RESOURCE MANAGEMENT IN EDGE COMPUTING SYSTEMS

by

TAYEBEH BAHREINI

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2021

MAJOR: COMPUTER SCIENCE

Approved By:

_____

Advisor Date

_____

_____

_____

_____

DEDICATION

*To **my dad** who taught me how to stay humble while thinking big, even after his passing.*

*To my **mom**, who showed me how to stay strong in difficulties.*

*To my **husband**, who has always supported me.*

*To my **daughter**, who has brought hope and joy to my life.*

ACKNOWLEDGMENTS

I would like to express my special thanks to my advisor, Dr. Daniel Grosu, for all his continuous help, guidance, and encouragement. He patiently supported me in all stages throughout my PhD studies and constantly motivated me to diligently move toward my goals. I would like to also thank Dr. Brocanelli for his support and contributions in this research. My sincere thanks also go to the rest of my dissertation committee: Dr. Mashayekhy, Dr. Schwiebert, and Dr. Shi for their helpful comments and encouragement.

This research was supported in part by the US National Science Foundation under grant no. IIS-1724227.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Introduction

Mobile devices have become the primary computing platforms for many applications, mainly because of their availability and convenience. Many mobile applications require heavy processing while mobile devices have limited computational resources and storage capacity. Furthermore, some mobile applications such as video streaming, speech recognition, and navigation consume a large amount of energy and reduce the battery life of mobile devices. In the last decade, computing technology has received a significant attention as an effective solution to the limitations of mobile devices. The main idea of these technologies is to offload the computation-intensive applications from mobile devices to the resourceful remote servers. To make the new technologies more efficient, new resource management and computation offloading methods have been designed. In this chapter, we review the current mobile computing platforms and their applications.

1.2 Mobile cloud computing

The limitations of mobile devices can be addressed by running applications remotely on a static infrastructure that does not suffer from these limitations. Mobile Cloud Computing (MCC) has been introduced to allow mobile applications to perform their computation on cloud servers. The mobile cloud computing forum defines MCC as follows [1]: "*Mobile cloud computing at its simplest, refers to an infrastructure where both the data storage and data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and MC to not just smartphone users but a much broader range of mobile subscribers*". Figure 1.1 shows a high level structure of MCC. In this figure, mobile devices are connected to the mobile networks via base stations (e.g., base transceiver station, access point, or satellite). Requests of mobile devices are sent to the central processors that are connected to the servers which provide mobile network services. Then, requests are sent

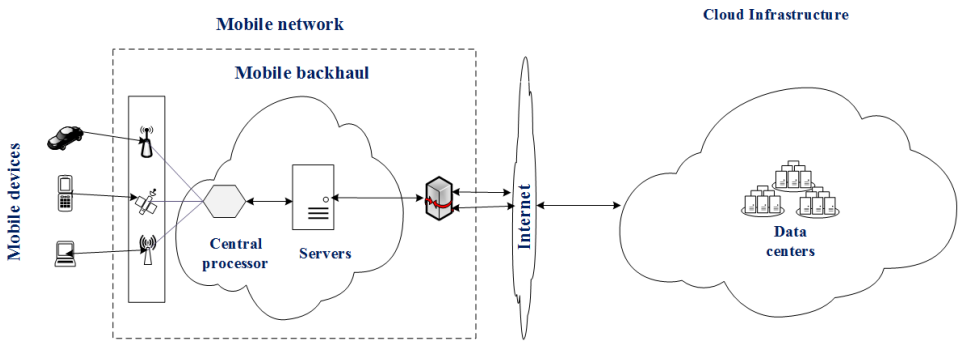


Figure 1.1: A schematic view of mobile cloud computing architecture

to the cloud via the Internet. In the cloud, controllers process requests to allocate cloud services to mobile devices [2].

MCC brings about many advantages for mobile users. It extends the battery lifetime of mobile devices by offloading computation to cloud servers. This reduces the execution time of applications which results in large amount of power consumption. It also improves data storage capacity and processing power. Moreover, it leverages the reliability of applications because the data and applications are stored and backed up on servers. However in MCC, data centers that are used by cloud services are usually far from the end-users; therefore, communication between mobile devices and datacenters involves many network hops and results in high latencies. Thus, MCC cannot be efficient for some applications that need a quick response time or have a large amount of data transmission [3]. Therefore, new extensions of MCC are required for these applications. In recent years, several paradigms have been developed to solve the inefficiency of MCC by sending a portion of data/computation to the edge of the network instead of cloud data centers. In the rest of this section, some of these paradigms are reviewed.

1.3 Cloudlet

Cloudlet introduced as an extension of MCC with the aim of "bringing the cloud closer", was proposed by Satyanarayanan et al. [4]. A Cloudlet, also known as mobile micro-cloud [5], is a cluster of multi-core computers with high internal connectivity that is

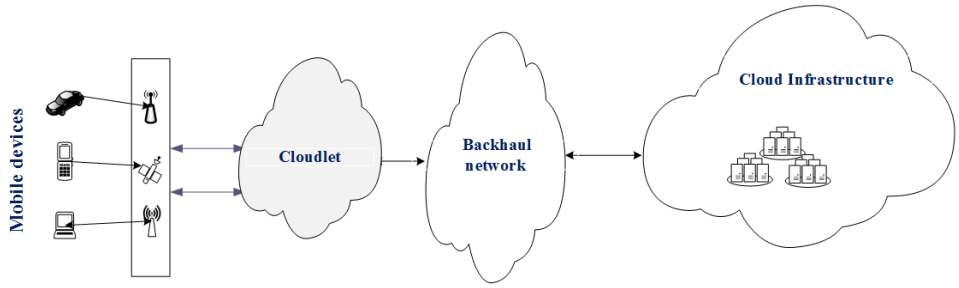


Figure 1.2: The Cloudlet architecture

available for nearby mobile devices.

Figure 1.2 illustrates the Cloudlet architecture. This architecture consists of two main levels. In the first level, there exists a traditional cloud infrastructure and cloud data centers. The second level consists of heterogeneous elements that are called Cloudlets. Each Cloudlet can be seen as a "second-class data center" which brings a better performance to mobile applications, especially for real time mobile applications. Experimental research indicates that using Cloudlets for computation offloading for wearable cognitive-assistance systems improves the response times by 80 to 200 ms and reduces the energy consumption by 30% to 40% [6].

1.4 Fog computing

Fog Computing was introduced by Bonomi et al. [7] in 2012. Fog computing enables computation at fog nodes of a network. In fog computing, infrastructures such as switches, routers, access points, and computers that can provide resource services are considered as fog nodes. Different from Cloudlets, fogs can be resource-poor or resource-rich [8]. In fact, fog computing is a scenario where a huge number of heterogeneous devices communicate and potentially cooperate to process applications and store data.

1.5 Mobile edge computing

Mobile Edge Computing (MEC) was introduced to enable processing data at the edge of the network in which an edge can be any computing resource of the network [9]. Fog computing and edge computing are very close concepts. The main difference between

them is that the fog computing mostly focuses on the infrastructure scalability while in edge computing the performance of mobile application is the first priority [10]. In contrast to Cloudlets, edge nodes are widely deployed and available to all mobile users, not just to some specific ones. Since edge nodes can be co-located with base stations, they can offer additional information such as position and mobility of users; while Cloudlets are mostly stationary computers with fast and stable internet access, offering computing, bandwidth, and storage resources to nearby mobile users; and users access them via a local area network such as Wi-Fi. Not being a part of mobile network, Cloudlets do not share network operator related knowledge [11].

The concept of edge computing dates back to the time when Akamai [12] introduced Content Delivery Networks (CDNs) to reduce the average time of loading pages. In this system, CDN nodes are placed close to the users to prefetch and cache web content. These nodes also customize some content of web pages. For example, they customize the advertisement on the web pages according to the location of users. CDNs especially improve the performance of video content, because the bandwidth savings from caching is significant [10]. Edge computing generalizes the CDN concept in which cloud computing infrastructure is leveraged by deploying local edge nodes. Figure 1.3 illustrates the MEC architecture. The main components of this architecture are as follows,

- *Mobile devices:* mobile users may request over time to run a specific application. They connect to the base stations which translate radio signals so that they can be routed through the networks.

- *Mobile network operators*: this part manages the base stations, mobile backhaul, and MEC servers. MEC servers, based on the network conditions, may process a request from mobile users directly or forward it to the remote data centers.

- *Internet backhaul:* internet backhaul contains internet infrastructure, routers, etc.

- *Service providers and cloud data centers:* after processing the request of mobile

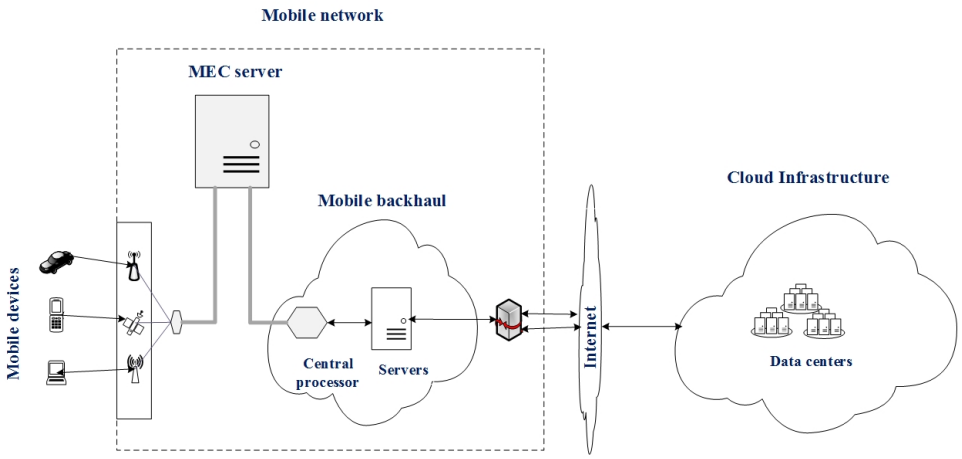


Figure 1.3: MEC architecture

users, the service providers assign the cloud services in the data centers to the mobile users.

The proximity of edge nodes helps cloud computing to improve its performance mostly in three different ways [10]:

- Reducing the response time of services: proximity of edge nodes to the mobile users reduces end-to-end latency to services that are placed on the edge of the network. This is valuable especially for real time applications that need very short response times.

