computing vs Edge Computing vs MEC

categorized as either static or dynamic. In static scheduling, information regarding every incoming task are made available to the system. On the other hand, in dynamic scheduling, the resource requirements of each task are not known until its arrival. Due to the dynamic nature of the numerous edge devices in a fog environment, this thesis's focus is dynamic scheduling algorithms, more specifically, real-time scheduling algorithms, where the tasks arrive to the system in a continuous manner. In this thesis, we will call this online scheduling.

However, traditional resource allocation techniques cannot be applied to the emerging fog computing environment. There are many new challenges regarding resource allocation due to the heterogeneity in hardware capabilities and heterogeneity in requested tasks [27].

There are many different metrics used to evaluate resource allocation techniques. Primarily, these metrics focus on the completion of tasks such as resource utilization, machine/task downtime, etc [24]. However, a lesser studied area is how to allocate

resources in a way that is *fair*.

2.4.2 Fairness

From a user's perspective, their Quality of Service (QoS) is heavily associated with their perceived fairness of the resource allocation. Unfair resource allocation can lower the willingness of IoT users to utilize their offloading system. Fair resource allocation can lead to very desirable attributes as well, such as no starvation of tasks, lack of jealousy between users, and better usage of resources [28]. However, fairness is hardly a well defined term in this regard. Is allocating an equal amount of each resource a fair allocation? Hardly so, considering each user will have heterogeneous tasks that require different amounts of different resources. The key to fair resource allocation is developing an appropriate metric to evaluate fairness such that maximizing said metric can guarantee positive attributes.

One of the most commonly used fair allocation schemes is called max-min. It is very generalizable due to its simplicity and has attracted a lot of use in recent years [27]. Essentially, it aims to maximize the minimum allocated share across all users. However, it is only applicable in single resource environments which is hardly the case in fog computing. Of course, there are also various multi-resource fairness schemes such as Dominant Resource Fairness (DRF), Asset Fairness, and Competitive Equilibrium from Equal Incomes (CEEI) [2]. Each of these schemes have their own advantages and limitations and will be further discussed in Chapter 4.

Chapter 3

A Multi-Objective Task Assignment Solution for Parked Vehicular Computing

The main contributions of this chapter can be summarized as follows:

- To reflect the multifaceted problem of task assignment within a VEC environment, we propose a novel formulation that includes a weighted multi-objective that aims to minimize both task delay and wireless channel load. We then prove the NP-completeness of the problem.
- We propose a many to one stable matching based heuristic to efficiently assign tasks to vehicles.
- We evaluate and confirm the performance of the proposed heuristic through various simulations.

3.1 Related Works

Vehicular Edge Computing (VEC) aims to use roadside edge servers to augment the computing capacity of vehicular environments. Under this framework, smart vehicles can reliably offload their computational tasks, alleviating the heavy burden placed on the vehicles' internal hardware. Recall that Fig. 2.2 illustrates the structure of the overall VEC system. The key advantages of VEC over Vehicular Cloud Computing (VCC) are: low latency, mobility support, real-time communication, heterogeneous device support, and lower cost of development. Although it is not without drawbacks: limited capacity and lower computing capability [1]. In terms of PVC, there exist some challenges that should be addressed to facilitate Parked Vehicular Computing (PVC). Firstly, scheduling computational tasks on participating Parked Vehicle (PV)s poses an interesting challenge [29]. PVs have an inherent problem in that they may leave unexpectedly, perhaps in the middle of a task, which results in the system having to offload the interrupted task elsewhere, delaying it further. Secondly, the VEC network mainly communicates over wireless channels and the overuse of such channels would cause an overload, thereby decreasing the quality of existing communications [30].

Huang *et al.* [31] designed an interactive protocol with request and response operations for service provision in PVC. Then, to solve the resource scheduling problem, they employ the Stackelberg game approach. Ge *et al.* [32] proposed an efficient Schoof–Elkies–Atkin algorithm to solve the vehicle selection and task assignment problem regarding service migration, namely, the transfer of tasks between base stations.

Wang *et al.* [33] implemented a system-level simulator of Long-Term Evolution

(LTE) Sidelink Cellular Vehiclar to Everything (C-V2X) Communication for Fifth Generation (5G). This simulation showed that the volume of data severely increased as the number of users in the network increased. The Packet Reception Ratio (PRR) also drastically decreased as the network size grew, reaching as low as 80.36% in a 1,920 user setting. As VEC is expected to be implemented on scales much larger than that, it is essential to maintain the quality of communication in these settings.

However, there are currently very few works that address the potential issues of large-scale implementation, specifically in terms of wireless communications. The current paradigm for wireless communication in vehicular networks is C-V2X, which has strict requirements on Quality of Service (QoS) such as high reliability and low latency [34]. Therefore, it is necessary for VEC networks to not only provide reliable high speed service, but also to maintain the quality of the communication channels used.

In this chapter, we propose a many-to-one stable matching algorithm to assign tasks to vehicles in such a way that not only minimizes task delay but also maintains the quality of the wireless communication channels. Stable matching is chosen since we are facing a multi-objective problem. Stable matching allows both the set of vehicles and the set of tasks to have preference rankings over the other set which can accurately represent both objectives of the problem.

3.2 System Model

We consider one cell which has one Base Station (BS), M users, and N PVs with available computing resources. During high usage hours, the BS will be overloaded and unable to complete all of the tasks offloaded to it by the users. Then, it will have

to offload K tasks to nearby PVs. The time-slot model is adopted where the set of tasks and PVs remain fixed within each time slot while varying across different slots. Therefore, in each time slot we have defined the following variables (it is assumed that this information will be available to the scheduling system):

- N = number of vehicles
- K = number of tasks
- p_i = computational power offered by vehicle i
- t_i = parking time estimation of vehicle i (the assumption of having knowledge of this variable will be discussed)
- w_j = computational power required for task j
- l_{ji} = task completion speed of task j if assigned to vehicle i

Regarding the parking estimation of each vehicle, statistical models are often used to perform estimations on the parking time of vehicles in the PVC environment [32]. However, incentive mechanisms could also be involved in obtaining an estimation of the predicted parking time. There are several existing frameworks for incentive mechanisms in PVC that encourage the participation of PVs and select a set of PVs for offloading. However, no existing system asks for extra information regarding each PV's parking time. It is possible that such information be requested as an addendum to existing incentive mechanisms. For example, Le *et al.* [35] propose an auction as the incentive mechanism for vehicle selection that minimizes social cost. Then, a randomized auction algorithm is used to approximate the winner determination

problem of the proposed auction. Results show that the approximation is within 5% of the optimal value at 40 vehicles.

To summarize, the system selects N vehicles who each offer p_i computational power such that the total computational power meets a given requirement from the BS while minimizing c_i (cost incurred to vehicles). After the selection of participating PVs, each selected PV could be requested to provide an estimate of parking time. PVs will be further rewarded for accurate estimates on top of their reward for offering up their computing resources using a truthful reward function where the closer the estimate is to the actual parking time, the higher the reward will be. An example reward function, r, would be:

Equation 3.1 Example Reward Function

$$r(x, y) = \max(0, x - |x - y|) \tag{3.0.1}$$

where x is the actual parking time and y is the submitted estimated parking time.

It is important to have estimations regarding the parking time of PVs as the system will have a better idea of how long each selected PV will remain parked and assign tasks accordingly. Then, at each time slot, for each available vehicle i, there will be a known t_i that is its estimated remaining parking time. If a PV leaves while computing an offloaded task, the task is interrupted and offloaded again, to another PV. This results in unnecessary overhead and the estimated remaining parking time aims to reduce this by preventing computational tasks being offloaded to PVs that are leaving soon. In short, we will assume that this information is made available to the system, whether by statistical models, incentive mechanisms, or a combination of the two.

To summarize the system model, the task assignment problem can be described

as assigning K tasks to N vehicles where each task is assigned to only one vehicle, each vehicle can be assigned multiple tasks but cannot exceed their computational capacity, assigned tasks' computation time should not exceed their vehicle's estimated parking time.

3.2.1 Problem Formulation

Now, we will formulate the task assignment problem as a weighted multi-objective Integer Linear Problems (ILP).

$$\underset{x,\,y}{\text{minimize}} \quad \alpha \sum_{j=1}^{K} \sum_{i=1}^{N} x_{ji} l_{ji} + \beta \sum_{i=1}^{N} y_i \tag{3.0.2a}$$

subject to

$$\sum_{i=1}^{N} x_{ji} = 1, \qquad j = 1, \ldots, K, \tag{3.0.2b}$$

$$\sum_{j=1}^{K} w_j x_{ji} \leq p_i y_i, i = 1, \ldots, N, \tag{3.0.2c}$$

$$\sum_{j=1}^{K} l_{ji} x_{ji} \leq t_i, \quad i = 1, \ldots, N, \tag{3.0.2d}$$

$$y_i \in \{0, 1\} \qquad i = 1, \ldots, N, \tag{3.0.2e}$$

$$x_{ji} \in \{0, 1\} \qquad j = 1, \ldots, K, i = \ldots, N \tag{3.0.2f}$$

Variables

The two variables in the formulated ILP are:

- $x_{ji} = 1$ if task j is assigned to vehicle i and 0 otherwise

- $y_i = 1$ if vehicle i is assigned at least one task and 0 otherwise

Constraints

The constraints can be summarized as follows:

- (2b): each task is assigned to only one vehicle.
- (2c): the assigned tasks cannot go over vehicle's max load.
- (2d): the task for each vehicle must be able to finish before the vehicle leaves.
- (2e): $y_i = 1$ if vehicle i is assigned a task and 0 otherwise (integrality constraint).
- (2f): $x_{ji} = 1$ if task j is assigned to vehicle i and 0 otherwise (integrality constraint).

This ILP is quite difficult to solve. Following, it will be shown that this problem is NP-Complete.

Objective

Firstly, α and β are constant objective weights. Their values decide which which objective should be prioritized. The first objective is to minimize the task completion time which is crucial in a VEC task assignment environment. The task completion time consists of two parts: computation time, and transmission time (both ways). For a particular task, its computation time, l^{comp}, depends on the vehicle it is assigned to, so l^{comp}_{ji} is the amount of time it takes vehicle i to compute task j. For the transmission time of a task, l^{trans}_{ji}, it depends on the transmission power of the BS.

Equation 3.2 Total Task Completion Speed

$$l_{ji} = l_{ji}^{comp} + l_{ji}^{trans} \tag{3.0.3}$$

Then, total task completion speed is the computation time plus the transmission time as shown in Equation 3.2.

Practically speaking, there may be other conditions placed on the completion speed of the tasks. For example, we may only care about completing tasks before their individual deadlines. Or perhaps, we have to complete each task before its deadline and as fast as possible. However, in this chapter, the only delay metric is completion time without deadlines. The second objective is the number of vehicles used for task assignment. This is because the deployment of tasks and any other form of information between the VEC base station and the parked vehicles will be done through wireless channels which are limited in size. To maintain the quality of communication on these wireless channels, especially in highly populated metropolitan areas, the number of vehicles used is also minimized.

Theorem 1. *(NP-Complete) The formulated ILP is NP-Complete.*

Proof. Consider the corresponding decision version of this problem. That is, given M, is there a task assignment that is within the defined constraints that has an objective value $\leq M$? Certificate: A certificate would be an assignment of tasks to the vehicles denoted by the (K, N) matrix x where $x_{ji} = 1$ if task j is assigned to vehicle i and 0 otherwise. To verify this certificate, we would need to check that the assignment satisfies each constraint and calculate the objective value, that is:

1. First obtain vector y from x where $y_i = 1$ if vehicle i has a task and 0 otherwise

2. Verify each task is assigned to only 1 vehicle

3. Verify assigned tasks do not go over vehicle's max load

4. Verify the assigned tasks finish before the vehicle has to leave

This would take $O(NK)$ time, which means verifying a solution is polynomial. We will now show that bin packing reduces to the formulated ILP [36]. First set the objective weights $\alpha = 0$ and $\beta = 1$. Then, set all $p_i = B$, where B can be any constant. Then set all l_{ji} and $t_i = 0$. Then the optimization problem becomes:

$$\underset{x,\,y}{\text{minimize}} \quad \sum_{i=1}^{N} y_i \tag{3.0.4a}$$

subject to

$$\sum_{i=1}^{N} x_{ji} = 1, \qquad j = 1, \ldots, K, \tag{3.0.4b}$$

$$\sum_{j=1}^{K} w_j x_{ji} \leq B y_i, i = 1, \ldots, N, \tag{3.0.4c}$$

$$y_i \in \{0, 1\} \qquad i = 1, \ldots, N, \tag{3.0.4d}$$

$$x_{ji} \in \{0, 1\} \qquad j = 1, \ldots, K, i = \ldots, N \tag{3.0.4e}$$

Notice that this is an exact formulation of the bin packing problem where w_j is the size of item j and B is the capacity of each bin. The decision bin packing problem is known to be NP-complete. Thus, the decision version of our optimization problem is NP-complete. Therefore, our optimization problem is NP-complete. □

3.3 Proposed Solution

The heuristic proposed is a stable matching based algorithm. The algorithm it is based on has several names including: "Extended Gale-Shapley algorithm", "the Capacitated Gale-Shapley algorithm", "the Roth-Shapley algorithm", and "the deferred acceptance algorithm". Following, it will be referred to as the Roth-Shapley (RS) algorithm [22]. The proposed heuristic is a version of the RS algorithm that is modified to fit the dynamic nature of the PVC environment. The assignment of tasks to vehicles can be described as a many-to-one matching.

Definition (Matching)**.** *A matching A is a mapping from the set of tasks T to the set of vehicles V, $T \to V$, which satisfies all of the following:*

- *for any task $j \in T$, $|A(j)| \leq 1$*
- *for any task $j \in T$, and any vehicle $i \in V$, $A(j) = i$ if and only if $j \in A(i)$*

The proposed algorithm requires both sets V and T to have preference rankings over each other. That is, for all $i \in V$, i must have a preference ranking including all $j \in T$ and vice versa.

Definition (Preference Ranking)**.** *For any vehicle $i \in V$, its preference ranking is a list L including all tasks in T. If task $j \in T$ comes before task $j' \in T$ in L, we say that vehicle i prefers task j to task j'. For tasks in T, their preference rankings are defined vice versa.*

How these rankings are to be computed are discussed later in this section. Next, we will define a stable matching, but first we will define two types of blocking pairs.

Definition (Type 1 Blocking Pair)**.** *Given a matching A,* $(j, i) \in (T, V)$ *forms a type 1 blocking pair if all of the following conditions hold:*

- *task* j *prefers vehicle* i *over* $A(j)$
- *there exists task* k *with vehicle* $i \in A(k)$ *such that vehicle* i *prefers task* j *to task* k *and the removal of task* k *allows the assignment of task* k *onto vehicle* i

The existence of a type 1 blocking pair $(j, i) \in (T, V)$ in a given matching A is unstable since it means that task j can be assigned to a more preferred vehicle and vehicle i can be assigned a more preferred task at the cost of a less preferred task. Apart from the type 1 blocking pair, there is also the type 2 blocking pair.