- Increasing the scalability of the network: by processing the data that comes from different sources such as video cameras, sensors, and navigators, the workload of the cloud data centers is significantly decreased and only extracted data and meta data must be transferred to them.

- Enhancing fault tolerance of the cloud services: When a cloud service is not available because of network/cloud failures, or it encounters denial-of-service attack, this service can be run on a nearby edge node temporarily.

In MEC systems, since we are dealing with smaller and more distributed servers, we expected a higher operating cost for servers and more restricted capacity compared to central-

ized cloud servers. On the other hand, non-deterministic parameters such as users movement or task arrivals can significantly impact the performance of the system if they are neglected.

1.6 Vehicular edge computing

Vehicular Edge Computing (VEC) systems [13] are edge computing systems where computational nodes can be deployed in cell towers, Road-Side Units (RSUs), and within connected vehicles. In fact, VEC systems are mainly composed of connected vehicles that have three main characteristics. First, they have an on-board computing node for workload execution (e.g., image recognition, infotainment). Second, they can perform Vehicle-to-Vehicle (V2V) and Vehicle-to-Roadside (V2R) wireless communication, so the vehicles can exchange workload. Third, assuming electric vehicles, their driving range is limited by the energy available in the battery.

The workloads of modern vehicles need a substantial amount of processing power in order to meet their performance requirements. Offloading workload to remote cloud-based computing systems is often limited by a high communication latency, which significantly degrades performance.

In a VEC system, computational nodes can be deployed in cell towers, Road-Side Units (RSUs), and within connected vehicles so that local data and workloads can be processed with a much lower latency compared to using the cloud nodes. On the other hand, these edge nodes have often limited computing capacities and energy budgets (e.g., in electric vehicles). These characteristics make the problem of ensuring good performance while minimizing the energy consumption in VEC systems a challenging task.

1.7 Contributions of this research

Efficient resource allocation on edge servers is one of the main challenges in edge computing. Due to the mobility of users, a poor allocation might impose high execution costs. Application placement is an important problem in edge computing which is associated with

deciding where the application should be run. In this problem, each application is represented as a graph in which each node is a component of the application and each weighted edge between two nodes indicates the communication between two components. Physical resources can also be represented by a graph in which the nodes represent computing resources (servers) and the edge between each two nodes denotes a communication link between two servers. The application placement problem is the problem of mapping application graphs onto the physical graphs. Application placement in edge computing has to consider several issues that were not present in the data-center or cloud computing settings. After the initial placement, mobile users may move to different locations [14, 15]. Therefore, an optimal decision made at the time of receiving a request may not remain optimal for the whole duration of user's application execution. Furthermore, compared to the cloud data centers, edge nodes have more restricted capacity. Therefore, it is not feasible to run a large size application on a single edge node. An efficient way to resolve this issue is to allow users to run the components of their applications on multiple edge nodes.

Application placement. In this dissertation, we address the Multi-Component Application Placement Problem (MCAPP) in MEC systems. We formulate this problem as a Mixed-Integer Linear Program (MILP) with the objective of minimizing the total cost of running the applications. In our formulation, we take into account two important and challenging characteristics of MEC systems, the mobility of users and the network capabilities. We analyze the complexity of MCAPP and prove that it is NP-hard, that is, finding the optimal solution in reasonable amount of time is infeasible. We design two algorithms, one based on matching and local search and one based on a greedy approach. We evaluate the performance of the algorithms by conducting an extensive experimental analysis driven by two types of user mobility models, real-life mobility traces and random-walk. The results show that the proposed algorithms obtain near-optimal solutions and require small execution times for reasonably large problem instances. The results of this research are published in Proceedings of the Second ACM/IEEE Symposium on Edge Computing

(SEC-2017) [16], IEEE Transactions on Cloud Computing [17], and the Proceedings of the International Conference on Edge Computing (EDGE-2019) [18].

Resource allocation and pricing. In MEC systems, due to limitation on resources capacity, the competition to get these resources is high. Thus, monetization and resource allocation is considered as one of the major challenges in these systems. In other words, the resource provider has to decide how to allocate and price edge/cloud resources so that a given system's objective, such as revenue or social welfare, is optimized. One promising approach is to allocate these resources based on auction models, in which users place bids for using a certain amount of resources. In another contribution of this research, we address the problem of resource allocation and pricing in a two-level edge computing system. We consider a system in which servers with different capacities are located in the cloud or at the edge of the network. Mobile users compete for these resources and have heterogeneous demands. We design an auction-based mechanism that allocates and prices edge/cloud resources. The proposed mechanism is novel in the sense that it handles the allocation of resources available at the two-levels of the system by combining features from both position and combinatorial auctions. We show that the proposed mechanism is individually-rational and produces envy-free allocations. The first property guarantees that users are willing to participate in the mechanism, while the second guarantees that when the auction is finished, no user would be happier with the outcome of another user. We evaluate the performance of the proposed mechanism, G-ERAP, by performing extensive experiments. The experimental results show that the proposed mechanism is scalable and obtains efficient solutions. The results of this research are published in the Proceedings of the Third ACM/IEEE Symposium on Edge Computing (SEC-2018) [19]. An extended version of this paper in which we also developed an LP-based approximation mechanism that does not guarantee envy-freeness, but it provides solutions that are guaranteed to be within a given distance from the optimal solution has been accepted for publication in the

IEEE Transactions on Parallel and Distributed Systems [20].

Resource management in VEC systems. The low-latency requirements of connected electric vehicles and their increasing computing needs have led to the necessity to move computational nodes from the cloud data centers to edge nodes such as road-side units (RSU). However, offloading the workload of all the vehicles to RSUs may not scale well to an increasing number of vehicles and workloads. To solve this problem, computing nodes can be installed directly on the smart vehicles, so that each vehicle can execute the heavy workload locally, thus forming a vehicular edge computing system. On the other hand, these computational nodes may drain a considerable amount of energy in electric vehicles. It is therefore important to manage the resources of connected electric vehicles to minimize their energy consumption. One promising way to improve the energy efficiency is to share and coordinate computing resources among connected EVs. However, the uncertainties in the future location of vehicles make it hard to decide which vehicles participate in resource sharing and how long they share their resources so that all participants benefit from resource sharing. We propose VECMAN, a framework for energy-aware resource management in VEC systems composed of two algorithms: (i) a resource selector algorithm that determines the participating vehicles and the duration of resource sharing period; and (ii) an energy manager algorithm that manages computing resources of the participating vehicles with the aim of minimizing the computational energy consumption. We evaluate the proposed algorithms and show that they considerably reduce the vehicles' computational energy consumption compared to the state-of-the-art baselines. Specifically, our algorithms achieve between 7% and 18% energy savings compared to a baseline that executes workload locally and an average of 13% energy savings compared to a baseline that offloads vehicles' workloads to RSUs. The results of this research were published in the IEEE International Conference on Cloud Engineering (IC2E- 2020) [21], IEEE Transactions on Mobile Computing [22], and the Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking (EdgeSys-2019) [23].

1.8 Organization

The rest of this dissertation is organized as follows. In Chapter 2, we present the related work. In Chapter 3, we present our research on the design of efficient algorithms for multi-component application placement problem in edge computing system. In Chapter 4, we present mechanisms for resource allocation in edge computing systems. In Chapter 5, we present VECMAN, a framework for energy-aware resource management in VEC systems. Finally, in Chapter 6, we present our conclusion and future research.

CHAPTER 2

RELATED WORK

Edge nodes' proximity to users is a promising feature of MEC that can be exploited to improve the latency of the system. On the other hand, the mobility of users, the limited capacity of resources, and the dynamic nature of demands are critical issues that can affect the performance of the system. In this chapter, we survey the development of various methods for computation offloading, resource allocation, and application placement in edge computing systems. We aim at analyzing how the mentioned challenges have been addressed in the literature.

2.1 Application placement

The existing techniques for application placement developed for cloud computing/data centers settings [24, 25] cannot be applied directly in the MEC setting because, when making the placement decisions, they do not consider the mobility of users and the differences in latency experienced by users at different locations. In this section, we review the existing literature on computation offloading and application placement problems in MEC addressing the above critical issues.

Many studies focused on computation offloading, where the computation requirements of applications and network conditions are taken into account to decide which tasks must be run locally and which tasks must be migrated to remote servers. In some studies [26, 27, 28, 29, 30], energy consumption and computing latency have been considered as important performance metrics in optimizing computation offloading. The revenue of the service providers [31] and system's utility [32] were also considered as objectives in the computation offloading problem.

Several approaches for solving variants of the application placement problem in MEC have been proposed recently. Many of the dynamic application placement approaches formulated the problem as a sequential decision making problem in the framework of Markov Decision Processes (MDPs). Ksentini et al. [33] modeled the application/service migra-

tion problem considering the mobility of users in the Follow Me Cloud paradigm using MDPs [34]. In their formulation, they considered one dimensional mobility patterns. They implemented the value iteration algorithm in MATLAB to find the optimal application migration policy. Urgaonkar et al. [35] modeled the application placement problem as an MDP. To reduce the state space, they converted the problem into two independent MDP problems with separate state spaces and designed an online algorithm for the new problem that is provably cost-optimal. Wang et al. [36] presented a novel online algorithm for the application placement problem in the context of MEC. They modeled the problem as an MDP in which states are defined only based on the distance between user and servers.

Some researchers studied application placement problems for specific types of application graphs. Wang et al. [37] designed an online approximation algorithm for the placement problem in which both the application and the resource graphs are trees. The considered objective is to minimize the maximum weighted cost on each physical node and link of the system. Pei et al. [38] and Zou et al. [39] investigated the service chain embedding problem in MEC systems in which the application graph is a linear chain.

A few studies have focused on the placement of both servers and applications in a given network to improve the system performance [40, 41, 42, 43]. Balancing the load of servers and minimizing the delay of applications are two objectives that are considered in these studies. Several solutions have been proposed for the application placement problem that minimize the cost or the latency [44, 45, 46, 47]. These studies focused on the placement of the whole application on a single server and did not consider the possibility of assigning different components of an application to different edge servers.