Definition (Type 2 Blocking Pair)**.** *Given a matching A,* $(j, i) \in (T, V)$ *forms a type 2 blocking pair if all of the following conditions hold:*

- *task* j *prefers vehicle* i *over* $A(j)$
- *vehicle* i *has enough resources to be assigned task* j

The existence of a type 2 blocking pair $(j, i) \in (T, V)$ in a given matching A is unstable since vehicle i is wasteful by not making full use of its resources.

Definition (Stable Matching)**.** *Given a matching A, A is a stable matching if and only if there are no type 1 or type 2 blocking pairs.*

3.3.1 Algorithm Design

A key characteristic of the RS algorithm is the preference ranking made by both parties. Since the preference ranking by either party is made independent of the other, they can be made to represent different objectives which is a desired trait in

multi-objective problems such as the one discussed in this chapter. The crux of the RS algorithm is how the preference rankings are formulated. For the algorithm to be effective, the preference rankings must be a good reflection of the optimization objectives.

- Preference ranking for tasks: prefer vehicles with the most tasks, tie breaks between vehicles by which vehicle completes said task faster. Task j prefers vehicle n over vehicle k if $\sum_{i=1}^{K} x_{in} > \sum_{i=1}^{K} x_{ik}$.
- Preference ranking for vehicles: prefer tasks that complete the fastest on said vehicle. Vehicle i prefers task a over task b if $l_{ai} < l_{bi}$.

The preference ranking for tasks aims to primarily minimize the number of vehicles used which decreases the load on the wireless channels. The preference ranking for vehicles aims to primarily minimize the task completion speed. Together, these two preference mechanics accurately represent the two objectives of the ILP formulated in the previous section.

Consider another matching algorithm, bipartite matching, where the optimization objective is represented only by edge weights. It is extremely difficult to formulate an accurate representation of both objectives when confined to a single form. In our case, it is especially difficult to represent the second objective, number of vehicles used, in such a way since it means we are minimizing the number of nodes covered in a bipartite matching. Hence, we can easily observe the motivation behind designing a heuristic based on the RS algorithm.

The proposed algorithm based on the RS algorithm is deployed to produce a stable matching. The proposed algorithm runs as follows: (The pseudocode is given in Algorithm 1):

1. Put all tasks into a list called unmatched. Go to 2.

2. Update preference rankings for vehicles. Go to 3.

3. Update preference ranking for tasks. Take any task in unmatched, j, and go to 4. If none, end algorithm.

4. Consider task j's most preferred vehicle, i. Go to 5. If task j has no preferred vehicle remaining, remove task j from unmatched list and go to 3.

5. If vehicle i can accommodate task j (has enough computational power and time). Match task j to vehicle i and go to 3. If vehicle i does not have enough time remaining, remove vehicle i from task j's preferences and go to 4. If vehicle i does not have enough computational power, go to 6.

6. Consider all vehicle i's currently matched tasks. Then of these tasks, consider the set of tasks that vehicle i prefers less than task j, call it U. Iterate through U from least preferred to most preferred. If unmatching task k allows vehicle i to have enough resources to be assigned task j, then unmatch task k and match task j. Then remove vehicle i from task k's preference ranking and go to 3. If not, then remove vehicle i from task j's preference and consider task j's next most preferred vehicle and go to 5.

3.3.2 Algorithm Analysis

First, we will analyze the output of the algorithm.

Lemma 2 (No Type 1 Blocking Pairs)**.** *The proposed algorithm produces a matching A that has no type 1 blocking pairs.*

Algorithm 1 Pseudocode of the proposed RS Based Heuristic

Require: preference ranking for both vehicles and tasks

while there are unmatched tasks **do**
  for any unmatched task j **do**
    update preference ranking of each task
    if task j's preferences are empty **then**
      remove task j from the algorithm
    end if
    $i \Leftarrow$ task j's most preferred vehicle
    if $p_i \geq w_j$ **then**
      assign task j to vehicle i
      break
    end if
    if $l_{ji} > t_i$ **then**
      remove vehicle i from task j's preferences
      break
    end if
    if $p_i > w_i$ **then**
      $U \Leftarrow$ tasks currently matched to vehicle i that is less preferred
      than task j in order from least preferred to most preferred
      for task $k \in U$ **do**
        if unmatching task k allows assignment of task j **then**
          unmatch task k and assign task j to vehicle i
          remove vehicle i from task k's preference ranking
          break
        end if
      end for
      remove vehicle i from task j's preference ranking
    end if
  end for
end while

Proof. Suppose for contradiction that produced matching A has type 1 blocking pair $(j, i) \in (T, V)$. Then, consider the point in the algorithm at which task j was assigned to vehicle $l = A(j)$. Since task j prefers vehicle i over vehicle l, that is $\sum_{i=1}^{K} x_{jl} > \sum_{i=1}^{K} x_{ji}$, vehicle i must have been considered before vehicle l. Then, vehicle i must have rejected task j at this point which means for all tasks k with $A(k) = i$ either of the following is true: no tasks who is assigned to vehicle i is less preferred than task j, or $p_i + w_k < w_j$. Then vehicle i would have been removed from task j's preferences. This contradicts the assumption that task j prefers vehicle l over vehicle i. Therefore, the matching A cannot have a type 1 blocking pair $(j, i) \in (T, V)$. □

Lemma 3 (No Type 2 Blocking Pairs)**.** *The proposed algorithm produces a matching A that has no type 2 blocking pairs.*

Proof. Suppose for contradiction that produced matching A has type 2 blocking pair $(j, i) \in (T, V)$. Then, consider the point in the algorithm at which task j was assigned to vehicle $l = A(j)$. Since task j prefers vehicle i over vehicle l, that is $\sum_{i=1}^{K} x_{jl} > \sum_{i=1}^{K} x_{ji}$, vehicle i must have been considered before vehicle l. Then, it must be that vehicle i rejected task j which means $p_i < c_j$ or $t_i < l_{ji}$. Then vehicle i would have been removed from task j's preferences which is a contradiction to the assumption of the existence of type 2 blocking pair (j, i). □

Theorem 4 (Stable Matching)**.** *The proposed algorithm produces a stable matching A.*

Proof. According to Lemma 1 and Lemma 2, there are type 1 or type 2 blocking pairs in the produced matching A. Therefore, the produced matching is stable. □

Table 3.1: Experiment Variables

Variables	Experiment Settings
p_i	random between 20-25
t_i	random between 15-30
w_j	random between 4-5
l_{ji}	random between 1-20

Now, we will analyze the termination condition and the complexity of the proposed algorithm.

Theorem 5 (Termination)**.** *The proposed algorithm terminates after at most NK iterations.*

Proof. First, notice that in each iteration of the algorithm, a task is either matched to a vehicle (may be after the unmatching of another task) or is removed from the algorithm. That is to say, the number of unmatched tasks never decreases in any iteration of the algorithm. Then, for an infinite loop to exist, there must be an infinite number of times where a task is unmatched from a vehicle. However, whenever a task is unmatched from a vehicle, it is removed from the vehicle's preference ranking. That is, the removed task will never be assigned to the vehicle it was once unmatched with. Therefore, there can be at most NK number of unmatchings and thus, an infinite loop is impossible and the algorithm is guaranteed to terminate. Furthermore, for any given iteration, for unmatchings to occur, some task must have been assigned to some vehicle. There can be at most NK number of such assignments since each task can be assigned to each vehicle at most once. Therefore, the algorithm will take NK iterations to terminate in the worst case. □

3.4 Experimental Results

There are two baseline algorithms that are used for evaluation. The first is a randomized algorithm that randomly assigns a vehicle as the "current" vehicle. Then, it will iterate through the tasks in an arbitrary order, assigning each task onto the "current" vehicle. If a task cannot fit onto the "current" vehicle, the system will choose another random vehicle as the "current" vehicle. It will be referred to as the next fit algorithm. The second baseline algorithm is a greedy algorithm that organizes tasks from largest to smallest, then orders the vehicles from most to least computational power offered. Then, the system will iterate through the tasks in order and, for each task, it will iterate through the vehicles in order until a vehicle is found able to take on the task. This algorithm is based on a greedy algorithm for the bin packing problem (the formulated problem is similar to the bin packing problem as shown by the NP-complete proof in Section 2). The complexity of these algorithms are $O(K + N)$ and $O(KN)$ respectively.

3.4.1 Example Case

A demonstration of all three algorithms are given in the following example scenario:

- 6 vehicles
- 12 tasks
- delay objective weight $\alpha = 1$, number of vehicles used objective weight $\beta = 10$

In this scenario, the vehicle to task ratio is 2 which simulates a balanced scenario

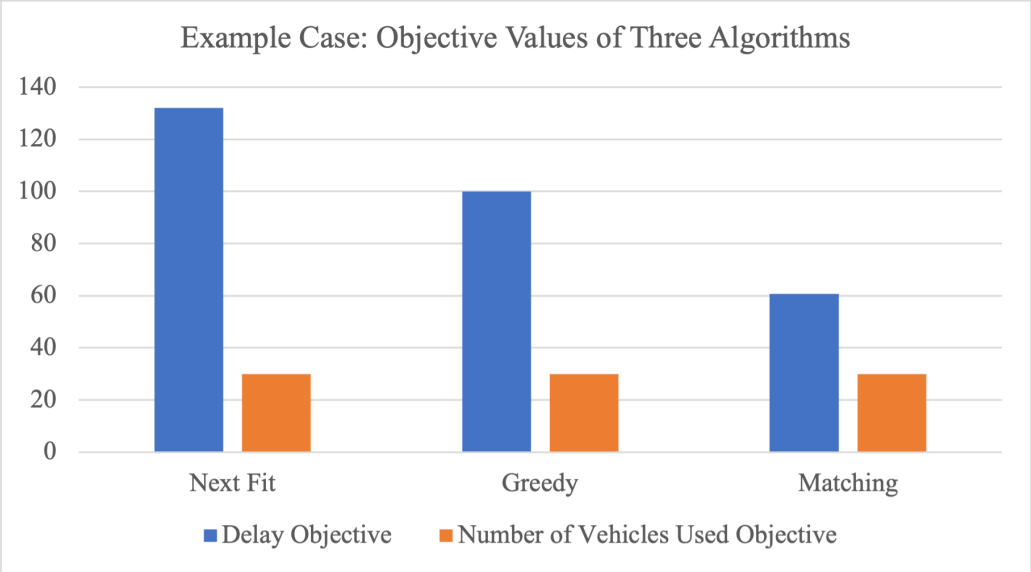

Figure 3.1: Example Case Objective Value Comparison

(neither peak hours nor off-hours) and the objective weights are such that both objectives are valued similarly. The exact configurations of the vehicles and tasks are randomized as displayed in Table 3.1.

The objective values of the three solutions are displayed in Fig. 3.1. Due to space limitations, the exact solution matrices of each algorithm in this example case are not shown but it is notable that the solutions were all different from each other. In terms of performance, all three algorithms performed the exact same in terms of the number of the vehicles used objective. This is to be expected to be similar, though it is only the exact same in very small settings. In terms of the delay objective, the matching algorithm performed better than the greedy algorithm which performed better than the next fit algorithm.

To evaluate how well the RS based heuristic performs, we will test its objective value against that of the optimal to find the approximation loss. However, due to

the high computational demands of computing the optimal solution of an ILP at large scales, this evaluation had at most 50 vehicles. The objective weights will be 1 and 10. This evaluation will be done at three different vehicles to task ratios to emulate how busy the environment is. The three ratios are: 1:1 (abundance of computational resources compared to tasks); 1:2 (moderate amount of computational resources compared to tasks); and 1:3 (scarcity of computational resources compared to tasks).

The variables for the experiments are once again randomized as indicated in Table 3.1 To minimize the effect of the randomized variables, each instance of the experiment was ran 20 times and the averaged results of the approximation loss experiment are displayed in Fig. 3.2. The y-axis represents how far from the optimum the results are. For example, the matching algorithm (1:1) gives a solution that is 1.7 times the optimum. Evident in these results, the matching algorithm performs significantly better than the other two algorithms, especially in settings with more tasks. Comparing different task to vehicle ratios, more tasks correlate to worse performance. This is to be expected as it is much more difficult to assign tasks optimally when computational resources are more limited.

3.4.2 Large Scale Experimentation

To examine the performance of the RS based heuristic at a large scale, several experiments on various settings were performed and evaluated against the two baseline algorithms. The same three different vehicle to task ratios were tested (1:1, 1:2, 1:3). Again, these vehicle to task ratios are designed to emulate how busy the environment is.

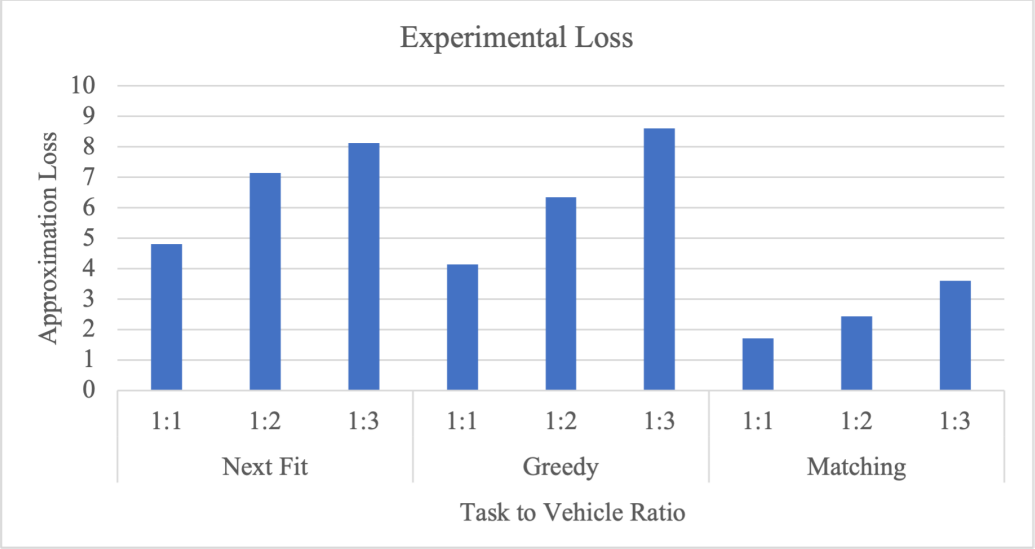

Figure 3.2: Average Experimental Loss with 50 Vehicles

Two different pairs of objectives weights were tested. These different objective weights are designed to emulate different valuations of the objectives (the first number is the weight of the delay objective α and the second number is the weight of the "number of vehicles used" objective β): (1, 10), (1, 100).

- Objective weights: 1, 10. The delay objective is much larger than the "number of vehicles used" objective and dominates the optimization (for situations where we mostly care about delay and not about the number of vehicles used).

- Objective weights: 1, 100. "number of vehicles used" objective is much larger than the delay objective and dominates the optimization (for situations where we mostly care about the number of vehicles used).