2.2 Resource allocation and pricing

A wide range of metrics could be considered when optimizing resource allocation in EC systems, and depending on the network system, business rules, and assumptions, different constraints should be taken into account. Due to this fact, researchers have addressed a broad spectrum of resource allocation problems in these systems. Some researchers have

devoted their efforts on developing novel algorithms for computation offloading, where computation requirements of applications, network conditions, and users' preferences are taken into account to decide which tasks must be migrated to remote servers [48, 49, 50, 51, 52, 53]. Service placement has been another area of interest for researchers, that is, determining the most efficient server to run a service, where servers could be located in clouds or at the edge of the network [16, 29, 54, 55, 56].

Monetization of services has been identified as a grand challenge in edge computing systems [6]. This fact has led several researchers to concentrate their efforts on designing incentive-based resource allocation mechanisms. In contrast to the significant efforts focused on developing incentive-based resource allocation mechanisms for MCC systems [57, 58, 59], there are only few studies that have recently addressed this challenge in MEC systems.

Xiong et al. [60] proposed a pricing mechanism for running mining processes of mobile blockchain applications on the edge servers. They formulated the problem as a two-stage Stackelberg game with the aim of maximizing the profit of the service provider and the individual utilities of the miners. In another work on employing MEC for mobile blockchain, Jiao et al. [61] developed a truthful mechanism that maximizes the social welfare. The authors have considered a single service provider and multiple users who are competing for a single type of computational resource on the edge servers. Luong et al. [62] also studied the problem of resource allocation in edge computing systems for mobile blockchain applications. They adopted a deep learning approach based on a multi-layer neural network architecture to optimize the loss function which has been defined as the expected, negated revenue of the service provider. Li et al. [63] developed a learning-based pricing mechanism in which no profit information of users is required. At equilibrium the edge server induces self-interested users to choose the correct priority class (based on their delay sensitivity) and make socially optimal offloading decisions. However, none of the learning-based approaches consider relative valuations for different computing resources. Baek et

al. [64] developed an auction-based mechanism for resource allocation in edge computing in which users bid to get more CPU cycles from the edge server. The authors showed that there exists a unique Nash Equilibrium (NE) in the system. Therefore, the payments of users will tend to be fixed after competitions, regardless of the initial payments, and the edge seller can predict the strategies of users and determine the amount of CPU cycles they need. Chen et al. [65] studied the problem of multiple resource allocation and pricing in edge computing systems. They decomposed the problem into a set of sub-problems in which each sub-problem only considers a single type of resources. They constructed a Stackelberg game framework for each subproblem and developed algorithms to compute the Stackelberg equilibrium for each type of resource. Kiani et al. [66] proposed a three-level hierarchical architecture for mobile edge computing and an auction-based mechanism for VM pricing and resource allocation. Their pricing mechanism is based on the Amazon's Elastic Compute Cloud (EC2) spot pricing. The system determines the price of each type of VMs in each access point. However, their problem assumes a non-combinatorial auction (NC-Auction) setting, where requests are placed for only one VM instance of a single type. In that setting, bundles of VM requests are not allowed, and if a user is willing to request multiple VM instances of the same type, he/she needs to submit multiple bids. Submitting multiple bids for individual VM instances involves the risk of ending up obtaining only a subset of the requested set of VM instances. However, our mechanisms allow requests for bundles of VM instances and for multiple VM instances of the same type.

Several researchers have applied some of the traditional auction models for solving resource allocation problems in cloud computing systems [67, 68, 69]. The Vickrey-Clarke-Groves (VCG) auction has been one of the most popular truthful auctions [70]. The VCG auction is known to be incentive-compatible and socially optimal. Since achieving truthfulness in VCG requires the optimal solution of the social welfare maximization problem, it is practically infeasible to be applied for problems where the exact solution to the social welfare maximization cannot be obtained in a reasonable amount of time. Also, other tra-

ditional auction models are not directly applicable to the edge computing settings where resources are distributed over multiple levels, users typically request bundles of multiple types of resources, and they have different valuations for the services provided from different levels. Since, users request bundles of multiple types of resources, the EC auction mechanisms fall into the class of combinatorial auctions. In addition, since users have different valuations for resources at different levels, the mechanisms can be classified as position auctions.

In fact, having a bundle of requests from each user classifies the auction as a combinatorial auction, while the relative valuation of resources puts the auction in the class of position auctions. Many traditional mechanisms designed for combinatorial auctions guarantee the truthfulness, while the truthfulness of the position auctions is not guaranteed [71]. An example of a non-truthful position auction mechanism is the generalized second price auction employed by Google to sell online advertising [72].

2.3 Resource management in VEC systems

A highly efficient offloading mechanism for VEC systems has to overcome several challenges that do not exist or are less significant in the cloud-based systems. Due to the dynamic nature of VEC systems that mainly stems from the mobility of vehicles, an optimal offloading for the current setting of the network might turn to the least efficient offloading in few seconds/minutes. Furthermore, the variability of the available resources over time affects two important performance metrics, i.e., the reliability of the network and the QoS.

High QoS. Some of the existing challenges of resource management in VEC systems have been addressed from different perspectives and using different approaches. Several studies focused on ensuring high QoS. Zheng et al. [73] addressed the variability of the resources in a system in which clouds, parked vehicles, and mobile vehicles provide computational services. In their system, the goal is to maximize the amount of power saving of vehicles while the latency and transfer costs are minimized. Yu et al. [13] proposed a hierarchical VM mi-

gration mechanism for mobile vehicles by integrating computing resources in data centers, RSUs, and vehicles. They presented a layered structure that allows vehicles to select their services resiliently. Yang et al. [74] proposed a task offloading scheme for cooperative edge servers scenario with the aim of minimizing the cost of offloading while guaranteeing the QoS. However, these studies considered integrated edge resources when making offloading decisions and did not consider the distributed and mobile nature of servers.

High Reliability. Some researchers have also considered speculative execution as a technique to ensure high QoS and low risk of failure. Zhiyuan et al. [75] studied the task replication problem to minimize the probability of deadline violations. Zhu et al. [76] proposed an online algorithm for task replication in vehicular fog computing with the aim of minimizing the maximum service latency while minimizing the total quality loss of tasks. Sun et al. [77] proposed a learning based algorithm for task replication in VEC systems. Their objective is to minimize the average offloading delay of tasks. Zhou et al. [78] considered a cooperative vehicle infrastructure system and addressed the execution time minimization problem. Bahreini et al. [23] developed a speculative execution framework for VEC systems, where the number of replicas for each requests is predetermined and is given by an energy manager. The replica manager decides how to allocate copies of the vehicles' workload on different nodes to ensure high reliability and low latency. Hou et al. [79] developed a computation offloading mechanism for latency-sensitive applications. To ensure high reliability, they jointly optimized task allocation and the reprocessing of failed subtask executions. However, these studies do not balance the number of replicas with the energy consumption: having a high number of replicas may lead to a small improvement in robustness to failure while causing energy waste on vehicles.

QoS vs. Energy. Several solutions have been proposed to trade off between latency and energy consumption in mobile edge computing systems [80, 81, 82, 83, 84, 85, 86]. These studies investigate how to account for the interference of multiple service requesters shar-

ing service providers with consideration of latency and energy consumption. In particular, Viswanathan et al. [85] collected statistics on availability of connections between service requesters and service providers to improve the QoS. Workload is then migrated across service providers to handle unexpected scenarios (e.g., loss of connectivity or empty battery). However, all the above solutions have a single point of failure, i.e., if the selected service provider fails, the workload has to be migrated or offloaded again, which increases the latency. Note that some of the studies mentioned above (e.g., [13, 76, 77]) use task replication to reduce the risk of failure but without consideration of the impact of the number of replicas on the energy consumption.

Resource allocation in VEC ssytems has received a considerable attention in the literature. Various approches have been developed for optimizing the performance of VEC systems. However, to the best of our knowledge, none of the above studies consider at the same time the problems of (a) determining the duration of resource sharing, (b) coordinating the computing resources among moving vehicles, and (c) determining the number of replicas for vehicular workloads to minimize the energy consumption of vehicles without violating the desired QoS levels.

CHAPTER 3

EFFICIENT ALGORITHMS FOR MULTI-COMPONENT APPLICATION PLACEMENT IN MOBILE EDGE COMPUTING

3.1 Introduction

The widespread usage of mobile devices generates an unprecedented amount of data that often requires real-time processing. This processing necessitates computational resources and storage capacity not available on mobile devices. Cloud computing is a promising technology that allows the mobile applications to offload their computations on cloud servers [87, 88, 89]. The main objective of offloading is to extend the battery life of mobile devices by executing heavy-computational components of the applications on remote servers. However, in cloud settings, computing services are usually far away from the end-user, and therefore, the communication between mobile devices and servers requires many network hops and results in high latencies. This is unfeasible for applications that require a very low latency or transmit large amounts of data [3].

In order to resolve this issue, several paradigms such as Cloudlet [4], Fog Computing [7], Follow Me Cloud [34], and Mobile Edge Computing (MEC) [90] have been recently proposed. The core idea of these paradigms is to offload a portion of data/computation to the edge of the network rather than offloading it to the cloud data-centers. Satyanarayanan et al. [4] proposed Cloudlet with the aim of bringing the cloud closer to the end user. A Cloudlet, also known as a micro-cloud, is a cluster of multi-core computers with high internal connectivity that is available to nearby mobile devices and provide computing, bandwidth, and storage services. Users access the Cloudlet servers via a local area network such as Wi-Fi. MEC [90] has been recently introduced to provide the required infrastructure for low latency computing services through running mobile applications at the edge of the network, where an edge can be any computing resources of the network. In MEC, the edge nodes are widely distributed in the network and available to all mobile users. On the other hand, being co-located with base stations, edge nodes have access to

some additional information such as location and mobility of users.

One of the challenging issues in MEC systems is the resource scarcity. Compared to the cloud data centers, edge nodes have more restricted capacity. Therefore, it is not feasible to run a large size application on a single edge node. An efficient way to resolve this issue is to allow users to run the components of their applications on multiple edge nodes. Platforms such as Open Edge [91] and Open Fog [92] are developed for this purpose. They deploy virtualization techniques to share the resources of the edge nodes that are located in the same geographical region. In these platforms, finding an efficient placement for the components of an application on the multiple nodes is a major challenge.