Each experiment will be run on a scale from 50 vehicles to 500 vehicles at intervals of 50 using all three algorithms. The two different objectives are evaluated separately. The variables for the experiments are once again randomized as indicated

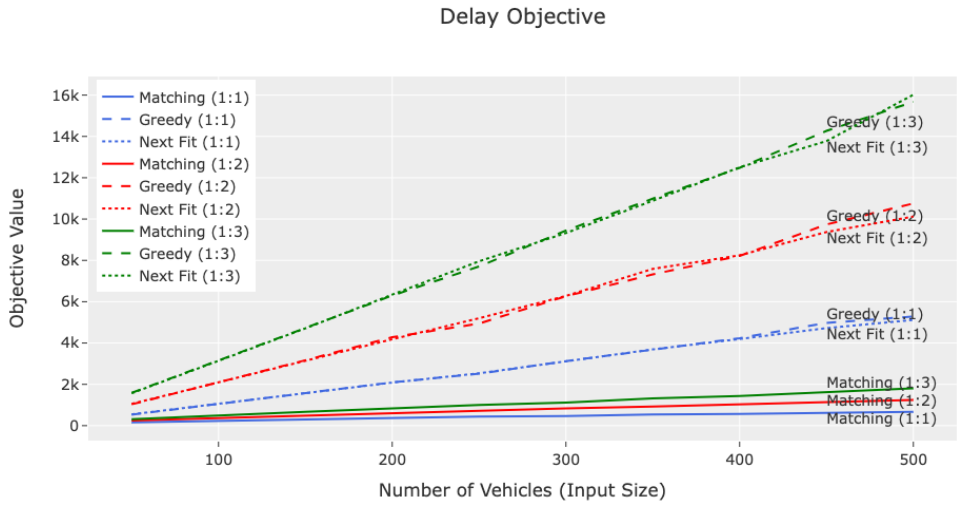

Figure 3.3: Objective Weights $\alpha = 1, \beta = 10$, Delay Objective

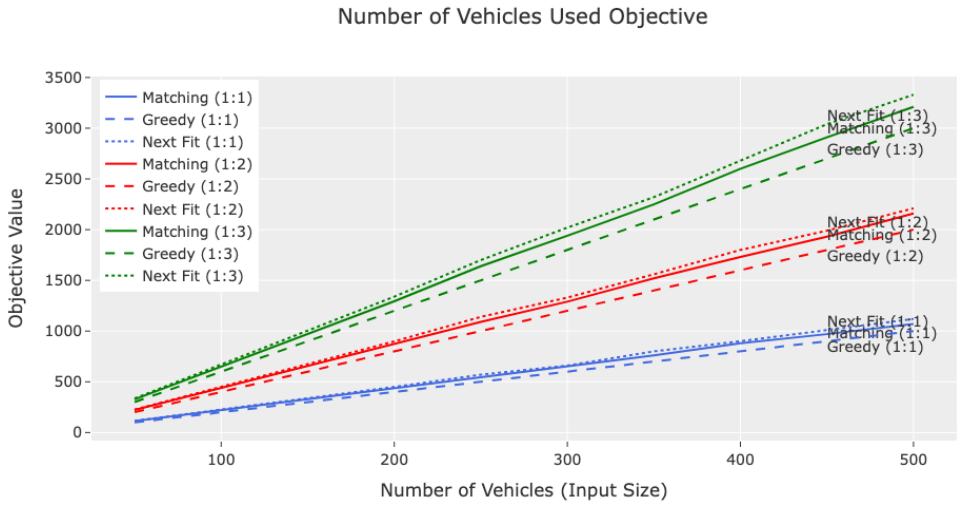

Figure 3.4: Objective Weights $\alpha = 1, \beta = 10$, Number of Vehicles Used Objective

in Table 3.1. To minimize the effect of the randomized variables, the experiment was performed 20 times. The Coefficient of Variation (CV) was at most 0.15, indicating very low variance in the data sample. Furthermore, any particular data point was at

most 31% away from the mean. Therefore, it is reasonable to conclude that the randomness of the initialized variables has little impact on the result of the experiments. The experimental results for objective weights (1, 10) are displayed in Fig. 3.3 and Fig. 3.4. In this case, where the delay objective dominates the number of vehicles used objective, the RS based matching algorithm performs significantly better in terms of delay while performing similarly to the other two algorithms in terms of the number of vehicles used.

In the second case, where the number of vehicles used objective has weight 100, the simulation results are displayed in Fig. 3.5 and Fig. 3.6. Again, the proposed RS based algorithm outperforms the other two algorithms by a large margin while performing similarly in terms of the other objective.

The reasoning behind this performance is because both baseline algorithms (greedy and next fit) are similar to bin packing algorithms and will inherently prioritize the number of vehicles used objective based on their design. While, the RS based algorithm aims to minimize both the delay objective and the number of vehicles used objective simultaneously. As a result, the RS based algorithm produces much less delay than the other two algorithms. Also, performing similarly to these two baseline algorithms in terms of the number of vehicles used objective can be considered a success for the proposed algorithm. Furthermore, the busiest vehicle to task ratio simulated is 1:3, which results in some vehicles not needing to be used.

Another simulation is performed where the vehicle to task ratio is 1:5 to replicate an extremely busy environment where most vehicles need to be used. Fig. 3.7 displays the results of this simulation. We can see that, on average, the proposed RS based algorithm performs slightly better than the two baseline algorithms. In this

type of setting, where most available vehicles need to be used, assigning the right tasks to the right vehicles becomes increasingly important. Therefore, the RS based algorithm's performance in the number of vehicles used objective is slightly better than the other two baseline algorithms while still performing significantly better in the delay objective.

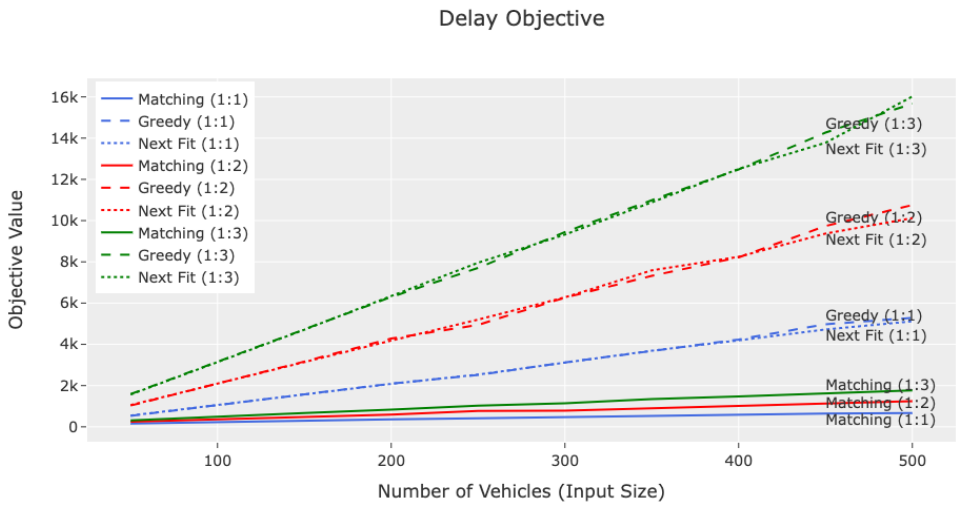

Figure 3.5: Objective Weights $\alpha = 1, \beta = 100$, Delay Objective

3.5 Conclusion

This chapter proposes a formulation of the task assignment problem in the PVC environment as a weighted multi-objective optimization problem that aims to minimize both task delay and wireless channel load. Then, a heuristic based on the RS algorithm is proposed and evaluated against two other baseline algorithms on various simulation settings on a scale of up to 500 vehicles and 1,500 tasks.

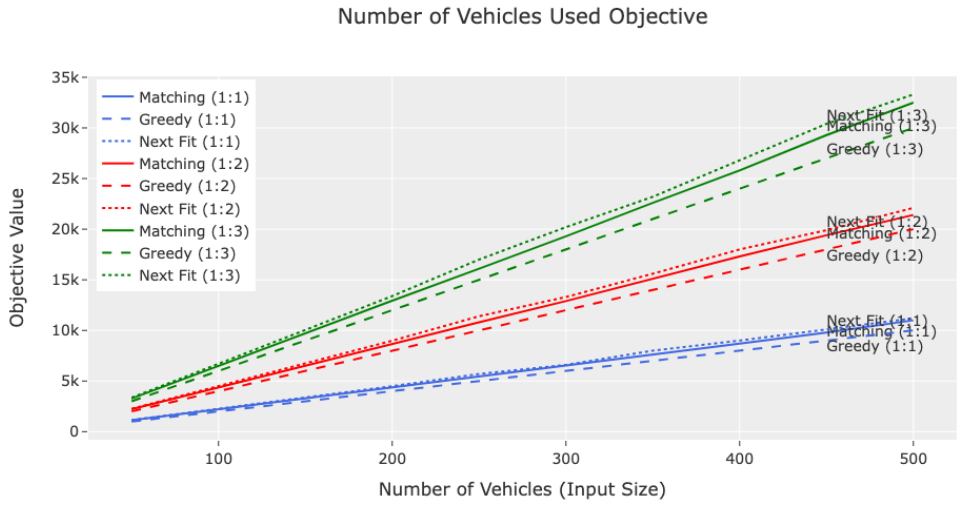

Figure 3.6: Objective Weights $\alpha = 1, \beta = 100$, Number of Vehicles Used Objective

Figure 3.7: Objective Weights $\alpha = 1, \beta = 10$, Both Objectives

Chapter 4

An Online Fair Resource Allocation Solution for Fog Computing

The main contributions of this chapter can be summarized as follows:

- Formulate multi-resource, multi-server, and heterogeneous task resource allocation problem as a fairness maximizing problem using the Dominant Resource Fairness (DRF) scheme.
- Propose three low complexity heuristics for different types of tasks: ordered/unordered, splittable/unsplittable.
- Evaluate the proposed heuristics against three baseline scheduling algorithms.

The rest of this chapter is organized as follows: we will first discuss some related works, then describe the system model, formulate the problem, propose our solution, analyze the proposed solution, display the simulation results, and conclude the chapter.

4.1 Related Works

Due to the dynamic nature of the numerous edge devices in a fog environment, this chapter will focus on online scheduling algorithms, more specifically, real-time scheduling algorithms, where the tasks arrive to the system in an online manner. There are two main approaches to online scheduling, the first is termed completely reactive scheduling where tasks are scheduled as they arrive based on a set of predefined rules or heuristics [37]. This approach easily handles randomly arriving tasks but often performs far from optimal due to the reactive nature of the algorithm [38]. The second approach is predictive-reactive scheduling where an initial schedule is constructed for existing tasks and then revised when certain events occur such as new task arrivals or machine breakdowns. Predictive-reactive scheduling usually performs better than completely reactive scheduling, but due to the dynamic nature of the fog computing environment where large amount of tasks are continuously arriving, it is inappropriate to adopt the predictive-reactive scheduling approach [39] [28].

There are many works regarding online scheduling techniques in various settings, Ouelhadj and Petrovic [38] give a detailed survey of such methods. Many of these scheduling schemes focus on some measure of task completion time (makespan, lateness, etc). However, resource fairness is an objective that is often critical in computing environments. Allocating resources with respect to fairness and efficiency is a fundamental problem in the design of fog computing environments. Maintaining fairness ensures that a balanced allocation of resources amongst the tasks where no tasks are starved at the expense of another. Furthermore, resource fairness also encourages maximizing the utilization of all available resources. Maintaining resource fairness is an area quite heavily explored during the development of cloud computing several

years ago. In 2009, Microsoft [40] published the "Quincy" scheduler that maps a scheduling problem to a graph data structure that computes the optimal schedule that maximizes a global cost function that includes locality and resource fairness. In 2010, Zaharia *et al.* [41] proposed an algorithm called "Delay Scheduling" to address the conflict between locality and fairness for the 600-node Hadoop cluster at Facebook.

However, since fog computing is an area that has only recently emerged, resource fairness within a fog environment is a much less explored topic. In 2018, Zhang *et al.* [42] proposed Fair Task Offloading (FTO) scheme that minimizes task delay and energy consumption while maintaining fairness between network nodes. In 2019, Mukherjee *et al.* [43] proposed a scheduling policy that maximizes the number of tasks completed before their deadline while maintaining fairness by keeping both high priority and low priority task queues stable. However, both of these scheduling schemes only maximize fairness between different fog nodes instead of between participating users. This is also true for the Quincy scheduler and the Hadoop Delay Scheduling algorithm.

Fairness between users who request tasks can be evaluated by DRF, an index developed by Ghodsi *et al.* in 2011 [2]. The DRF scheme is a multi-resource generalization of max-min fairness. According to the DRF scheme, in a multi-resource environment where tasks have heterogeneous resource demands, a user's dominant share is the maximum global share that the user has been allocated of any resource. In essence, the DRF scheme aims to have all users have equal dominant share values in any allocation. In their publication defining the DRF scheme, Godhsi *et al.* [2] considered many other fairness schemes including Asset Fairness and Competitive

Equilibrium from Equal Incomes (CEEI). Asset Fairness aims to equalize the aggregate resource value allocated to each user assuming that equal shares of different resources are worth the same. That is, 1% of resource 1 is equivalent to 1% of resource 2. CEEI, the preferred method of fair resource allocation in microeconomic theory [2], initially allocates $\frac{1}{n}$ of each resource to each user and subsequently, each user can trade their resources with other users. Godhsi *et al.* [2] found that the DRF scheme is the only fairness scheme that satisfies the following four key qualities:

- Sharing Incentive: every task's allocation is better than that obtained by dividing every resource evenly between tasks
- Strategy-proof: tasks cannot get better allocation by lying about their requirements
- Pareto Efficiency: all available resources are allocated subject to satisfying the other properties, and without preempting existing allocations
- Envy Free: no task prefers the allocation of another task

Asset Fairness violates the sharing incentive quality and CEEI violates the strategy-proof quality. All of the above qualities are desired in a fog computing environment and would increase the Quality of Service (QoS) for users. So far, it remains a difficult problem to design an online resource allocation scheme while maintaining multi-server multi-resource fairness. In 2019, Bian *et al.* [28] tackle this problem and proposed "FairTS", a fair online task scheduling scheme that uses DRF as their fairness scheme. FairTS uses a reinforcement learning approach to minimize average task slowdown while maximizing the minimum dominant resource share of each task. However, as a

machine learning based algorithm, FairTS falls behind heuristics in terms of computation speed, a much desired attribute in fog computing environments. Furthermore, FairTS is designed for a simplified model where all resources are located in one computer/server. However, in a realistic fog setting, task requests would be sent to be processed by a collection of fog servers. In 2014, Wang *et al.* [3] proposed "Dominant Resource Fairness with Heterogeneous Servers", which is a generalization of DRF for multiple heterogeneous servers. This generalization preserves the strategy-proof, Pareto efficiency, and envy free qualities of DRF. The sharing incentive strategy is not well defined in a multi-server setting as there is are very many different ways to evenly divide the resource pool. This paper uses the generalized version of DRF put forward by Wang *et al.* [3] since the environment we consider also includes multiple heterogeneous servers. However, the work proposed in this chapter differs from Wang *et al.*'s work in 2014 since we consider a system with non-divisible and heterogeneous tasks.

In 2017, Östman [44] proposed the Distributed Dominant Resource Fairness (DDRF) scheme that uses a gradient network topology overlay to create a dynamic directed sorted graph based on their users' resource share. The proposed scheme allows multiple servers to allocate resources in parallel to achieve faster allocation time. However, similar to the proposed DRF scheme, the environment considered is also very basic. It only considers the case where each user will have an unbounded number of divisible homogeneous tasks. In fact, after the proposal of the DRF scheme in 2011, its assumption of divisible tasks is a heavily discussed topic as it contradicts the Leontief preferences [45].

In practice, users will usually have indivisible tasks that are also heterogeneous.

Zaharia *et al.* [41] revealed the resource usage profiles of tasks in a 2000-node Hadoop cluster at Facebook over one month (October 2010) and showed that the requested tasks have very different resource requirements. Furthermore, there may be other requirements and characteristics for each user such as ordering and splittablility. A user's tasks can be ordered where they must be completed in a task chain. A user's tasks can also be splittable where they can be allocated more than their required amount of resources to be completed faster. This chapter will consider the fair resource allocation problem under these more realistic conditions.

4.2 System Model

In this section, we will define the system model used in this chapter which includes the DRF scheme proposed by Ghodsi *et al.* [2] and the generalization made by Wang *et al.* [3]. This chapter will consider a time step system, where a set of users will arrive to the system at the beginning of each time step in an online manner. Before the system model is formally defined, we first give an intuitive description of how the system operates at each time step:

1. A set of users arrive to the system (may be an empty set). Each user will have a set of tasks to be completed (assumed to be non-empty). Information regarding these tasks are not known to the system until their arrival.

2. A user's tasks can be ordered or unordered, splittable or unsplittable. We will consider all four possible cases. We will also assume tasks are non-preemptive, that is, they cannot be stopped once they have started computing.

3. The system will then allocate a portion of resources on any amount of servers

to each user (new and existing) based on the DRF scheme. This means that existing users can have their allocation changed.

4. After the resources have been allocated, the tasks will be run for one time step using the resources that they have been assigned.

5. Any users who have completed all their tasks will now leave the system.

6. Repeat for next time step.

In a fog computing system, n users will be connected to k nearby fog node servers $S = \{1, ..., k\}$. Each server contributes m resources $R = \{1, ..., m\}$. Each server j has resource capacity $c_j = (c_{j1}, ..., c_{jm})$ where each element is represented as a fraction of the total amount of said resource in the entire system. That is, for every resource, the total capacity of all servers added together is 1 as shown in Equation 4.1.

Equation 4.1 Total Resource Capacity

$$\forall s \in \{1, \ldots, m\}, \sum_{x=1}^{k} c_{xs} = 1 \tag{4.0.1}$$

Each user $i = 1, 2, ...$ will have a set of heterogeneous tasks that need to be completed. As mentioned previously, this chapter uses the DRF scheme to evaluate fairness in resource allocation. To define this fairness scheme, each task $n = 1, 2, ...$ will be defined by the following characterizations [3]:

- Task n's arrival time is t_n^{arr}.
- Task n's time steps required for computation is l_n (assuming resource demands are met).

- Task n's resource demand vector,D_n, is the transpose of the vector $(D_{n1}, ..., D_{nm})^T$ where each D_{nr} is a fraction over the total amount of resource r in the entire system (assumed to be non-negative).

- Task n's global dominant resource $r_n^* := \arg\max_{r \in R} D_{nr}$. That is, r_n^* refers to the resource that user n demands the most of (with respect to the overall resource pool).

- Task n's normalized resource demand vector $d_n = (d_{n1}, ..., d_{nm})^T$ where $d_{nr} = D_{nr}/D_{nr_n^*}$ for each resource $r = 1, ..., m$. In other words, normalizing the vector D_n results in the vector d_n.

It is important to note that a task's resource demand vector, D_n, is defined as a fraction over the total amount of resources in the system. For example, consider a system with 10 units of CPU and 100 units of memory with a task, n, that requires 1 unit of CPU and 2 units of memory. Task n's resource demand vector, D_n, is $(\frac{1}{10}, \frac{2}{100})$ or $(\frac{1}{10}, \frac{1}{50})$. Then, task n's global dominant resource r_n^* is CPU since task n requires 10% of all CPU units in the system and only 2% of all memory units in the system. This may seem counter intuitive as task n requires more units of memory than CPU but the DRF scheme used in this chapter deals in percentages with respect to the whole system. Next, we will define the key metric used to measure the fairness of allocation. Under the DRF scheme, each user is characterized by their *global dominant share* [2] [3].

Equation 4.2 Global Dominant Share [2] [3]

$$G_i(A_i) := \sum_{j \in S} \min_{r \in R} \{A_{ijr}/d_{ir}\} \tag{4.0.2}$$

Definition. *Under the DRF scheme, the global dominant share of user i under allocation A_i is defined in Equation 4.2 where, for each server j and user i, $A_{ij} = (A_{ij1}, ..., A_{ijm})^T$ is the allocation vector and A_{ijr} is the share of resource r allocated to user i on server j represented as a fraction over the total amount of resource r in the entire system. And, $A_i = (A_{i1}, ..., A_{ik})$ is the allocation matrix for user i.*

In other words, the global dominant share of a user is how much of the system's resource has been given to the most limiting resource of that user. An example case will be given in the next subsection to further explain the global dominant share definition. Due to the heavy notation used in this section, Table 4.1 gives all the relevant notation used in this chapter. The notation used in this chapter is very similar to the notation used in Ghodsi *et al.*'s [2] and Wang *et al.*'s [3] work in defining the DRF scheme.

4.2.1 Example Case

To illustrate the idea behind the DRF scheme, a simple example with 2 servers and 2 users is given. The setting is as follows:

- Server 1 has 5 CPU units and 11 memory units
- Server 2 has 11 CPU units and 5 memory units
- Server 1 capacity vector c_1: $(\frac{5}{16}, \frac{11}{16})$
- Server 2 capacity vector c_2: $(\frac{11}{16}, \frac{5}{16})$
- User 1's task requirements: 1 CPU units and 0.5 memory units
- User 2's task requirements: 0.5 CPU units and 1 memory units

c_j	resource capacity of server j
t_n^{arr}	task n's arrival time
l_n	task n's time steps required for computation (execution time)
D_n	task n's resource demand vector represented as a vector of fractions over the total amount of resources in the system
r_n^*	task n's global dominant resource (the resource that task n is demanding the most of in terms of percentage)
d_n	task n's normalized resource demand vector (D_n normalized)
A_i	user i's received allocation represented as an (m, k)-matrix where m is the number of resources and k is the number of servers
$G_i(A_i)$	user i's global dominant share under allocation A_i which is a measure of the amount of resources allocated to user i (we are trying to maximize this value for each user)
z_i	user i's size characteristic (only relevant for user's with ordered tasks or task chains) and is a vector representing the largest demands of each resource in the entire task chain
$F(n, j)$	the fitness of assigning task n to server j and is defined as $F(n,j) = \|\|d_n - c_j/c_j^*\|\|$ where c_j/c_j^* is the normalized vector of the remaining resources of server j and $\|\|.\|\|$ is the 2-norm of a vector.
$F'(i, j)$	the fitness of assigning task chain i to server j and is defined as $\|\|z_i/z_i^* - c_j^*/c_{j1}^*\|\|$ where z_i/z_i^* is the normalized size characteristic of user i, c_j/c_j^* is the normalized vector of the remaining resoures of server j, and $\|\|.\|\|$ is the 2-norm of a vector.

Table 4.1: Table of Notations

- Assume both users have an unbounded number of these divisible tasks
- User 1's resource demand vector D_1: $(\frac{1}{16}, \frac{1}{32})$
- User 2's resource demand vector D_2: $(\frac{1}{32}, \frac{1}{16})$
- User 1's normalized resource demand vector d_1: $(1, \frac{1}{2})$
- User 2's normalized resource demand vector d_2: $(\frac{1}{2}, 1)$

From the normalized resource demand vectors, we can see that CPU is user 1's global dominant resource r_1^* and memory is user 2's global dominant resource r_2^*. Optimally, it is obvious that user 1 should be allocated all of server 2 and user 2 should be allocated all of server 1. Following this allocation, user 2's allocation is $(\frac{5}{16}, \frac{11}{16})$. User 2's global dominant share is $\min\{\frac{5/16}{1/2}, \frac{11/16}{1}\} = \frac{10}{16}$ (as defined in Equation 4.2), which is associated with the CPU resource. Similarly, user 1's global dominant share is $\frac{10}{16}$. Notice that the two users' global dominant shares are the same and also the maximum that they can be. So, the system would allocate all of server 1 to user 2 and let it run 10 tasks and allocate all of server 2 to user 1 and let it run 10 tasks as well. This will continue until a new user arrives and the resources will be reallocated. This allocation results in equal global dominant share and is displayed in Fig. 4.1. We can see that it achieves extremely high resource utilization of $\frac{15}{16}$ or around 94%.

Instead of using global dominant share, there exists a simpler way to extend the DRF scheme to our multi-server environment by applying the DRF scheme to each server individually. However, it is easy to show that this approach is naively inappropriate using the same example case as above.

Following this approach, we would consider the two servers separately. In server 1, we can see that both users' dominant resource is CPU. So, to maximize the minimum dominant share, both users will receive half of the available CPU which would be 2.5 each. Similarly, in server 2, we see that both users' dominant resource is memory. Then, they will both receive half of the available memory resource which is also 2.5 each. So, user 1 will be able to run 2.5 tasks on server 1 and 5 tasks on server 2. User 2 will be able to run 5 tasks on server 1 and 2.5 tasks on server 2. Both users will be

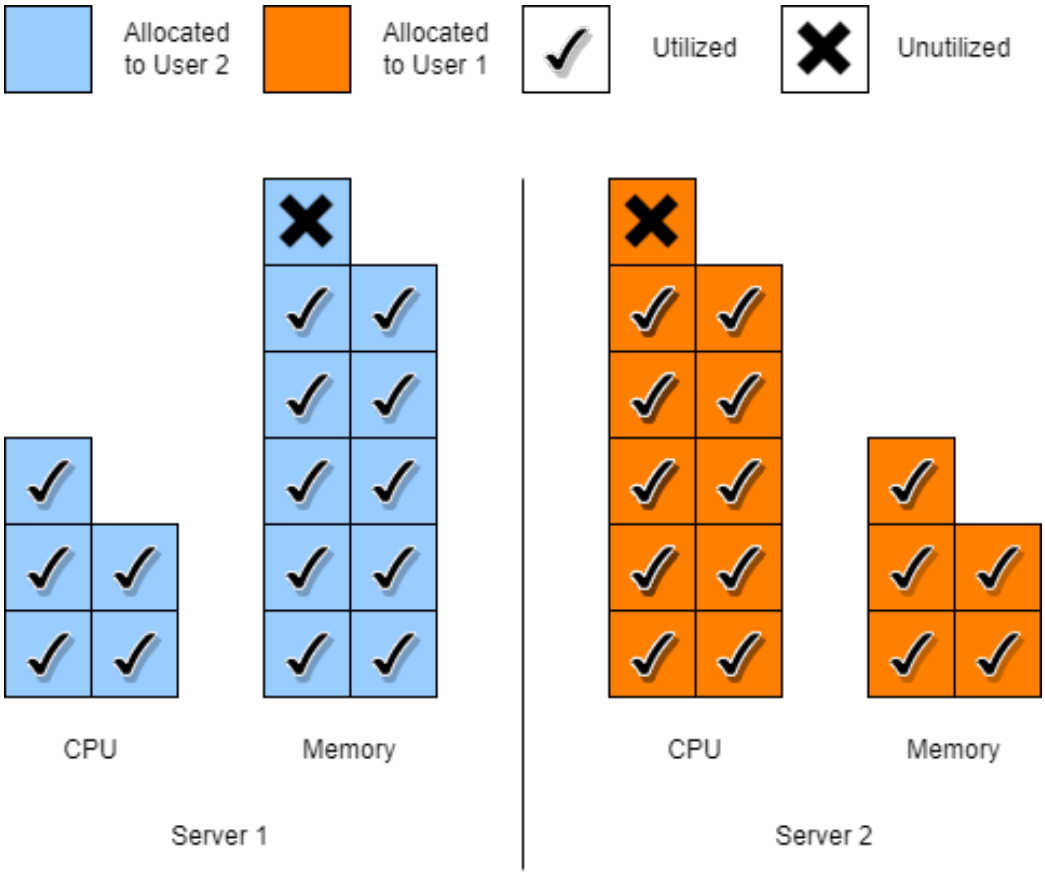

Figure 4.1: Optimal Allocation Under DRF

able to run 7.5 tasks each. This is far worse than the solution found previously. This naive allocation is displayed in Fig. 4.2. Evidently, it achieves much worse resource allocation than the previous solution provided by the DRF scheme. The resource utilization for this allocation is $\frac{22.5}{32}$ or around 70% which is significantly worse than the previous allocation.

4.3 Problem Formulation

The fairness maximizing optimization problem is defined as follows:

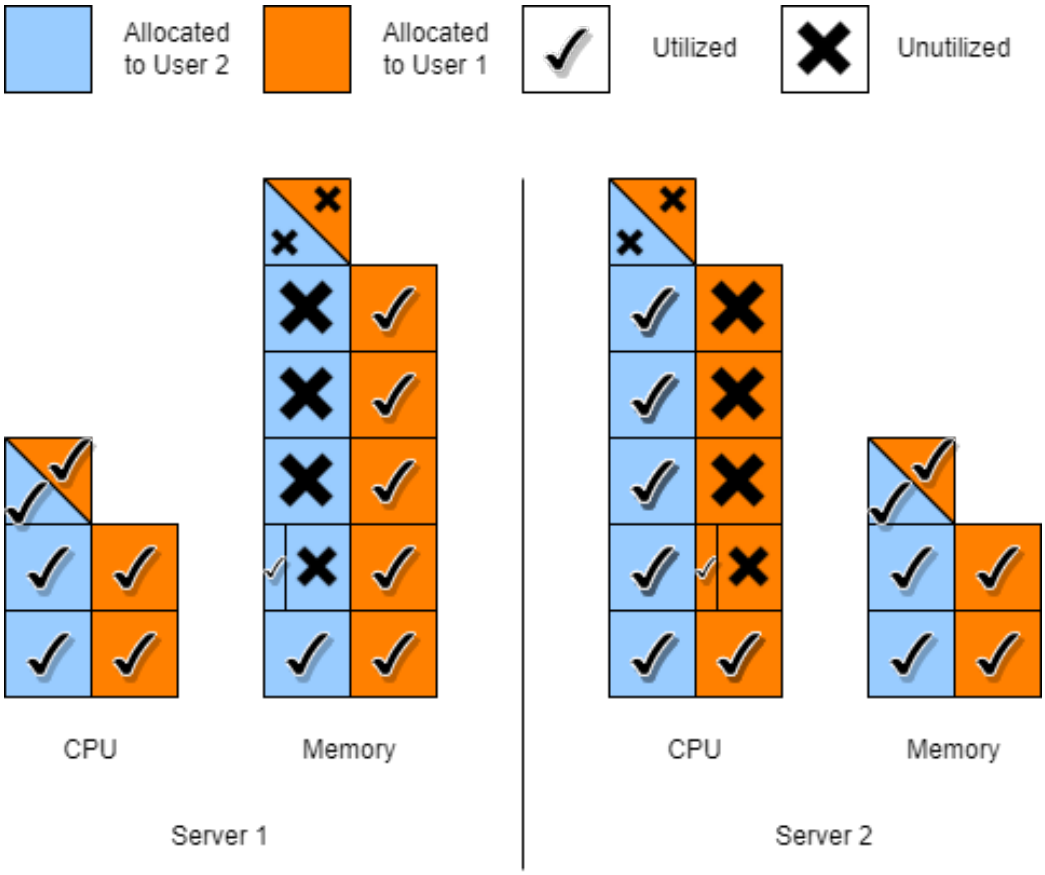

Figure 4.2: Naive Allocation

$$\text{maximize} \quad \min_{i \in T} G_i(A_i) \tag{4.0.3a}$$

subject to

$$\sum_i A_{ijr} \leq c_{jr}, j = 1, ..., k, r = 1, ..., m, \tag{4.0.3b}$$

$$A_i \in \{0, 1\}^{mk}, i \in T \tag{4.0.3c}$$

Recall that $A_i = (A_{i1}, ..., A_{ik})$ is the allocation matrix for user i and $G_i(A_i) :=$

$\sum_{j \in S} \min_{r \in R}\{A_{ijr}/d_{ir}\}$ is the *global dominant share* of user i. Objective 4.0.3a aims to maximize the minimum global dominant share among all users. Constraint 4.0.3b makes sure that all feasible allocations do not exceed the resource capacity of each server. Constraint 4.0.3c is the integer constraint for the problem.