Mobile users change their locations dynamically and the current assignment of the application to the edge nodes might not be the best in terms of the costs involved. In addition to the mobility of users, the resource availability and network conditions may also change dynamically. Therefore, in order to provide high quality services with the minimum costs, the application may need to migrate from one edge/core node to another, dynamically. The problem becomes more complex when an application has multiple components with heterogeneous requirements. The problem of assigning components of an application to the edge/core nodes such that the total cost of execution is minimized is called the *Multi-Component Application Placement Problem* (MCAPP). The components of a users' application can be run either on the core cloud, or on the edge of the network.

In MCAPP, an application can be represented as a graph in which the components of the application are the vertices, and the edges between two vertices represent the communication between the corresponding components. Similarly, physical resources can be represented as a graph in which vertices are the computing resources (i.e., servers) and the edges between two vertices represent the communication links between the corresponding physical resources. Thus, MCAPP can be viewed as the problem of mapping the application graph onto the resource graph.

3.1.1 Our contributions

Our main contributions are as follows:

(1) Formulate the MCAPP problem as a Mixed Integer Non-Linear Program (MINLP). Our formulation of the problem departs from the existing work since it does not impose any restrictions on the topology of the graphs characterizing both the applications and the physical resources.

(2) Prove that MCAPP is NP-hard, which means that it is not solvable in polynomial time, unless $P = NP$.

(3) Design two efficient algorithms for solving MCAPP. Our goal is to design heuristic algorithms based on purely combinatorial techniques such as matching and local search, and to avoid the use of stochastic control-based approaches. The proposed algorithms have low complexity, and thus, add a negligible overhead to the execution of the applications.

(4) Evaluate the performance of the proposed algorithms by an extensive experimental analysis. The experiments are driven by two types of user mobility models, one derived from real-life mobility traces [93] and the other one based on the random-walk model [94]. We compare the performance of our algorithms against the optimal solution under the two types of mobility models. Our experimental results show that the proposed algorithms obtain near optimal solutions and require very small execution times.

3.1.2 Organization

The rest of the chapter is organized as follows. In Section 3.2, we introduce the multi-component application placement problem and present its MINLP formulation. In Section 3.3, we present the proposed heuristic algorithms. In Section 3.4, we illustrate the execution of the algorithms on a small instance of the problem. In Section 3.5, we present

and analyze the experimental results. In Section 3.6, we conclude the chapter and suggest possible directions for future research.

3.2 Multi-component application placement problem

In this section, we formulate MCAPP in MEC systems. We consider a time slotted system, where T is the total number of time slots required to complete the execution of the application. The goal of the system is to determine the allocation of the components of the application in each time slot so that the total cost over T time slots is minimized. We formulate the problem for each time slot, where the relocation costs are determined by the allocation on the previous time slot. To make the formulation easier to understand and to avoid the use of an additional index to indicate the time slot for each variable, we present the problem only for one time slot.

In this formulation, we consider a two-dimensional grid area managed by an edge provider that periodically runs a resource manager. The system is composed of m servers $\{S_1, \ldots, S_m\}$ that are located at the edge of the network (e.g., at base stations). Note that in the rest of the chapter, we use S_i and i interchangeably when referring to server S_i. We assume that the location of users may change from one time slot to another, where the location of a user is specified by its coordinates in a two-dimensional grid of cells.

The user requests to offload an application with n components $\{C_1, \ldots, C_n\}$. In the rest of the chapter, we use C_j and j interchangeably when referring to component C_j. The processing requirement of component j is denoted by p_j. This represents the amount of component j's load that needs to be processed. We do not impose any restrictions on the communication between the components, any component can communicate with any other component of the application (i.e., the graph modeling the application is not restricted). We also assume that a server can communicate with any other server (e.g., via internet) incurring different costs for different servers. Here, the *objective* is to find an assignment of components to servers, such that the total placement cost of the application is minimized. The total placement cost is composed of four types of costs:

(i) γ_{ij}: *the cost of running component j on server i.* This cost is defined as the product of the cost of processing a unit load at server i and the amount of load that needs to be processed:

$$\gamma_{ij} = c_i \cdot p_j \tag{3.1}$$

(ii) $\rho_{ii'j}$: *the cost of relocating component j from server i to server i'.* In MEC, the locations of users may change during the execution of their applications. Also, the workload of the edge servers and other conditions of the network may vary from time to time. Therefore, it may be required to change the location where the components are running. The relocation cost is defined as follows:

$$\rho_{ii'j} = l_{ii'} \cdot q_j \cdot r \tag{3.2}$$

where $l_{ii'}$ is the distance between servers i and i', q_j is the size of component j that would migrate, and r is the cost of transferring one unit of data over one unit of distance. Since the managed area is a two-dimensional grid, the distance between servers is the Manhattan distance, that is, if server i is located in cell (x, y) and server i' is located in cell (x', y'), then the distance between the two servers is given by $l_{ii'} = |x - x'| + |y - y'|$.

(iii) δ_{ij}: *the communication cost between component j (assigned to server i) and the user.* In each time slot, data communication between components and the user may be required. This cost is defined as follows:

$$\delta_{ij} = d_i \cdot h_j \cdot r \tag{3.3}$$

where h_j is the size of data that must be transferred between component j and the user, and d_i is the distance between server i (that runs component j) and the user.

Table 3.1: Notation

Notation	Description
m	Number of servers.
n	Number of components.
γ_{ij}	Cost of running component j on server i.
c_i	Cost of processing one unit of load on server i.
p_j	Processing requirement of component j.
$\rho_{ii'j}$	Cost of relocating component j from server i to server i'.
$l_{ii'}$	Distance between servers i and i'.
q_j	Size of component j.
r	Cost of transferring one unit of data over a unit of distance.
δ_{ij}	Communication cost between component j (assigned to server i) and the user.
d_i	Distance between server i and the user.
h_j	Size of data that needs to be transferred between component j and the user.
$\tau_{ii'j'}$	Communication cost between components j and j' that are located on servers i and i', respectively.
$g_{jj'}$	Size of data that needs to be transferred between components j and j'.
x_{ij}	Binary variable associated with the assignment of component j to server i.

The distance between the server and user is the Manhattan distance as defined in (ii) above.

(iv) $\tau_{ii'jj'}$: *the communication cost between components j and j' that are located on servers i and i', respectively*. Suppose that component j is located on server i and component j' is located on server i'. The communication cost between components is defined as follows:

$$\tau_{ii'jj'} = l_{ii'} \cdot g_{jj'} \cdot r \tag{3.4}$$

where $g_{jj'}$ is the size of data that must be transferred between component j and component j'.

Considering these, the total cost of the placement, which is the objective function of

MCAPP, is given by,

$$\sum_{i=1}^{m}\sum_{j=1}^{n}(\gamma_{ij}+\delta_{ij})\cdot x_{ij}+\sum_{i=1}^{m}\sum_{i'=1}^{m}\sum_{j=1}^{n}\rho_{ii'j}\cdot\bar{x}_{i'j}\cdot x_{ij}+\sum_{i=1}^{m}\sum_{i'=1}^{m}\sum_{j=1}^{n}\sum_{j'=1}^{n}\tau_{ii'jj'}\cdot x_{ij}\cdot x_{i'j'} \tag{3.5}$$

The decision variables x_{ij} are defined as follows: $x_{ij}=1$, if component j is assigned to server i in the current time slot; and 0 otherwise. Furthermore, $\bar{x}_{i'j}$ is not a decision variable but a parameter denoting the assignment of component j in the previous time slot, that is, $\bar{x}_{i'j}=1$ if component j was assigned to server i' in the previous time slot, and 0, otherwise. Note that in any time slot, the assignment of components in the previous time slot is known. Therefore, the objective function can be rewritten as,

$$\sum_{i=1}^{m}\sum_{j=1}^{n}(\omega_{ij}\cdot x_{ij}+\sum_{i'=1}^{m}\sum_{j'=1}^{n}\tau_{ii'jj'}\cdot x_{ij}\cdot x_{i'j'}) \tag{3.6}$$

Note that to make it easier to work with the objective function, we define ω_{ij} as $\omega_{ij}=\gamma_{ij}+\delta_{ij}+(\sum_{i'=1}^{m}\rho_{ii'j}\cdot\bar{x}_{i'j})$. In the rest of the chapter, we call ω_{ij} the *server-component cost* and $\tau_{ii'jj'}$ the *inter-component cost*. In Table 3.1, we present the notation that is used throughout the chapter. We now formulate MCAPP as a Mixed Integer Non-Linear Program (MINLP) and show that it is NP-hard. Then, we provide two heuristic algorithms to solve it.

MCAPP-MINLP:

$$\min \quad \sum_{i=1}^{m}\sum_{j=1}^{n}(\omega_{ij}\cdot x_{ij} + \sum_{i'=1}^{m}\sum_{j'=1}^{n}\tau_{ii'jj'}\cdot x_{ij}\cdot x_{i'j'}) \tag{3.7}$$

subject to:

$$\sum_{j=1}^{n} x_{ij} \leq 1 \quad i = 1,\ldots,m \tag{3.8}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad j = 1,\ldots,n \tag{3.9}$$

$$x_{ij} \in \{0,1\} \quad i = 1,\ldots,m; \quad j = 1,\ldots,n \tag{3.10}$$

According to the above formulation, the objective function of MCAPP is to minimize the total placement cost. The set of constraints (4.3) guarantees that each server is used by at most one component. The set of constraints (4.4) ensures that each component is assigned to exactly one server. The set of constraints (3.10) represents the integrality requirement for the decision variables. The optimal solution obtained by solving MCAPP-MINLP will be used in the experimental results section as a lower bound for the solution obtained by our proposed algorithms.

3.2.1 Complexity of MCAPP

We show that the decision version (MCAPP-D) of MCAPP is NP-complete. This implies that MCAPP is NP-hard. An instance of MCAPP-D is defined by: an application graph, a resource graph, server-component cost, ω_{ij}, component-component cost, $\tau_{ii'jj'}$, and a bound $B \in \mathbb{R}^{+}$. In the application graph, each vertex corresponds to a component and the weight of the edge between every two verices j and j' gives the total amount of inter-component communication between the corresponding components (i.e., $(g_{jj'}+g_{j'j})\cdot r$). In the resource graph, each vertex corresponds to a server and the weight of the edge between every two vertices i and i' gives the distance between the corresponding servers (i.e., $l_{ii'}$). The decision question is whether there is an assignment of components to servers such that

the total cost of the assignment defined by Equation (3.6) does not exceed B.