Theorem 6. *The formulated fairness maximizing optimization problem is NP-hard.*

Proof. Consider the bin packing problem [36] with bin size B and item weights w:

$$\underset{x,\,y}{\text{minimize}} \quad \sum_{i=1}^{N} y_i \tag{4.0.4a}$$

subject to

$$\sum_{i=1}^{N} x_{ji} = 1, \qquad j = 1, \ldots, K, \tag{4.0.4b}$$

$$\sum_{j=1}^{K} w_j x_{ji} \leq B y_i, i = 1, \ldots, N, \tag{4.0.4c}$$

$$y_i \in \{0, 1\} \qquad i = 1, \ldots, N, \tag{4.0.4d}$$

$$x_{ji} \in \{0, 1\} \qquad j = 1, \ldots, K, i = \ldots, N \tag{4.0.4e}$$

Now, consider a single resource instance of our resource allocation problem where items and bins correspond to tasks and servers, respectively. The single resource corresponds to the size limitations in the bin packing problem, that is, the size of a item w_j corresponds to the resource requirement of a task and the capacity of a bin B corresponds to the resource capacity of a server. Then, the bin packing problem exactly corresponds to a single resource instance of our problem where each user has exactly one task. The bin packing problem (decision version) asks if all items can fit

into A bins where A is a given constant. Notice that fitting all tasks onto A servers in our resource allocation problem would produce a unique objective value which would be the global dominant share of the smallest task. This value is unique since if any task was to remain unassigned to a server, the minimum global dominant share would be 0. Therefore, the following questions are equivalent:

- Can all tasks be assigned onto A servers?
- Can there be such a task assignment onto A servers where the minimum global dominant share is the minimum w_j over all $j = 1, \ldots, K$?

Therefore, we can see that the bin packing decision problem can be reduced to a decision instance of the fairness maximizing optimization problem. Since the bin packing problem is known to be NP-hard, our formulated resource allocation problem is NP-hard as well.

4.4 Proposed Solution

As mentioned previously, there are different types of tasks that can be sent to the system. A user's tasks can be *splittable* which means they can be allocated an integer multiple of its resource requirements to complete proportionally faster (two times the required resources means two times the completion speed) or *unsplittable* which means that they complete at the same speed as long as its requirements are satisfied. A user's tasks can also be *ordered* where each task must be completed in a specific order or *unordered* where the tasks can be completed in any order and at the same time. For ordered tasks, it is likely that data must be transferred between the tasks, as such they must be completed on the same server since it will be inefficient

and expensive to offload tasks of the same chain onto different servers. Furthermore, a task chain cannot be interrupted once it has started. In this section, we will consider four different cases:

1. all users have unsplittable and unordered tasks
2. all users have splittable and unordered tasks
3. all users have unsplittable and unordered tasks
4. all users have splittable and ordered tasks

4.4.1 Unsplittable and Unordered

In this section, we will assume that all users have a set of unsplittable and unordered tasks. This means that tasks from the same user need not be on the same server since they are unordered. This is the least restricted case where tasks can be scheduled on any server and in any order. It is also meaningless to assign a task more than its required resource since it does not increase its completion speed. Algorithm 2 is a proposed heuristic to schedule tasks in this scenario and is inspired by Wang *et al.*'s work [3] since the unsplittable and unordered case is similar to their considered problem. However, since they do not consider heterogeneous tasks, their proposed heuristic is not ideal for our case. The key difference in our proposed Algorithm 2 is the ordering of each user's task from most resource demanding to least resource demanding.

In Algorithm 2, at each time step, the system repeatedly picks the user with the lowest global dominant share and schedules its most resource demanding task until there are no remaining tasks left or there are no more resources available. By

Algorithm 2 Unsplittable and Unordered

```
1: for each time step do
2:     new users arrive
3:     available tasks ← all unassigned tasks of every user
4:     while there are available tasks do
5:         current_user i ← user with the smallest global dominant share G_i(A_i)
6:         current_task n ← available task of current_user i with the largest resource
   demands ∑ d_n
7:         if no server can fit current_task n then
8:             remove current_task n from available tasks
9:         else
10:            assign current_task n to best fit server
11:            update remaining server resources and available tasks
12:        end if
13:    end while
14: end for
```

picking the user with the lowest global dominant share (line 5), we aim to maximize the minimum global dominant share across all users, which is the objective of our optimization problem. Instead of picking any task, the most resource demanding tasks are picked first to further minimize the variance between the global dominant shares (line 6). Consider the following example:

- 1 server with 10 units of a single resource
- User 1 has three small tasks and one big task. The small tasks each require 1 resource unit and the big task requires 4 resource units.
- User 2 has the same 4 tasks.

If we schedule the tasks based on some arbitrary order, say we schedule the tasks from smallest to largest. Then, based on Algorithm 2, we will alternate between the two users until we schedule all six small tasks. There remains 4 units of resources

left on the server which would be allocated to user 1 (without loss of generality). The result would be user 1 getting 7 units of resource and user 2 getting only 3. However, if the most resource demanding tasks are picked first, then the result would be both users getting 5 units of the resource which means equal global dominant share between the two users. Hence, it would be the optimal solution since it maximizes the minimum global dominant share. From this example we can see that scheduling the most resource heavy tasks can be beneficial.

After picking the most resource heavy task, we assign it to the best fit server. Recall that the fitness of assigning task n to server j is defined as $F(n, j) = ||d_n - c_j/c_j^*||$ (introduced in Table 4.1). Then in line 10 of Algorithm 2, task n's best fit server is characterized by the server who produces the smallest $F(n, j)$ value. In other words, the best fit server is the server whose remaining resource vector is the most similar to its resource demands. This ensures that CPU heavy tasks are assigned to CPU heavy servers, for example.

4.4.2 Splittable and Unordered

In this section, we will assume that all users have a set of splittable and unordered tasks. Tasks from the same user need not be on the same server since they are unordered. Tasks can be allocated an integer multiple of their required resources to decrease completion time. Splittable tasks add a new dimension to the resource allocation problem since it must be considered if a resource is better used to speed up an existing task or to begin a new task. Furthermore, since it is now possible to allocate more than the required resources to a task, we can get closer to equal global dominant shares than in the previous case. Algorithm 3 is the proposed heuristic for

the splittable and unordered case.

Algorithm 3 Splittable and Unordered

1: **for** each time step **do**
2: new users arrive
3: available tasks ← all unassigned tasks of every user
4: **while** there are available tasks **do**
5: current_user i ← user with the smallest global dominant share $G_i(A_i)$
6: current_task n ← available task of current_user i with the largest resource demands $\sum d_n$
7: **if** no server can fit current_task nj **then**
8: remove current_task n from available tasks
9: **else**
10: assign current_task n to best fit server
11: allocate kD_j resources where k minimizes $Variance_i(G_i(A_i))$ and $k < l_j$
12: update remaining server resources and available tasks
13: **end if**
14: **end while**
15: **end for**

Due to the similarities of the two cases, Algorithm 3 works similarly to Algorithm 2. Once again, at each time step, the system repeatedly picks the user with the lowest global dominant share (line 5) and schedules its most resource demanding task (line 6) until there are no remaining tasks left or there are no more resources available. The best fit server mentioned in line 10 is the same characterization as in Algorithm 2. To reiterate, the fitness of assigning task n to server j is defined by $F(n, j) = ||d_n - c_j/c_j^*||$.

The reasoning for this design is the same as in the previous case. The key difference is how to determine what integer multiple of the required resources to allocate to each scheduled task.

Recall that the DRF scheme's objective is to maximize the minimum global dominant share across all users, however, it is difficult to maximize this in each step of the

allocation process since the lowest global dominant share user just needs to surpass another user's global dominant share to do so. However, the aim behind the maximization of the minimum global dominant share is to achieve a balanced resource share across all tasks. Therefore, the proposed algorithm allocates resources with a goal to minimize the variance of all global dominant shares (line 11). Minimizing variance is not the same as maximizing the minimum global dominant share across all users with non-divisible tasks, but it is commonly used as a substitute since the minimum global dominant share is a value that is difficult to maximize at each time step [28]. However, there is a requirement that a task j should not be allocated more than l_j times its resource requirements where l_j is its execution time.

Theorem 7. *In the unordered and splittable case, a task j should not be allocated more than l_j times its resource requirements.*

Proof. By allocating l_j times its resource requirements to task j, its execution time is reduced to 1 time step, which is the lowest that it can be in our time step model. Increasing its allocation any further becomes a wasteful use of resources. Therefore, task j must not be allocated more than l_j times its resource requirements.

4.4.3 Unsplittable and Ordered

In this section, we will assume that all users have a set of unsplittable and ordered tasks. Tasks from the same user must be on the same server.

Since we are approaching the task scheduling problem from a fair allocation perspective, this case becomes trivial. In this case, it is meaningless to assign more than the required resources to a user since the tasks are unsplittable. Moreover, since each user has a task chain, a user will only need to run one unsplittable task at any given

point. As such, there is no decision to be made from a fairness perspective as each user only requires a fixed amount of resources at any given moment and allocating more will not be beneficial to the user at all. Therefore, we will not propose a heuristic for this case.

4.4.4 Splittable and Ordered

In this section, we will assume that all users have a set of splittable and ordered tasks. Tasks from the same user must be on the same server and a task chain cannot be interrupted. Tasks can be allocated an integer multiple of their required resources to decrease completion time. This case is rather complex since a task chain can include tasks with wildly different resource demands, which makes it difficult to allocate without being wasteful. Furthermore, allocation can be changed during a task chain based on the task currently being executed. Overall, this means a lot more decisions will need to be made in order to achieve the most fair allocation. Algorithm 4 describes the proposed algorithm. Since, in this case, each user has exactly one task chain, we will use the terms "user" and "task chain" interchangeably. That is, user i and task chain i refer to the same thing.

At each time step, if new users arrive, Algorithm 4 will first compute every new user's size characteristic, which is defined as follows. For user i, recall that their size characteristic is defined by z_i (line 7). This is the vector representing the largest demand of each resource in the entire task chain. For example, the largest demand of each resource in a task chain with resource demands $(1, 5), (5, 1), (6, 2)$ is $(6, 5)$. Then, the size characteristic z_i for the user with this task chain would be $(6, 5)$. In other words, if user i is allocated z_i resources, then it can execute its task chain in

Algorithm 4 Splittable and Ordered

1: **for** each time step **do**
2: new users arrive
3: **if** new users is empty **then**
4: pass
5: **end if**
6: **for** user $i \in$ new users **do**
7: compute size characteristic z_i
8: **end for**
9: **for** user $i \in$ existing users **do**
10: reduce A_i to z_i
11: **end for**
12: available users $\leftarrow$ all users
13: **while** available users is non empty **and** server resources are non empty **do**
14: current_user $i \leftarrow$ user with the smallest global dominant share $G_i(A_i)$ tie break by largest size characteristic z_i
15: **if** no server can fit current_user i **then**
16: remove current_user i from available users
17: **else**
18: assign current_user i to best fit server
19: allocate kz_i resources where k minimizes $Variance_i(G_i(A_i))$ with $k < max_{j \in i} l_j$
20: **end if**
21: update server resources and available users
22: **end while**
23: **end for**

its entirety. Then, we reduce all users currently executing task chains down to its lowest required amount of resources which is z_i for user i (line 10). This is to allow room for new users to be allocated resources but at the same time not to interrupt any existing task chain. Then, the algorithm will select the user with the smallest global dominant share, however, since all new users have global dominant share 0, this selection will have a tie break by largest size characteristic z_i (line 14). This is because the largest size characteristic typically denotes the most resource demanding users. We select the most resource demanding users to allocate first due to it being

easier to allocate large task chains when there is more resource available. Then, we select the best fit server based on the size characteristic and assign the task chain to this server until its completion (line 18). Recall that the best fit server for a given task chain is characterized differently than for a single task. The fitness of server j for task chain i is defined as $F'(i,j) = ||z_i/z_i^* - c_j^*/c_{j1}^*||$ where c_j is the vector representing the remaining resources of server j and z_i is the size characteristic of task chain i (introduced in Table 4.1). Then, the proposed algorithm allocates an integer multiple amount of resources such that to minimize the variance of all global dominant shares (line 19). However, there is the requirement that a user i should not be allocated more than $max_{n\in i}l_j$ times its size characteristic z_i. Repeat for next time step.

Theorem 8. *In the splittable and ordered case, a task chain i should not be allocated more than $max_{j\in i}l_j$ times its size characteristic z_i.*

Proof. Consider allocating $max_{n\in i}l_j$ times its size characteristic z_i amount resources to task chain i. Then, for each task n in task chain i, it will be executed on at $max_{n\in i}l_j$ times z_i amount of resources. By definition, $max_{n\in i}l_j \geq l_j$ and $z_j \geq D_j$. Hence, each task n in task chain i has its execution time reduced to less than or equal to 1 time step which is the minimum it can be. Therefore, it is wasteful to allocate more resources to this task chain. Therefore, a task chain i should not be allocated more than $max_{n\in i}l_j$ times its size characteristic z_i.

4.5 Algorithm Analysis

A key characteristic is that the proposed algorithms must be very low complexity. This is due to the highly dynamic nature of online scheduling in fog computing.

Existing cloud clusters have server populations in the tens of thousands with even more users and task requests. As an augmentation to the cloud computing paradigm, fog computing is expected to be implemented at the same scale. Furthermore, these algorithms must be run at every time step, therefore the scheduling algorithms must be highly efficient. Following, we will prove the termination and complexity for each of the proposed algorithms.