Theorem 1. *MCAPP-D is NP-complete.*

Proof. We prove that MCAPP-D is NP-complete by showing that: (i) MCAPP-D belongs to NP, and, (ii) the Traveling Salesman Problem (TSP), a well-known NP-complete problem, can be reduced to this problem in polynomial time.

It is easy to show that MCAPP-D is in NP. We only need to guess an assignment from components to servers, and then, compute the total cost of the assignment (using Equation (3.6)) in polynomial time and check if it exceeds B or not.

For the second condition, we show that TSP is reduced to MCAPP-D in polynomial time. Let us define an arbitrary instance of TSP with bound L on the length of a tour, graph $G = (V, E)$, where V is the set of cities, E is the set of edges, and the weights $w_{ii'}$ for each edge (i, i') (i.e., the distance between cities i and i').

Now, we construct an instance of MCAPP-D, called P, based on G, such that the total cost of the assignment is less than B, if and only if we can find a tour in G with the total length less than L. Instance P has m servers and n components such that $m = n =\mid V \mid$. We also set $B = L$. In this instance, the resource graph, G', is the same as G and therefore, the distance between each pair of servers is the same as the distance between each pair of cities in G. For the application graph, we consider a ring graph in which the weight of the edge between each pair of adjacent vertices is 1 and the weight of edge between non-adjacent vertices is zero. Also, we assume that server-component computation costs, ω_{ij}, are zero.

We claim that G has a tour of total length less than L if and only if there is a solution for P, where the total cost is less than B. Let us consider a tour in G with total length less than L. We can consider this tour in G' and assign components to this tour in the order that they appear in the ring. Obviously, the total cost of this assignment is the same as the total length of the tour, and therefore, is less than B.

Conversely, suppose that there is an assignment from components to servers in P with

total cost less than B. We assign these components to the corresponding cities in G and define a tour based on the order of the component in the ring. Clearly, the total length of this tour is the same as the total cost of the assignment, which does not exceed L. □

3.3 Algorithms for MCAPP

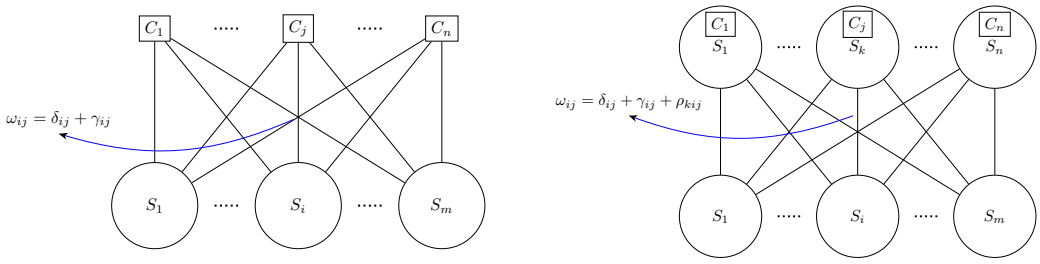
In the previous section, we showed that MCAPP is NP-hard. Therefore, it is not possible to find an optimal solution for it in polynomial time, unless $P = NP$. Thus, we need to design efficient algorithms that obtain near-optimal solutions to MCAPP in polynomial time. For this purpose, we design two efficient heuristic algorithms MATCH-MCAPP and G-MCAPP. MATCH-MCAPP is an algorithm based on matching and local search techniques. This algorithm is very efficient for MEC systems with a relatively low number of servers and components, and applications with less intensive communication among components. In the absence of inter-component communications, MATCH-MCAPP obtains the optimal allocation. G-MCAPP is a greedy algorithm that is more suitable for MEC systems with a large number of servers and components, as well as for applications with intensive communication among components. Thus, in practice they can be deployed based on the types of instances that need to be solved. In the following, we describe the algorithms and discuss their properties.

3.3.1 MATCH-MCAPP algorithm

We observe that the only factor that makes MCAPP NP-hard, is the existence of communication among components. Without this type of communication, the problem can be viewed as a matching problem (Figure 3.1a) which is solvable in polynomial time. In the first time slot, the problem is to match components to servers, where each assignment of component j to server i has a specific cost given by:

$$\omega_{ij} = \delta_{ij} + \gamma_{ij} \tag{3.11}$$

The relocation cost is not considered in the first time slot, since $\rho_{ikj} = 0$. In the next



(a) The first time slot.

(b) Other time slots.

Figure 3.1: Matching components to servers.

time slots, the problem is to reassign components to servers taking users' location dynamics and other mentioned factors into account. In other words, the algorithm must decide whether a component stays on the current server or migrates to another one (Figure 3.1). The cost of assigning component j to server i must include the relocation cost and thus, it is given by:

$$\omega_{ij} = \delta_{ij} + \gamma_{ij} + \rho_{kij} \tag{3.12}$$

where k is the location of component j in the previous time slot.

Based on this fact, we design our first algorithm, called MATCH-MCAPP. This algorithm operates in two phases. In the first phase, it determines the best matching between components of the application and the servers without considering the communication requirements among the components (i.e., $\tau_{ii'jj'} = 0$). For this purpose, the algorithm uses the Hungarian algorithm [95] which finds the minimum cost assignment of the components to servers. In the second phase, the algorithm considers the communication requirements among the components and uses a local search procedure to improve the solution.

The Hungarian algorithm is a polynomial time algorithm that solves the assignment problem optimally. The algorithm has as input the weights ω_{ij} of the edges of the bipartite graph in which one partition is composed of vertices corresponding to the servers, and the other composed of vertices corresponding to the application components. The algorithm finds a perfect matching that gives the minimum computation cost of components to servers. Once the assignment is determined, the algorithm takes into account the commu-

Algorithm 1 MATCH-MCAPP Algorithm

{Executed every time slot }
Input: $\boldsymbol{\omega}$: server-component costs
$\boldsymbol{\tau}$: inter-component costs
$\boldsymbol{y} \leftarrow$ HUNGARIAN($\boldsymbol{\omega}$)
$cost \leftarrow \sum_{j=1}^{n}(\omega_{y_j j} + \sum_{j'=1}^{n} \tau_{y_j y_{j'} j j'})$
$toVisit \leftarrow \{1, \ldots, n\}$
while $toVisit \neq \emptyset$ **do**
  for each $j \in toVisit$ **do**
    $\Lambda_j \leftarrow \sum_{j'=1}^{n} \tau_{y_j y_{j'} j j'}$
  end for
  $b \leftarrow \text{argmax}_{j \in toVisit}\{\Lambda_j\}$
  $toVisit \leftarrow toVisit \setminus \{b\}$
  for $i = 1, \ldots, m$ **do**
    $\boldsymbol{y} \leftarrow$ SWAP-COMPONENTS(y_b, i)
    $new\_cost \leftarrow \sum_{j=1}^{n}(\omega_{y_j j} + \sum_{j'=1}^{n} \tau_{y_j y_{j'} j j'})$
    if $new\_cost < cost$ **then**
      $cost \leftarrow new\_cost$
    else
      $\boldsymbol{y} \leftarrow$ SWAP-COMPONENTS(i, y_b)
    end if
  end for
end while
for $j = 1, \ldots, n$ **do**
  $x_{y_j j} \leftarrow 1$
end for
Output: $(X, cost)$

nication costs between the components, $\tau_{ii'jj'}$ and performs a local search procedure that obtains the final solution to MCAPP.

MATCH-MCAPP is given in Algorithm 1. The algorithm is executed in each time slot for each application. The input to the algorithm consists of the cost parameters, ω_{ij}, and $\tau_{ii'jj'}$. To make it easy to describe the algorithm, we use the following notation: $\boldsymbol{\omega}$ is the array of server-component costs; and $\boldsymbol{\tau}$ is the array of inter-component costs. The values of these parameters are determined during the previous time slot and are used as input to the algorithm in the current time slot. The output of the algorithm is the assignment matrix $X = \{x_{ij}\}$, and the total cost of running the application on the assigned servers, $cost$.

In the first phase, the algorithm determines the optimal assignment of the components

to servers by calling the function HUNGARIAN($\boldsymbol{\omega}$) (Line 1). This function implements a variant of the Hungarian algorithm and takes as input the cost $\boldsymbol{\omega}$ and returns the assignment as the vector $\boldsymbol{y}$; where $y_j = i$ if component j is assigned to server i. Since the Hungarian algorithm is well known we will not describe it here, but we refer the reader to Kuhn [95] .

Since the inter-component cost is not considered, the Hungarian algorithm is able to determine the optimal assignment of components to servers. This optimal assignment is not a solution for the MCAPP problem, it is an optimal assignment for the MCAPP with zero costs for the communication between components (i.e., $\tau_{ii'jj'} = 0$). Then, in the second phase, MATCH-MCAPP performs a local search that takes into account the cost of communication between components (Lines 2-15).

In the second phase, first, MATCH-MCAPP computes the cost of the current assignment determined by the Hungarian algorithm and also adds the inter-component costs to obtain the total cost (Line 2). Then, the algorithm defines a set, $toVisit$, and initializes it with the set of all components (Line 3). Next, it computes the total inter-component cost, Λ_j, of each component j in the $toVisit$ set (Lines 5-6). Then, in line 7, it determines the index of the *bottleneck component*, denoted by b. The bottleneck component is the component that has the maximum value for the total inter-component cost, Λ_j. The algorithm removes this component from the $toVisit$ set (Line 8). Thus, this component will not be selected as the bottleneck in the next iterations. After that, it executes a for loop (Lines 9-15) in which it tries to find a lower total cost assignment by swapping the component that is currently placed on server i with the bottleneck component. This is done by calling the function SWAP-COMPONENTS(y_b, i). This function swaps the components that are located at servers y_b and i, that is, assigns the bottleneck component to i and the component that is currently placed on server i to the server in which b resided. If there is no component on server i, then b is assigned to i and the server on which b resided is marked as available. The function outputs a new assignment vector $\boldsymbol{y}$. After this, in line 11, MATCH-MCAPP computes new_cost, the total cost of the system under the new assignment. If there is an

improvement in the cost, it updates the total cost, $cost$, otherwise it restores the previous assignment by calling the SWAP-COMPONENTS function (Lines 12-15). The algorithm continues this procedure as long as there is an unvisited component. Then, it updates the assignment matrix X based on the assignment vector y (Lines 16-17).

We now investigate the time complexity of MATCH-MCAPP. The time complexity of the first phase, Hungarian algorithm, is $O(\max(m^3, n^3))$. Since, we assume that the number of servers is greater than the number of components, the time complexity of the Hungarian algorithm is $O(m^3)$. The most computational expensive section of the second phase consists of lines 4-15. The time complexity of the first part of this section (Lines 5-8) is $O(n^2)$ while that of the second part (Lines 9-15) is $O(mn^2)$. Furthermore, since each component is not chosen as the bottleneck more than once, these two parts are not executed more than n times. Thus, the time complexity of the second phase is $O(mn^3)$. Therefore, the time complexity of MATCH-MCAPP is $O(m^3 + mn^3)$.

3.3.2 G-MCAPP algorithm

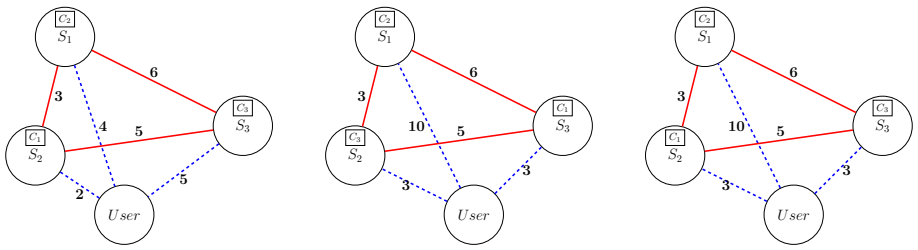
G-MCAPP is a greedy algorithm that finds the assignment of components to servers iteratively. To determine the assignment, the algorithm considers both the server-component and the inter-component costs simultaneously. The idea of G-MCAPP is to assign a component to a server in each iteration in such a way that the minimum cost is added to the total cost. For this purpose, the algorithm employs the *assignment cost* variable, σ_{ij}.

In the first iteration, since the assignment of none of the components has been determined yet, G-MCAPP only decides based on the server-component cost (i.e., $\sigma_{ij} = \omega_{ij}$). The algorithm selects a server-component pair (i^*, j^*) that has the minimum value of $\sigma_{i^*j^*}$. Then, for each unassigned server-component pair, (i, j), the algorithm updates the assignment cost by adding the inter-component cost between the previously selected component and the current component (i.e., $\sigma_{ij} = \omega_{ij} + \tau_{ii^*jj^*} + \tau_{i^*ij^*j}$). In the next iterations, G-MCAPP decides the assignment based on the updated costs and continues the procedure of selecting the server-component pair with the minimum assignment cost.

Algorithm 2 G-MCAPP Algorithm

{Executed every time slot }
Input: $\boldsymbol{\omega}$: server-component costs
$\boldsymbol{\tau}$: inter-component costs
1: $S \leftarrow \emptyset$
2: **for** $i = 1, \ldots, m$ **do**
3: **for** $j = 1, \ldots, n$ **do**
4: $\sigma_{ij} \leftarrow \omega_{ij}$
5: $S \leftarrow S \cup \{(i, j)\}$
6: **end for**
7: **end for**
8: **while** $(S \neq \emptyset)$ **do**
9: $(i^*, j^*) \leftarrow \text{argmin}_{(i,j)\in S}\{\sigma_{ij}\}$
10: $x_{i^*j^*} \leftarrow 1$
11: **for each** $(i, j) \in S$ **do**
12: **if** $i = i^*$ **or** $j = j^*$ **then**
13: $S \leftarrow S \setminus \{(i, j)\}$
14: **end if**
15: **end for**
16: **for each** $(i, j) \in S$ **do**
17: $\sigma_{ij} \leftarrow \sigma_{ij} + \tau_{ii^*jj^*} + \tau_{i^*ij^*j}$
18: **end for**
19: **end while**
20: $cost \leftarrow \sum_{i=1}^{m} \sum_{j=1}^{n} \omega_{ij} \cdot x_{ij} + (\sum_{i'=1}^{m} \sum_{j'=1}^{n} x_{ij} \cdot x_{i'j'} \tau_{ii'jj'})$
Output: $(X, cost)$

G-MCAPP is given in Algorithm 2. The input to the algorithm consists of: $\boldsymbol{\omega}$, the server-component cost matrix; and $\boldsymbol{\tau}$, the array of inter-component costs. In this algorithm, S is the set of all possible pairs of servers and components. For each pair (i, j), variable σ_{ij} is initialized to ω_{ij} (Lines 2-5). Then, iteratively, the algorithm finds the assignment of each component. First, the algorithm selects a server-component pair (i^*, j^*) that has the minimum server-component cost and assigns component j^* to server i^* (Lines 7-8). Since each component is assigned to exactly one server and each server is used by at most one component, the algorithm removes all pairs that contain i^* or j^* from set S (Lines 9-11). Then, for each unassigned component j and server i, the algorithm updates the assignment cost by considering the inter-component communication between the component j^* and component j (i.e., $\sigma_{ij} \leftarrow \sigma_{ij} + \tau_{ii^*jj^*} + \tau_{i^*ij^*j}$) (Lines 12-13). Therefore, in the next iteration of the algorithm, the assignment costs are updated and the inter-component communication



(a) The placement of components on servers in the previous slot.

(b) The placement of components obtained by MATCH-MCAPP.

(c) The placement of components obtained by G-MCAPP.

Figure 3.2: Example.

costs are considered. The algorithm continues this procedure until all the components are assigned. Finally, it determines the total cost based on the assignment of components (Line 14).

Now, we investigate the time complexity of G-MCAPP. The algorithm executes n iterations. The most time consuming part of each iteration consists of computing the minimum pair and updating the cost of the remaining pairs. In the first iteration, since there are $m \cdot n$ pairs, the time complexity of these operations is $O(mn)$. In the next iteration, the number of pairs reduces to $(m-1)(n-1)$. Generally, in the i-th iteration, the number of pairs is $O((m-i+1)(n-i+1))$. Therefore, the time complexity of G-MCAPP is $O(\sum_{i=1}^{n}(m-i+1)(n-i+1)) = O(mn^2)$.

3.4 An illustrative example

We provide a numerical example to show how our algorithms work. We consider an edge system consisting of three servers $\mathcal{S} = \{S_1, S_2, S_3\}$, and an application with three components $C = \{C_1, C_2, C_3\}$. Figure 3.2a shows the user-server and server-server distances (i.e., d_i and $l_{ii'}$) in the previous time slot. In this figure, the weights on solid line edges are the server-server distances and the weights on the dashed line edges are the user-server distances. We assume that in the previous time slot, components C_1, C_2, and C_3 have been assigned to servers S_2, S_1, and S_3, respectively.

In the next time slot, as the user's location changes, the user-server distances change

too (See Figure 3.2b). Therefore, the algorithms may need to change the assignment. Figure 3.2b and Figure 3.2c show the new assignment of the components in the next slot obtained by MATCH-MCAPP and G-MCAPP, respectively. In these figures, we observe that MATCH-MCAPP changes the location of components C_1 and C_2 while G-MCAPP decides not to change the location of any component. In the rest of this section, we show how these two algorithms decide on their assignment. The values of the parameters are provided in Table 3.2. Based on these parameters, we obtain ω_{ij} given in Table 3.3.

3.4.1 MATCH-MCAPP

In the first phase of MATCH-MCAPP, the Hungarian algorithm is employed to determine the placement, which is C_1 to S_1, C_2 to S_2, and C_3 to S_3 (Figure 3.3a). According to Equation (3.7), the total cost of this assignment is 757. In the second phase of MATCH-MCAPP, the local search, takes the inter-component communication ($\tau_{ii'jj'}$) into account. In each iteration of the local search, the total inter-communication cost, Λ_j, of each com-

Table 3.2: Example: The values of the cost parameters .

Parameter	Value
r	1
$< g_{12}, g_{13} >$	$< 12, 15 >$
$< g_{21}, g_{23} >$	$< 13, 20 >$
$< g_{31}, g_{32} >$	$< 20, 30 >$
$< h_1, h_2, h_3 >$	$< 5, 10, 10 >$
$< q_1, q_2, q_3 >$	$< 4, 2, 2 >$
$< p_1, p_2, p_3 >$	$< 2, 3, 2 >$
$< c_1, c_2, c_3 >$	$< 5, 10, 12 >$

(r, data transmission cost rate; $g_{jj'}$, size of data transferred between components j and j'; h_j, size of data transferred between user and component j; p_j, processing requirement of component j; c_i, cost of processing of one unit load on server i)

Table 3.3: Example: The values of the server-component costs, ω_{ij}.

i/j	1	2	3
1	72	115	122
2	32	66	62
3	51	76	54

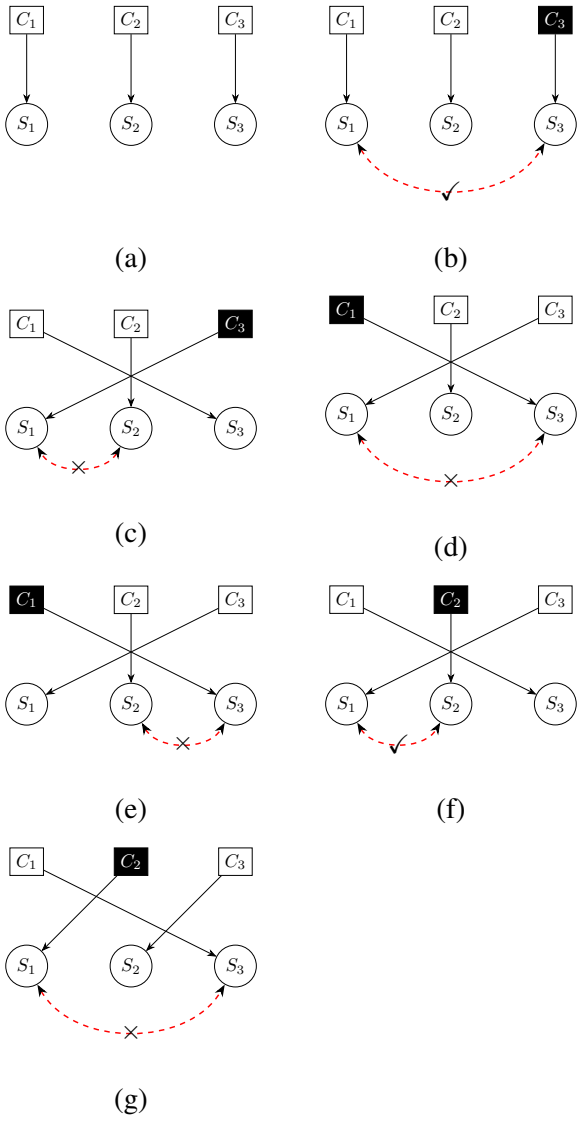


Figure 3.3: Example: Second phase (local search) of MATCH-MCAPP on an instance with three components and three servers.