Theorem 9. *Algorithm 2 terminates and has complexity $O(n^2)$ where n is the number of available tasks.*

Proof. Suppose there are n available tasks and m servers at each time step. Then, we will need to first find the user with the smallest global dominant share which takes $O(logn)$ time in the worst case. Then, we must select this user's task with the largest resource demands which takes $O(logn)$ time in the worst case. Then, we must find the best fit server for the selected task which takes m time since we must compute the best fit function for each server. In the worst case, we must do this for all n tasks. Then, the complexity of this algorithm is $O(n(m+logn))$. Without loss of generality, assume that $n > m$, then $O(n(m + logn)) = O(n^2 + nlogn) = O(n^2)$. Therefore, Algorithm 2 is $O(n^2)$.

Theorem 10. *Algorithm 3 terminates and has complexity $O(n^2)$ where n is the number of tasks.*

Proof. Suppose there are n tasks and m servers at each time step. Algorithm 3 performs similarly to Algorithm 2. The key difference is needing to find the variance of all users' global dominant share after the selection of a task. Computation of variance takes $O(n)$ time in the worst case [46]. Then, the complexity of this algorithm is

$O(n(m+n))$. Without loss of generality, assume that $n > m$, then $O(n(m+n)) = O(n^2+n^2) = O(n^2)$. Therefore, Algorithm 3 is $O(n^2)$.

Theorem 11. *Algorithm 4 terminates and has complexity $O(n^2)$ where n is the number of tasks.*

Proof. Suppose there are n users and m servers at each time step. Algorithm 4 will compute the size characteristic z_i for all new users and reduce allocations A_i for all existing users which takes $O(n)$ time. Then, we will need to find the user with the smallest global dominant share tie broken by largest size characteristic which takes $O(logn)$ time in the worst case. Then, we must find the best fit server for the selected user which takes m time since we must compute the best fit function for each server. Then, variance of the global dominant shares is calculated which takes $O(n)$ time [46]. In the worst case, we must do this for all n users. Then, the complexity of this algorithm is $O(n(m+logn+n+n))$. Without loss of generality, assume that $n > m$, then $O(n(m+logn+n+n)) = O(nm+nlogn+n^2+n^2) = O(n^2)$. Therefore, Algorithm 4 is $O(n^2)$.

4.6 Simulation Results

The simulation task data comes from usage traces of Google clusters [47]. These traces provide the resource demand information of over 900 users on a Google cluster of over 10,000 servers. The server configurations of these servers are provided by Wang *et al.* in 2014 [3]. The tasks and servers used in our simulations are randomly extracted from the above data. Note that this data does not indicate the characteristics of each task (ordered/unordered, splittable/unsplittable), so these characteristics were assumed in the simulations. Each time, we simulate 5 users who arrive based

on a Poisson process with mean rate of 0.5 on 3 servers with 2 resource types (CPU and memory). Each user has a random number of tasks.

To evaluate the performance of the proposed algorithms, three baseline algorithms will be used: First Come First Server (FCFS), Randomized, and Shortest Job First (SJF). The first two algorithms are self-explanatory. SJF is a very efficient algorithm used to minimize waiting time for scenarios where tasks' execution time is known [48]. It works by scheduling the tasks with the shortest execution time first. In fact, it is optimal, in that for a given set of processes and their execution times it gives the least average waiting time for each process [48]. While it is not optimal in our multi-server, multi-resource, online environment, it still provides a good benchmark. Since we have multiple servers, SJF will decide on which server to assign each task to using the same best fit characterization used in the proposed algorithms. Furthermore, to evaluate Algorithm 3 and Algorithm 4, splittable versions of all three baseline algorithms are used. In these versions, each task is allocated a random integer multiple of its required resources. The splittable versions of the baseline algorithms work the same otherwise. Each experiment was run 20 times for more generalized results since the input data are randomly selected.

4.6.1 Algorithm 2 and Algorithm 3

We will first evaluate the performance of Algorithm 2 and Algorithm 3. Algorithm 3 and Algorithm 2 are discussed together because they are very similar in design. Additionally, it allows us to observe the effect of having splittable tasks in the system.

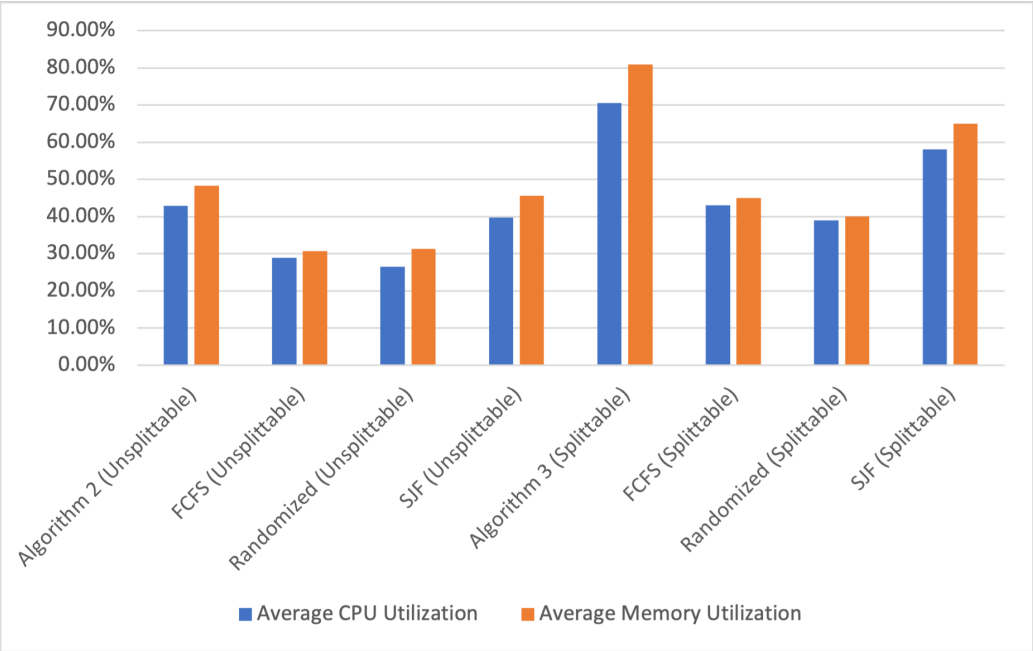

Figure 4.3: Average Resource Utilization (Unsplittable/Unordered and Splittable/Unordered)

Resource Utilization

Our first evaluation focuses on how well the algorithms utilize the available resources in the scheduling system. Fig. 4.3 shows the average resource utilization percentage of all simulated algorithms. We can see that Algorithm 2 slightly outperforms SJF (unsplittable) and severely outperforms FCFS (unsplittable) and Randomized (unsplittable). Higher resource utilization correlates with faster completion times and Fig. 4.4 reflects this. Algorithm 2 completes tasks at a similar speed as SJF (unsplittable) and much faster than FCFS (unsplittable) and Randomized (unsplittable). Since Algorithm 2 is designed such that it maximizes the minimum global dominant share across all users, performing similar to SJF (unsplittable) which focuses on task completion speed can be considered a success.

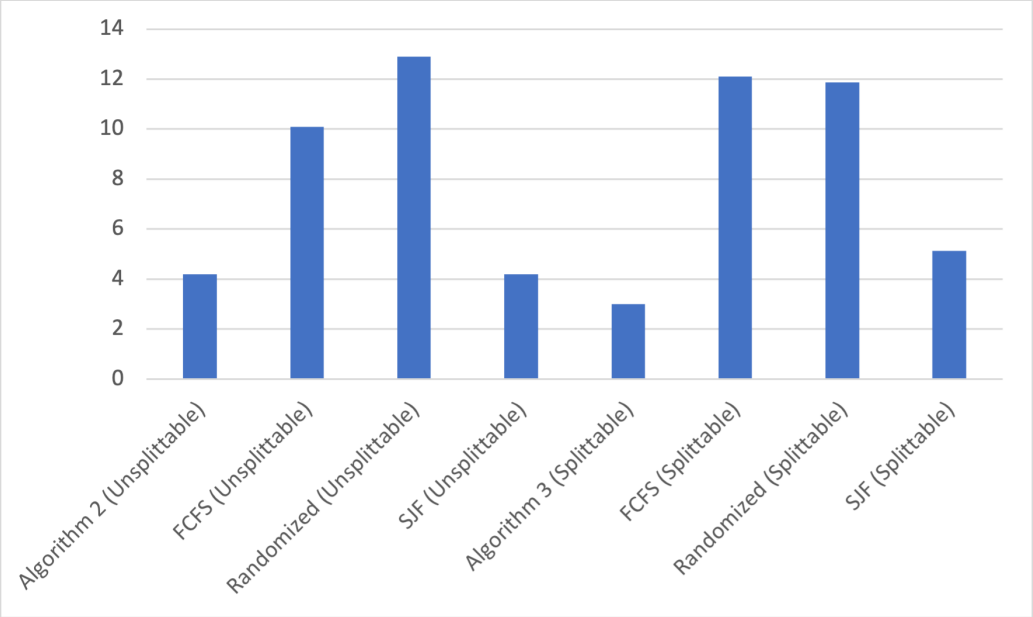

Figure 4.4: Average Completion Time (Unsplittable/Unordered and Splittable/Unordered)

In the splittable case, a similar result is shown. Algorithm 3 performs the best in terms of resource allocation followed by SJF (Splittable) and then by Randomized (Splittable) and FCFS (Splittable). This is mirrored in Fig. 4.4.

It is expected that Algorithm 3 outperforms all other algorithms, both splittable and unsplittable in terms of resource utilization. This is due to splittable tasks being able use more resources since the system can fill gaps of unused resources by allocating more to a splittable task, and consequently these splittable tasks will complete faster resulting in better performance in completion speeds as well. While Algorithm 2 performs similarly to SJF (Unsplittable), Algorithm 3 outperforms SJF (Splittable) by a rather significant margin.

In summary, the proposed algorithms performed similarly or better than their counterparts in terms of both resource utilization and task completion times.

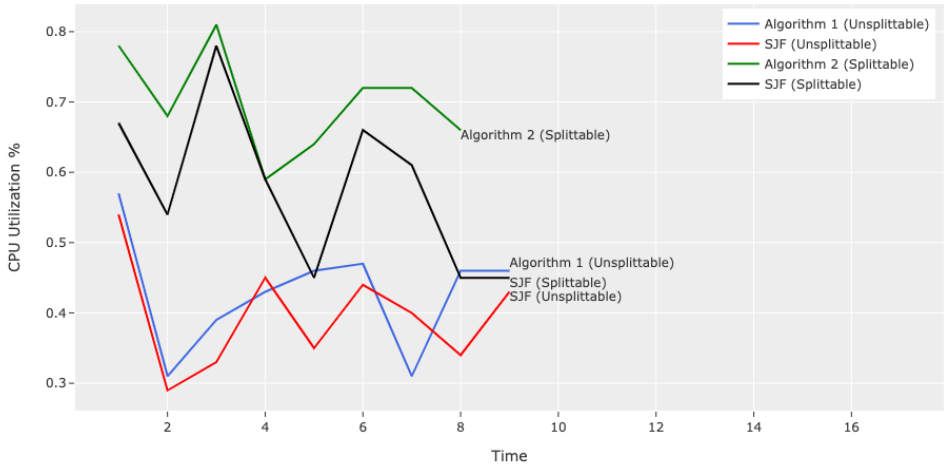

Figure 4.5: CPU Utilization Over Time (Unsplittable/Unordered and Splittable/Unordered)

Fig. 4.5 and Fig. 4.6 shows the resource utilization over the duration of one sample simulation. In these figures, we can observe the changes in resource utilization over each individual time step. The FCFS and Randomized algorithms are omitted so as to not clutter the figures. We can see that while Algorithm 2 and Algorithm 3 perform better than their SJF counterparts on average, they may perform worse during specific time steps. This is due to the nature of online environments where users arrive continuously. Then, since Algorithm 2 and Algorithm 3 complete tasks faster than their SJF counterparts, they may have more downtime between the arrival of users. This would explain why the SJF algorithms have higher resource utilization at specific time steps but lower completion times overall.

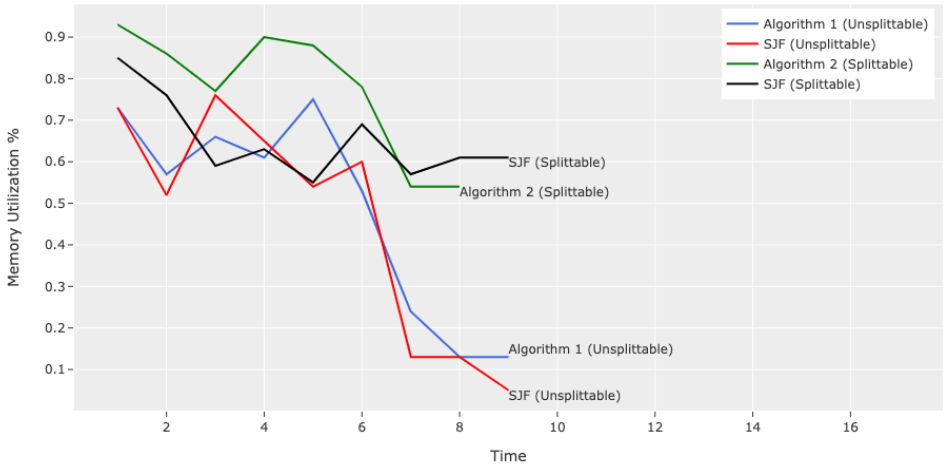

Figure 4.6: Memory Utilization Over Time (Unsplittable/Unordered and Splittable/Unordered)

Fairness

The second evaluation will focus on the main objective of this chapter, fairness between users. Recall that the DRF scheme aims to maximize the minimum global dominant share across all users. Then, to evaluate this objective, we use two metrics: average variance of all global dominant shares and average minimum dominant share. Fig. 4.7 and Fig. 4.8 summarize our results. Fig. 4.7 displays the variance of the global dominant shares of all users at every time step averaged throughout the simulations. Lower variance typically correlates to higher minimum global dominant share. Fig. 4.8 displays the minimum global dominant share at every time step averaged throughout the simulations. As per the DRF scheme, a higher minimum global dominant share means a fairer allocation. Between the two proposed algorithms, Algorithm 3 performs better than the nonsplittable case in terms of the two metrics. This is due to the splittability of its tasks allows it to have more freedom when

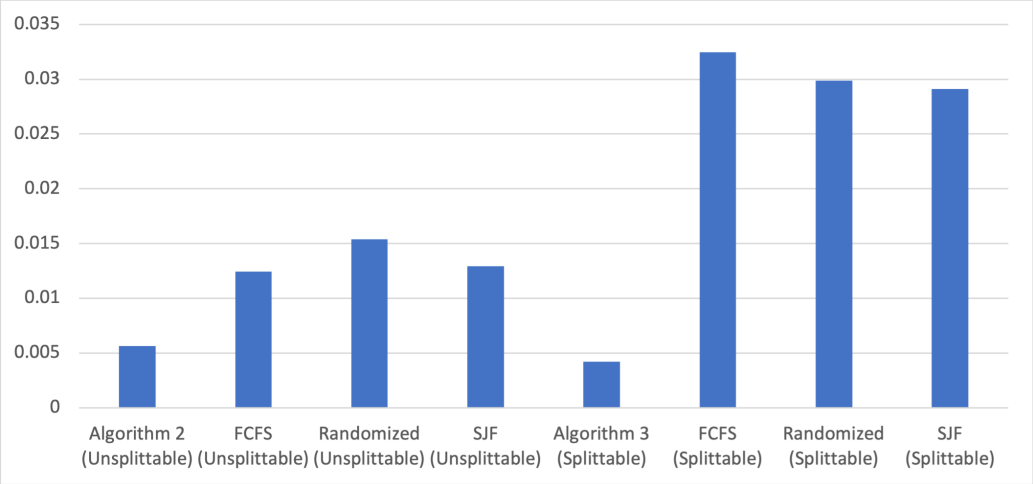

Figure 4.7: Average Dominant Share Variance (Unsplittable/Unordered and Splittable/Unordered)