ponent j is computed and the component with the maximum value of Λ_j is selected as the bottleneck. In the first iteration, the total communication cost of each component is computed: $\Lambda_1 = 285$, $\Lambda_2 = 325$, $\Lambda_3 = 460$. Therefore, component C_3 is selected as the bottleneck. In the figures, the bottleneck component is represented by a solid black square. Then, the algorithm swaps the bottleneck with the component in the next server if it leads to a reduction in the total cost. In this case, the algorithm assigns C_3 to S_1 and C_1 to S_3, because it reduces the total cost from 757 to 714 (Figure 3.3b). In the next iteration, the

algorithm skips swapping the bottleneck with component C_2 because the total cost of this possible assignment is 758 which is greater than the total cost obtained from the previous assignment. Since all possible swaps for the current bottleneck have been tried, the algorithm starts the next iteration to select another bottleneck. In the next iteration, the total inter-component communication costs of the remaining components are calculated: $\Lambda_1 = 335$, $\Lambda_2 = 275$. Therefore, C_1 is selected as the bottleneck. The algorithm skips swapping server S_3 with S_1 since it will not reduce the total cost (Figure 3.3d). In the next step, the algorithm again skips swapping S_3 and S_2, because it will not reduce the total cost (Figure 3.3e).

In the next iteration of the algorithm, component C_2 is selected as the bottleneck. The algorithm swaps S_2 with S_1 because it reduces the total cost from 714 to 703 (Figure 3.3f). The algorithm skips swapping S_1 with S_3 because it will not reduce the total cost (Fig 3.3g) and it stops because all the components have been visited. Therefore, the total placement cost obtained by MATCH-MCAPP is 703.

3.4.2 G-MCAPP

Now, we show how G-MCAPP determines the placement of the components. The algorithm initializes the assignment cost between each pair of components and servers, σ_{ij}, based on the values of the server-component costs (i.e., $\sigma_{ij} = \omega_{ij}$). In each iteration, the algorithm selects a pair of server i^* and component j^* for which the value of $\sigma_{i^*j^*}$ is minimum and assigns component j^* to server i^*. Then, it updates the value of σ_{ij} for the unassinged pairs of servers and components. Table 3.4 shows the values of the assignment costs in each iteration of G-MCAPP. The algorithm selects (C_1, S_2) with the cost of 32 as the pair with the minimum server-component cost and assigns C_1 to S_2. Then, it updates the assignment cost of each remaining pair (i, j) by adding the inter-component cost between C_1 and component C_j (i.e., $\sigma_{ij} = \omega_{ij} + \tau_{i2j1} + \tau_{2i1j}$). Therefore, for example the cost of assigning component C_2 to server S_1 is updated to $\sigma_{12} = \omega_{12} + \tau_{1221} + \tau_{2112}$. The values of the assignment costs obtained in the second iteration of the algorithm are given

Table 3.4: Example: Assignment costs, σ_{ij}, in each iteration of G-MCAPP

iteration	σ_{11}	σ_{12}	σ_{13}	σ_{21}	σ_{22}	σ_{23}	σ_{31}	σ_{32}	σ_{33}
1	72	115	122	**32**	66	62	51	76	54
2	–	190	227	–	–	–	–	201	**189**
3	–	**490**	–	–	–	–	–	–	–

in the second row of Table 3.4. In this table, the pairs that are not allowed to be selected (due to the Constraints (4.3) and (4.4)) are marked with " – ".

In the second iteration, the pair (C_3, S_3) with a cost of 189 is selected as the pair with the minimum cost and therefore, C_3 is assigned to S_3. Then, the assignment costs of the remaining pairs are updated (i.e., $\sigma_{ij} = \omega_{ij} + \tau_{i3j3} + \tau_{3i3j}$). In the last iteration, the algorithm assigns component C_2 to server S_1. Therefore, based on Equation (3.7), the total placement cost obtained by G-MCAPP is 765.

Comparing the results obtained by the two algorithms, the total cost obtained by G-MCAPP is 8.8% higher than that obtained by MATCH-MCAPP. In Section 3.5, we show that the quality of solutions obtained by MATCH-MCAPP is better than G-MCAPP for small-size problem instances, specifically, when the amount of inter-component communication is not high.

3.5 Experimental results

We perform extensive experiments in order to investigate the properties of the proposed algorithms. We compare the performance of the algorithms against that of the optimal solution obtained by solving MCAPP-MINLP and that of another placement algorithm. In the following, we describe the experimental setup and analyze the experimental results.

3.5.1 Experimental setup

Because the development of MEC is still in the early stages, there are no MEC workload traces that are publicly available. Therefore, for our experiments, we have to rely on synthetically generated instances for the MCAPP problem. In the following, we describe how we generate the problem instances that drive our simulation experiments and describe

the experimental setup.

We consider a time slotted system in which the locations of users in the network may change from one time slot to another, but do not change during one time slot. To evaluate the efficiency of the algorithms, we consider two different mobility models for the users: (i) Trace-Driven (TD), based on real-world mobility data [93], and (ii) Random Walk (RW) model [94].

For the trace-driven experiments, we use the CRAWDAD data set containing mobility traces of taxi cabs in San Francisco, CA [93]. The data set contains the GPS coordinates of about 500 taxi cabs collected over 30 days. We randomly choose the traces of 150 taxi cabs whose locations are updated every 10 seconds and use them as mobility traces for the users in our experiments. We also consider 200 edge servers that are co-located with 200 selected cell towers in the San Francisco area. The locations of these towers are obtained from antennasearch.com. In our experiments we do not consider towers with height less than 100 feet. Figure 3.4 shows the distribution of towers in San Francisco and the most frequent paths that are used by the selected 150 taxis in the area. For the trace-driven experiments, we set the length of a time slot to 5 minutes. Every experiment is repeated ten times and each time, we select a taxi randomly from the data set and run the experiments.

For the second sets of experiments, those using the random walk mobility model, we assume that the mobility of users is based on the random walk model in a two-dimensional space. The users and servers are located within a two-dimensional grid of 50×50 cells. In fact, we consider the area of the San Francisco taxi traces as a 50×50 grid. Initially, a user can be in any cell of the grid network and its location is drawn randomly from a uniform distribution over the locations of the grid. In our setting, in every new time slot, a user can stay in its place or move into any of neighboring cells with equal probability. The servers are located within the same two-dimensional grid network and the coordinates of their positions are the same as those of servers we considered in the experiments with the trace-driven data set. The distance between servers and users is the Manhattan distance (as

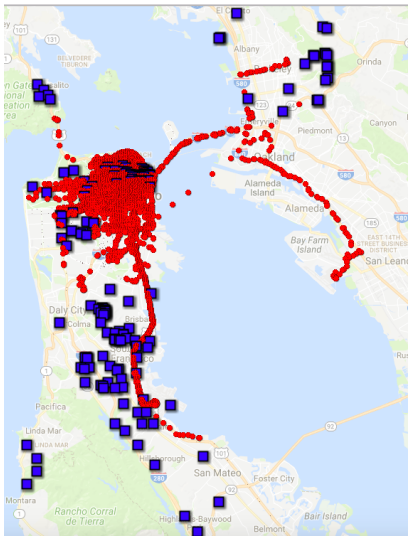


Figure 3.4: Distribution of the edge servers (blue squares) and the frequent paths of the users (red dots)).

(Image generated using GPS Visualizer [96]).

defined in Section 3.2).

We generate several problem instances with different values for n, the number of components of the application, and m, the number of servers in the network. The number of components for each application ranges from 2 to 100, while the number of servers ranges from 10 to 200. The reason for choosing these ranges is that in practice the number of components of an average application rarely exceeds 100 and most likely is on the lower part of the range considered here. Also, we assume that the number of time slots needed to run an application is 10. To generate the cost parameters defined in Section 3.2 we take into account the type of applications we consider.

Since the determinant factors in the performance of any algorithm for solving MCAPP are the server-component costs and the inter-component costs, we decided to generate the instances according to the value of a metric called *Inter-component cost to Server-component cost Ratio* (ISR). This metric is defined as the ratio of the average inter-component cost of each component (i.e., $\frac{\sum_{j=1}^{n}\sum_{j'=1}^{n}\bar{l}\cdot g_{jj'}\cdot r}{n}$) and the average server-component cost per assignment (i.e., $\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\omega_{ij}}{n\cdot m}$), where, $\bar{l}$ is the average distance between servers. Based on the value of ISR, we define three classes of applications with low, medium, and

Table 3.5: Simulation parameters

Parameter	Description	Distribution
c_i	Cost of processing one unit of load on server i.	$N(\mu_i, 0.2\mu_i)$, $\mu_i \sim U[1, 10]$
p_j	Processing requirement of component j.	$N(\mu_j, 0.2\mu_j)$, $\mu_j \sim U[0, 10]$
r	Data transmission cost.	$U[0, 1]$
q_j	Size of component j.	$U[10, 40]$
h_j	Size of data transferred between user and server j.	$U[1, 20]$
$g_{jj'}$	Size of data transferred between components j and j'	low :$U[1, 10]$ medium: $U[10, 100]$ high: $U[1000, 10000]$

high ISR.

Table 3.5 shows the type of distributions used to generate the parameters characterizing the problem instances used in our simulation experiments. We consider different ranges for the distribution for three classes of applications. All the cost parameters for these three types of applications are the same, except for parameter $g_{jj'}$. This parameter indicates the inter-component communication intensiveness of the application. In Table 3.5, we denote by $U[x, y]$, the uniform distribution within interval $[x, y]$, and by $N(\mu, v)$, the normal distribution with mean μ and variance v. We assume that the cost of processing one unit of load on the servers is within the same range for all servers and does not vary significantly. Therefore, we use the normal distribution for the cost of processing. Similarly, we use the normal distribution for the processing requirement of the components.