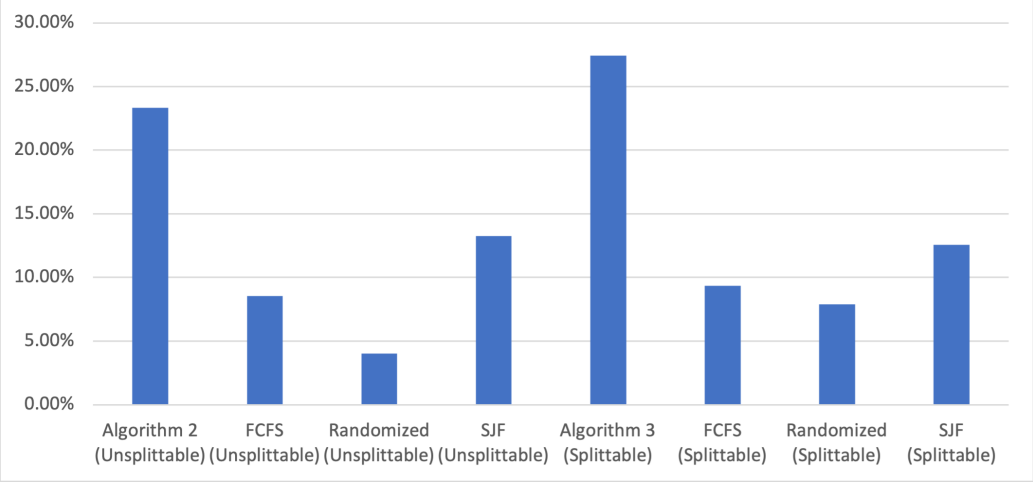

Figure 4.8: Average Minimum Dominant Share (Unsplittable/Unordered and Splittable/Unordered)

allocating resources to users. It also performs significantly better than Splittable. We can see the two figures show similar results where Algorithm 2 and Algorithm 3 have better results than the other algorithms. This is to be expected as the other algorithms do not consider global dominant shares at all.

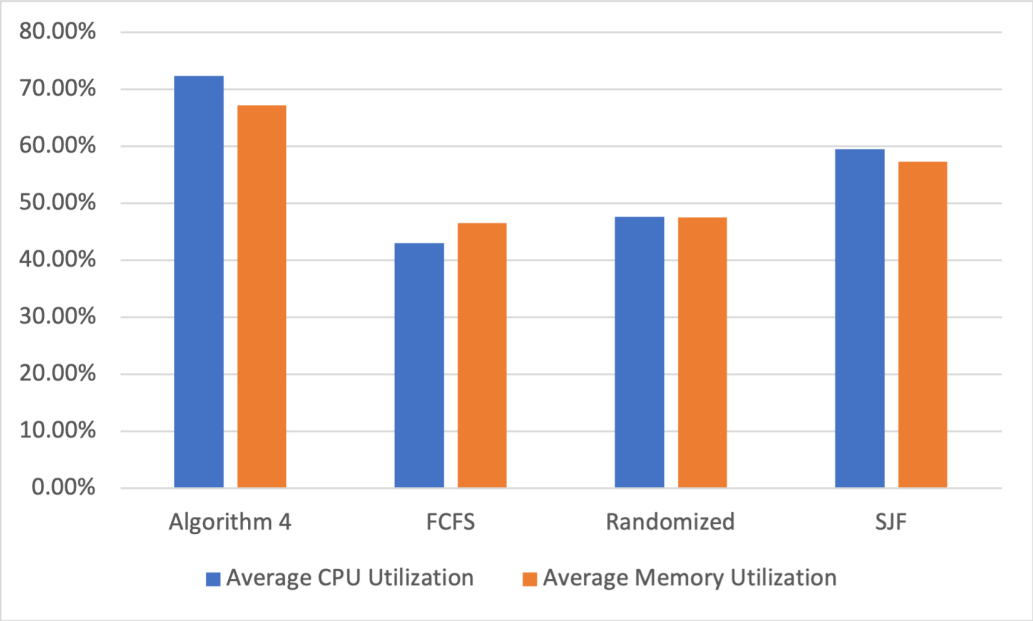

Figure 4.9: Average Resource Utilization (Splittable/Ordered)

4.6.2 Algorithm 4

To evaluate the performance of the Algorithm 4, the same 3 baseline algorithms will be used: FCFS, Randomized, and SJF. However, since none of the three algorithms are meant for splittable tasks, they will allocate a random integer multiple of the required amount of resources per task chain. They will also pick servers using the best fit characterization. The simulations were run for 20 times for more generalized results since the input data are randomly selected.

Resource Utilization

Our first evaluation, once again, focuses on how well the algorithms utilize the available resources in the scheduling system. Fig. 4.9 shows the average resource utilization percentage of all simulated algorithms. Results show that the proposed

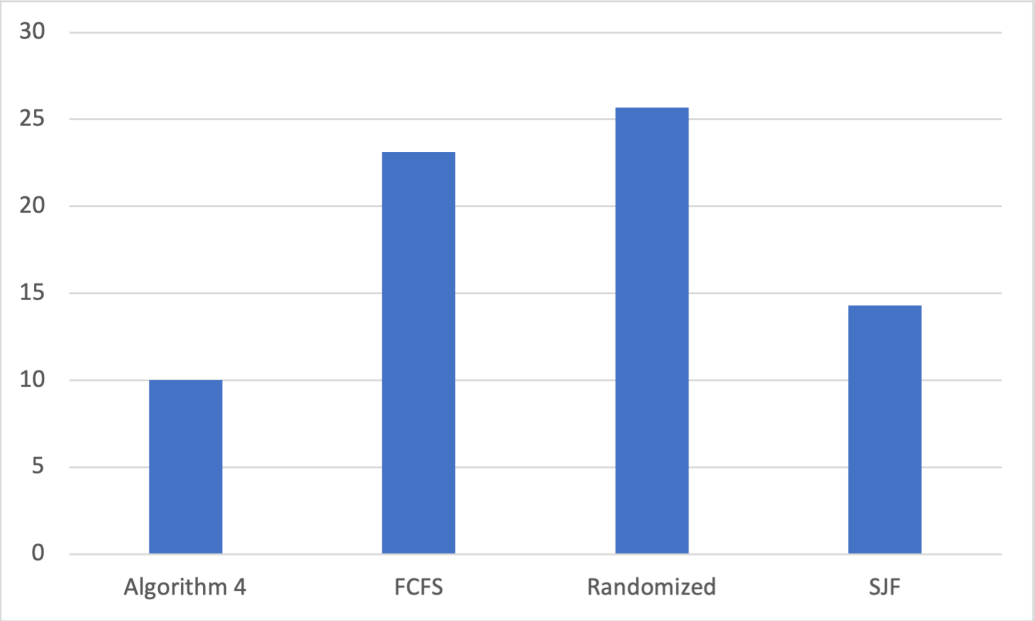

Figure 4.10: Average Completion Time (Splittable/Ordered)

Algorithm 4 outperforms all of the other algorithms by a significant margin. Consequently, Fig. 4.10 shows a similar result in completion times. In both metrics, SJF performs better than FCFS and Randomized but not as good as Algorithm 4.

Fig. 4.11 and Fig. 4.12 shows the resource utilization over the duration of one sample simulation. In these graphs, once again, we see that the baseline algorithms have higher resource utilization at specific time steps compared to the proposed Algorithm 4. Algorithm 4 executes users' task faster and the users will leave the system faster which results in downtime between the arrival of users where the available resources aren't being used.

Fairness

The second evaluation will focus on fairness between users. To evaluate this objective, we will once again use two metrics: average variance of all global dominant

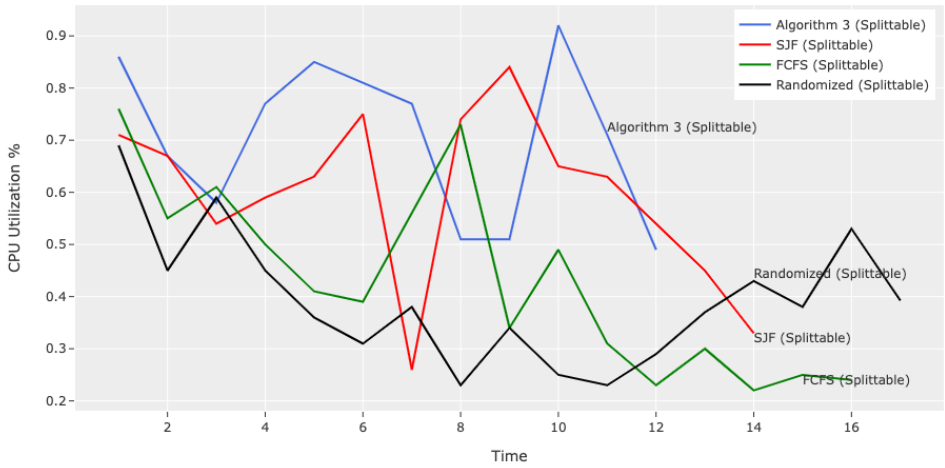

Figure 4.11: CPU Utilization Over Time (Splittable/Ordered)

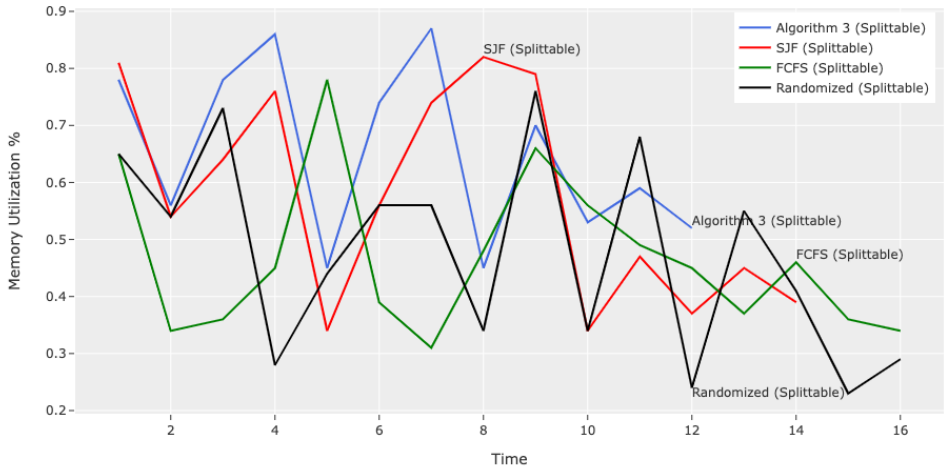

Figure 4.12: Memory Utilization Over Time (Splittable/Ordered)

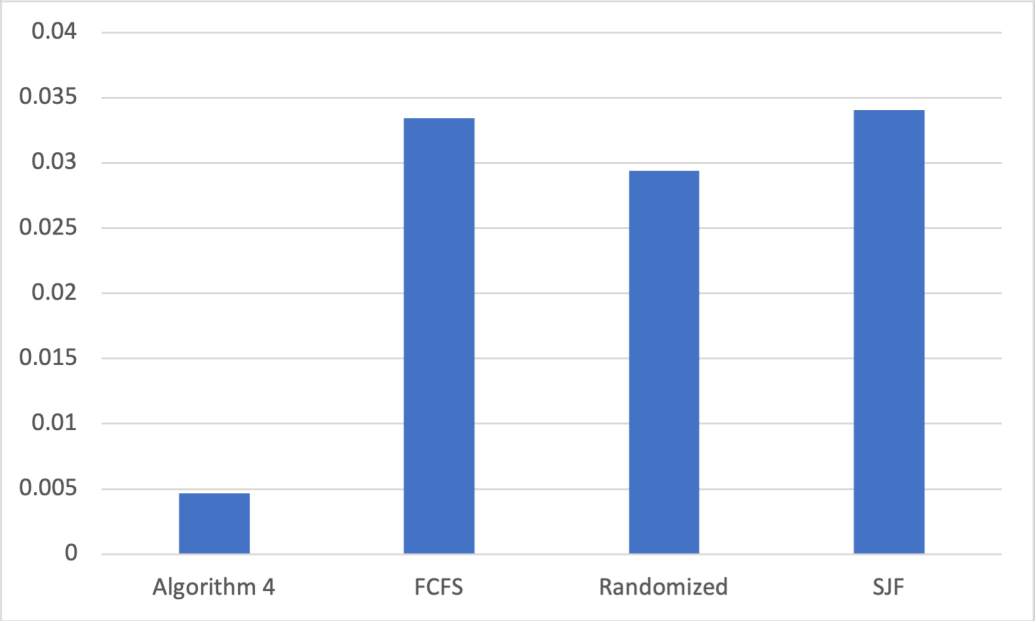

Figure 4.13: Average Dominant Share Variance (Splittable/Ordered)

shares, and average minimum dominant share. Fig. 4.13 and Fig. 4.14 summarize our results. Fig. 4.7 displays the variance of the global dominant shares of all users at every time step averaged throughout the simulations. Fig. 4.8 displays the minimum global dominant share at every time step averaged throughout the simulations. We see that Algorithm 4 performs the best out of all simulated algorithms. However, Algorithm 4 has extremely low global dominant share variance but this does not translate to average minimum dominant share as Algorithm 4 only slightly beats out SJF in average minimum dominant share. This is due to the nature of the ordered environment. Since each user's tasks are ordered, they cannot be completed in parallel and will hence stay in the system for longer. Then, there will be more users executing tasks in the system at any given time.

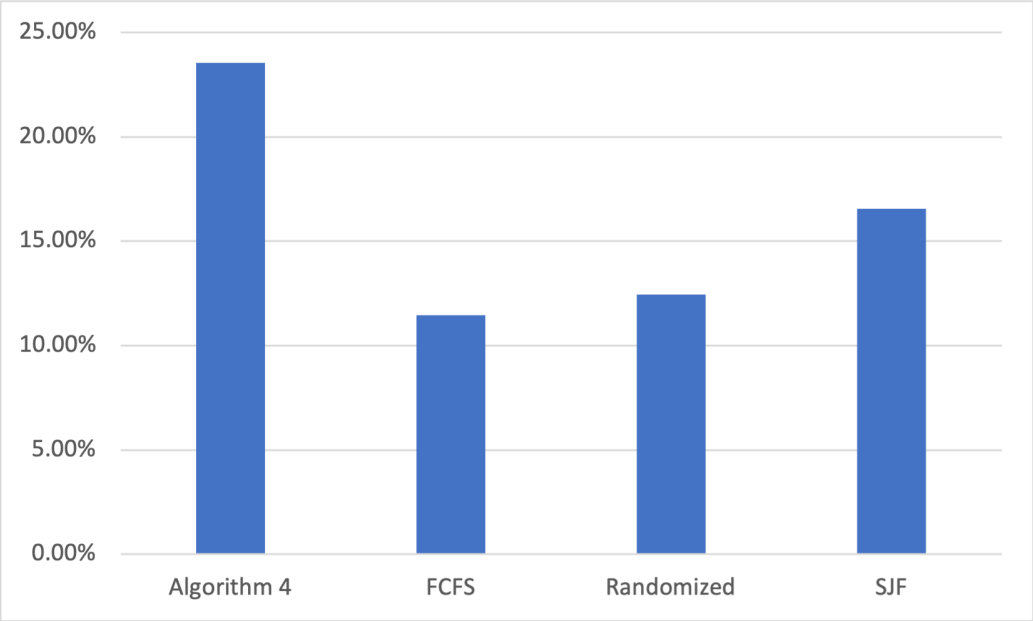

Figure 4.14: Average Minimum Dominant Share (Splittable/Ordered)

4.7 Conclusion

This chapter examines the online resource allocation problem within fog computing under an environment that is multi-server, multi-resource, and includes heterogeneous tasks. This research problem is approached from a fairness perspective where the DRF scheme is used to formulate a fairness maximizing optimization problem. Four different types of tasks are considered: ordered/unordered, splittable/unsplittable. Three low complexity heuristics are proposed and are evaluated against three baseline algorithms. Results show that the proposed algorithms perform similar to better in terms of task completion speed but significantly better in terms of user fairness.

Chapter 5

Conclusion

In this thesis, we examined two different scheduling problems under two emerging computing paradigms, Multi-access Edge Computing (MEC) and fog computing. We approached these problems using low complexity heuristics to match the dynamic nature of the MEC and fog environments.

The first problem studied in Chapter 3 was the task assignment problem in Parked Vehicular Computing (PVC) where we formulated the problem as a weighted multi-objective optimization problem that aimed to minimize both task delay and wireless channel load. By minimizing task delay, we hope to reduce the task completion times. By minimizing the wireless channel load, we hope to maintain stable and high-quality wireless communication between the participating parties. To solve this problem, we proposed a stable matching based heuristic where the respective preference rankings are based on the two objectives.

The second problem studied in Chapter 4 was the resource allocation problem in fog computing where we approached the problem from a fairness perspective.

We examined an environment that is multi-server, multi-resource, and includes heterogeneous tasks. Furthermore, four different types of tasks were considered: ordered/unordered, splittable/unsplittable. We proposed three low complexity heuristics that aim to maximize the minimum global dominant share across all users as per the Dominant Resource Fairness (DRF) scheme.

We conclude this thesis by putting forward some details on current limitations and future research directions.

For the task assignment problem in PVC, formulation of the optimization problem is a key area. Currently, only the total task completion speed is measured in terms of delay as it is the simplest measure of delay. However, practically speaking, there may be other requirements of task completion such as deadlines. Then, a different and more general task completion metric would have to be used as the objective. Furthermore, the transmission delay is also an important area to look at. In this thesis, we only minimized the number of vehicles in use as an attempt to maintain communication quality and decrease transmission delay. However, there could be other approaches to minimizing transmission delay which could be implemented in future algorithms. Secondly, only a general formulation of an incentive mechanism is proposed in this thesis. Future work could include a formal formulation of such a mechanism and evaluating the ability of the incentive mechanism to provide accurate remaining parking time estimates through various simulations. This should then be compared against the effectiveness of parking time estimation via statistical modelling. Another extension would be designing a hybrid solution using both statistical analysis and incentivized user submissions.

For the fair resource allocation problem in fog computing, a key limitation in the

algorithms proposed is that they are not very generalizable. Realistically, not all requested tasks will be the same in terms of ordered/unordered and splittable/unsplittable. In the future, we should work towards an algorithm that is able to handle an environment where there are tasks with a variety of characteristics. To do so, machine learning techniques such as deep reinforcement learning would likely have to be used.

However, currently, machine learning is used to produce highly specific models that are only equipped to handle some very specific instances of an optimization problem. We are still quite far from being able to produce a generalizable machine learning model that can be used to solve several, or perhaps an entire class of, optimization problems. Nevertheless, it is without a doubt that machine learning is emerging as a promising approach to solving optimization problems such as the two examined in this thesis.

Bibliography

[1] S. Raza, S. Wang, M. Ahmed, and M. R. Anwar, "A survey on vehicular edge computing: Architecture, applications, technical issues, and future directions," *Wirel. Commun. Mob. Comput.*, vol. 2019, pp. 3 159 762:1–3 159 762:19, 2019.

[2] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *Nsdi*, vol. 11, no. 2011, 2011, pp. 24–24.

[3] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 583–591.

[4] H. Guo, J. Liu, J. Zhang, W. Sun, and N. Kato, "Mobile-edge computation offloading for ultradense iot networks," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4977–4988, 2018.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[6] A. Markus and A. Kertesz, "A survey and taxonomy of simulation environments modelling fog computing," *Simulation Modelling Practice and Theory*, vol. 101, p. 102042, 2020.

[7] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi, and A. Leon-Garcia, "Fog computing: a comprehensive architectural survey," *IEEE Access*, vol. 8, pp. 69 105–69 133, 2020.

[8] A. B. De Souza, P. A. L. Rego, T. Carneiro, J. D. C. Rodrigues, P. P. R. Filho, J. N. De Souza, V. Chamola, V. H. C. De Albuquerque, and B. Sikdar, "Computation offloading for vehicular environments: A survey," *IEEE Access*, vol. 8, pp. 198 214–198 243, 2020.

[9] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Netw Appl*, 2020.

[10] P. Hosseinioun, M. Kheirabadi, S. R. Kamel Tabbakh, and R. Ghaemi, "atask scheduling approaches in fog computing: a survey," *Transactions on Emerging Telecommunications Technologies*, p. e3792, 2020.

[11] B. Guenin, J. Könemann, and L. Tuncel, *A gentle introduction to optimization*. Cambridge University Press, 2014.

[12] L. Tom and V. Bindu, "Task scheduling algorithms in cloud computing: A survey," in *International Conference on Inventive Computation Technologies*. Springer, 2019, pp. 342–350.

[13] M. Pinedo, *Scheduling*. Springer, 2012, vol. 29.

[14] G. Optimization, "Inc.,"gurobi optimizer reference manual," 2015," 2014.

[15] R. Lima and E. Seminar, "Ibm ilog cplex-what is inside of the box," in *Proc. 2010 EWO Seminar*, 2010, pp. 1–72.

[16] A. S. Minkoff, "A systematic approach to osl application programming," *IBM systems journal*, vol. 31, no. 1, pp. 49–61, 1992.

[17] P. Salot, "A survey of various scheduling algorithm in cloud computing environment," *International Journal of Research in Engineering and Technology*, vol. 2, no. 2, pp. 131–135, 2013.

[18] A. IMANE, "Mobile edge computing for the internet of things." 2019.

[19] X. Yang, Z. Chen, K. Li, Y. Sun, N. Liu, W. Xie, and Y. Zhao, "Communication-constrained mobile edge computing systems for wireless virtual reality: Scheduling and tradeoff," *IEEE Access*, vol. 6, pp. 16 665–16 677, 2018.

[20] S. Arif, S. Olariu, J. Wang, G. Yan, W. Yang, and I. Khalil, "Datacenter at the airport: Reasoning about time-dependent parking lot occupancy," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 11, pp. 2067–2080, 2012.

[21] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[22] A. E. Roth, "Deferred acceptance algorithms: History, theory, practice, and open questions," *international Journal of game Theory*, vol. 36, no. 3, pp. 537–569, 2008.

[23] Y. Liu, S. Wang, J. Huang, and F. Yang, "A computation offloading algorithm based on game theory for vehicular edge networks," in *2018 IEEE International*

Conference on Communications, ICC 2018, Kansas City, MO, USA, May 20-24, 2018. IEEE, 2018, pp. 1–6.

[24] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," *IEEE access*, vol. 6, pp. 47 980–48 009, 2018.

[25] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE access*, vol. 6, pp. 6900–6919, 2017.

[26] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[27] S. M. Parikh, "A survey on cloud computing resource allocation techniques," in *2013 Nirma University International Conference on Engineering (NUiCONE)*. IEEE, 2013, pp. 1–5.

[28] S. Bian, X. Huang, and Z. Shao, "Online task scheduling for fog computing with multi-resource fairness," in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*. IEEE, 2019, pp. 1–5.

[29] K. Zhang, Y. Mao, S. Leng, S. Maharjan, A. Vinel, and Y. Zhang, "Contract-theoretic approach for delay constrained offloading in vehicular edge computing networks," *Mob. Networks Appl.*, vol. 24, no. 3, pp. 1003–1014, 2019.

[30] M. Whaiduzzaman, M. Sookhak, A. Gani, and R. Buyya, "A survey on vehicular cloud computing," *Journal of Network and Computer Applications*, vol. 40, pp. 325–344, 2014.

[31] X. Huang, R. Yu, J. Liu, and L. Shu, "Parked vehicle edge computing: Exploiting opportunistic resources for distributed mobile applications," *IEEE Access*, vol. 6, pp. 66 649–66 663, 2018.

[32] S. Ge, M. Cheng, X. He, and X. Zhou, "A two-stage service migration algorithm in parked vehicle edge computing for internet of things," *Sensors*, vol. 20, no. 10, 2020.

[33] D. Wang, R. R. Sattiraju, A. Weinand, and H. D. Schotten, "System-level simulator of LTE sidelink C-V2X communication for 5g," *CoRR*, vol. abs/1904.07962, 2019.

[34] L. Ji, A. Weinand, B. Han, and H. D. Schotten, "Multi-rats support to improve V2X communication," in *2018 IEEE Wireless Communications and Networking Conference, WCNC 2018, Barcelona, Spain, April 15-18, 2018*. IEEE, 2018, pp. 1–6.

[35] T. H. Thi Le, N. H. Tran, Y. K. Tun, O. Tran Thi Kim, K. Kim, and C. S. Hong, "Sharing incentive mechanism, task assignment and resource allocation for task offloading in vehicular mobile edge computing," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–8.

[36] B. Korte and J. Vygen, *Bin-Packing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 471–488.

[37] I. Sabuncuoglu and M. Bayız, "Analysis of reactive scheduling problems in a job shop environment," *European Journal of operational research*, vol. 126, no. 3, pp. 567–586, 2000.

[38] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of scheduling*, vol. 12, no. 4, pp. 417–431, 2009.

[39] Z. Wang, J. Zhang, and S. Yang, "An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals," *Swarm and Evolutionary Computation*, vol. 51, p. 100594, 2019.

[40] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 261–276.

[41] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European conference on Computer systems*, 2010, pp. 265–278.

[42] G. Zhang, F. Shen, Y. Yang, H. Qian, and W. Yao, "Fair task offloading among fog nodes in fog computing networks," in *2018 IEEE international conference on communications (ICC)*. IEEE, 2018, pp. 1–6.

[43] M. Mukherjee, M. Guo, J. Lloret, R. Iqbal, and Q. Zhang, "Deadline-aware fair scheduling for offloaded tasks in fog computing with inter-fog dependency," *IEEE Communications Letters*, vol. 24, no. 2, pp. 307–311, 2019.

[44] A. Östman, "Distributed dominant resource fairness using gradient overlay," 2017.

[45] D. C. Parkes, A. D. Procaccia, and N. Shah, "Beyond dominant resource fairness: Extensions, limitations, and indivisibilities," *ACM Transactions on Economics and Computation (TEAC)*, vol. 3, no. 1, pp. 1–22, 2015.

[46] D. E. Knuth, *The art of computer programming*. Pearson Education, 1997, vol. 3.

[47] J. W. C. Reiss and J. L. Hellerstein, ""google cluster-usage traces," https://github.com/google/cluster-data.

[48] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating systems: Three easy pieces*. Arpaci-Dusseau Books LLC, 2018.

[49] X. Lin, J. Li, W. Yang, J. Wu, Z. Zong, and X. Wang, "Vehicle-to-cloudlet: Game-based computation demand response for mobile edge computing through vehicles," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019, pp. 1–6.

[50] D. Han, W. Chen, and Y. Fang, "A dynamic pricing strategy for vehicle assisted mobile edge computing systems," *IEEE Wireless Communications Letters*, vol. 8, no. 2, pp. 420–423, 2019.

[51] J. Du, F. Yu, X. Chu, J. Feng, and G. Lu, "Computation offloading and resource allocation in vehicular networks based on dual-side cost minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1079–1092, 2019.

[52] W. Zhan, H. Duan, and Q. Zhu, "Multi-user offloading and resource allocation for vehicular multi-access edge computing," in *2019 IEEE International Conferences on Ubiquitous Computing Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS)*, 2019, pp. 50–57.

[53] P. Sun, A. Boukerche, and R. W. L. Coutinho, "A novel cloudlet-dwell-time estimation method for assisting vehicular edge computing applications," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[54] C. Li, S. Wang, X. Huang, X. Li, R. Yu, and F. Zhao, "Parked vehicular computing for energy-efficient internet of vehicles: A contract theoretic approach," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6079–6088, 2019.

[55] G. Qiao, S. Leng, K. Zhang, and Y. He, "Collaborative task offloading in vehicular edge multi-access networks," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 48–54, 2018.

[56] F. Lin, X. Lü, I. You, and X. Zhou, "A novel utility based resource management scheme in vehicular social edge computing," *IEEE Access*, vol. 6, pp. 66 673–66 684, 2018.

[57] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylae-Jaeaeski, "Fog following me: Latency and quality balanced task allocation in vehicular fog computing," in *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2018, pp. 1–9.

[58] J. Sun, Q. Gu, T. Zheng, P. Dong, A. Valera, and Y. Qin, "Joint optimization of computation offloading and task scheduling in vehicular edge computing networks," *IEEE Access*, vol. PP, pp. 1–1, 01 2020.

[59] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. ZHANG, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, 2017.

[60] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377–4387, 2019.

[61] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang, "Optimal delay constrained offloading for vehicular edge computing networks," in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[62] Z. Zhou, P. Liu, Z. Chang, C. Xu, and Y. Zhang, "Energy-efficient workload offloading and power control in vehicular edge computing," in *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2018, pp. 191–196.

[63] L. Zhang, Z. Zhao, Q. Wu, H. Zhao, H. Xu, and X. Wu, "Energy-aware dynamic resource allocation in uav assisted mobile edge computing over social internet of vehicles," *IEEE Access*, vol. 6, pp. 56 700–56 715, 2018.

[64] Y. Ku, P. Chiang, and S. Dey, "Quality of service optimization for vehicular edge computing with solar-powered road side units," in *2018 27th International*

Conference on Computer Communication and Networks (ICCCN), 2018, pp. 1–10.

[65] X. Huang, P. Li, R. Yu, Y. Wu, K. Xie, and S. Xie, "Fedparking: A federated learning based parking space estimation with parked vehicle assisted edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, p. 9355–9368, Sep 2021. [Online]. Available: http://dx.doi.org/10.1109/TVT.2021.3098170

[66] [Online]. Available: https://www.data.act.gov.au/Transport/SmartParkingHistory/grth-myzs

Index