We compare the performance of our algorithms, MATCH-MCAPP and G-MCAPP, with that of another algorithm called MATCH and with that of the optimal solution obtained by solving MCAPP-MINLP. The MATCH algorithm implements a variant of the Hungarian algorithm [95] and does not take into account the communication among components when making the placement decisions. We compare with this algorithm in order to investigate the improvement in the quality of the solution due to considering the communication among components in the local search phase of MATCH-MCAPP.

For each type of instance, we determine the number of runs based on the observed

variance of the results [97]. We observe that the standard deviation of the results for ten random instances is low. Thus, we execute MATCH-MCAPP, G-MCAPP, and MATCH algorithms for ten random instances (all the plots presented in the next section show the average values). The performance of the algorithms is evaluated by computing the *performance ratio*, PR, which is defined as the ratio of the value V^* of the optimal solution for MCAPP-MINLP, and V, the value of the solution obtained by a given algorithm, (i.e., $PR = \frac{V^*}{V}$). To obtain the optimal solution, we transform MCAPP-MINLP into an equivalent mixed integer linear program (called MCAPP-MILP) and solve it with the CPLEX solver. The transformation is performed by replacing $x_{ij} \cdot x_{i'j'}$ in the objective with a binary variable $y_{iji'j'}$, and adding the following constraints to the program,

$$x_{ij} + x_{i'j'} - 1 \leq y_{iji'j'} \quad \forall i, j, i', j' \tag{3.13}$$

$$y_{iji'j'} \in \{0, 1\} \quad \forall i, j, i', j' \tag{3.14}$$

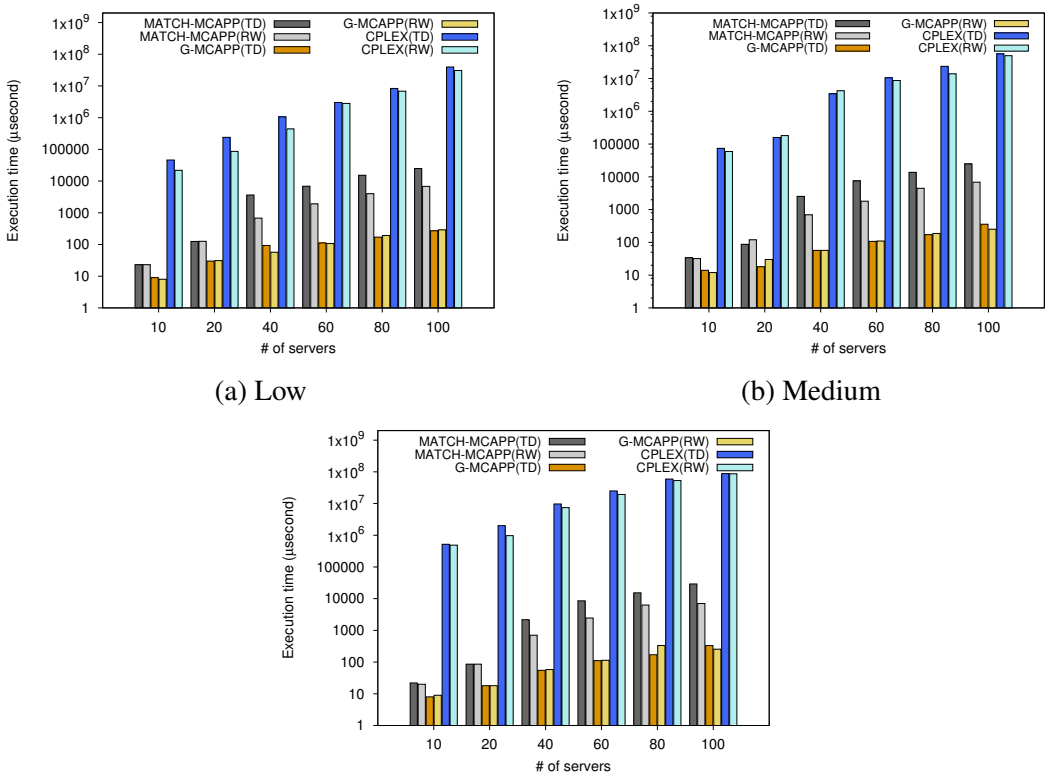
These constraints guarantee that binary variable $y_{iji'j'}$ is 1, if both variables x_{ij} and $x_{i'j'}$ are 1; and 0 otherwise.

The MATCH-MCAPP, G-MCAPP, and MATCH algorithms are implemented in C++ and the experiments are conducted on an Intel 1.6GHz Core i5 with 8 GB RAM system. For solving MCAPP-MILP we use the CPLEX 12 solver provided by IBM ILOG CPLEX optimization studio for academics initiative [98].

3.5.2 Analysis of results

In this section, we study the total placement cost of applications and compare the performance and the scalability of the algorithms for different types of applications with varying number of components and servers.

Performance with respect to the number of servers. We investigate the effect of the



(a) Low

(b) Medium

(c) High

Figure 3.5: Execution time (microseconds) vs. number of servers for different level of inter-component communications.

number of servers on the performance of MATCH-MCAPP and G-MCAPP algorithms considering the two mobility models. We characterize the performance of the algorithms using two main metrics, the performance ratio, and the execution time on a set of instances that runs the application over 10 time slots ($T = 10$) and consists of $n = 4$ components. We chose this type of instances in order to be able to solve them optimally using CPLEX and compare the performance of our algorithms with that of the optimal solution. We vary the number of servers from 10 to 100. These servers are chosen randomly from the data set. We select three types of instances for these experiments, instances with high inter-component costs, instances with medium inter-component costs, instances with low inter-component costs, and perform a detailed analysis of the results.

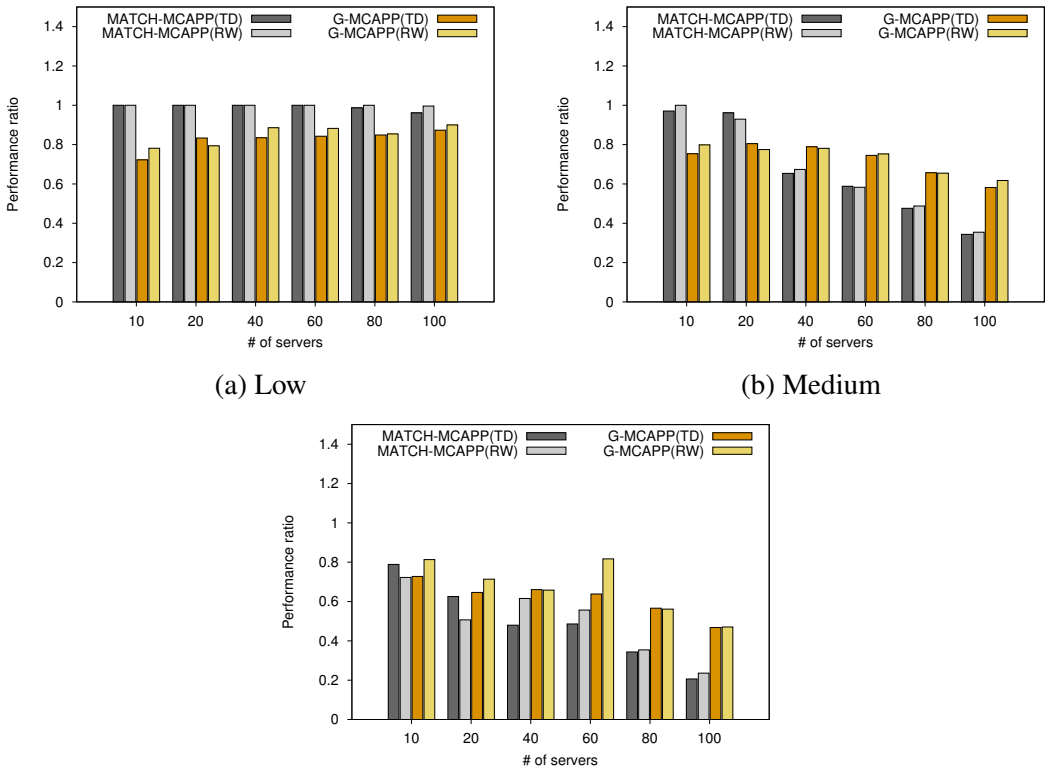
In Figure 3.5, we plot the average execution time per time slot obtained by MATCH-MCAPP, G-MCAPP, and CPLEX under both mobility models and on those instances

for different values of m using a logarithmic scale. In the plots, we denote by MATCH-MCAPP(TD) the cases in which MATCH-MCAPP is executed on instances generated using the trace-driven data sets, and by MATCH-MCAPP(RW), the cases in which MATCH-MCAPP is executed on the instances generated using the random-walk mobility model. We use a similar notation in the cases of G-MCAPP and CPLEX.

The execution time of CPLEX is several orders of magnitude higher than the execution times of both MATCH-MCAPP and G-MCAPP algorithms for all three types of instances. The execution time of MATCH-MCAPP and G-MCAPP, is under 1 millisecond for problem instances with small number of servers ($m < 40$), making them very suitable for deployment in real MEC systems.

We observe an increase of the execution time of MATCH-MCAPP with the increase in the number of servers. This is because of the cubic growth in terms of m of the running time of the algorithm. For example, in the case of instances with low inter-component communication, with $m = 20$ under the trace-driven model, the average execution time obtained by MATCH-MCAPP is around 0.08 milliseconds, while for $m = 80$, the average execution time is around 15 milliseconds. However, this execution time is reasonable because it is much less than the duration of a slot. Therefore, it will not make our algorithm a significant contributor to the overhead of placing the application components on edge servers.

Generally, the G-MCAPP algorithm obtains a lower execution time than MATCH-MCAPP. In all instances, the execution time obtained by G-MCAPP is under 1 millisecond. Also, we observe that in most cases, the total execution time of MATCH-MCAPP, G-MCAPP, and CPLEX under trace-driven data set is slightly greater than that under the random walk model. In fact, under the random walk model, a user changes his/her direction with the same probability in each time slot. Therefore, his/her distance from the servers may not change as significantly as the trace-driven case in which the user may only follow one direction for multiple consecutive time slots. Therefore, under the random walk model, the execution time of the algorithms is lower than that of trace-driven case, because



(a) Low

(b) Medium

(c) High

Figure 3.6: Performance ratio vs. number of servers for different level of inter-component communications.

the algorithms may not need to relocate components in each time slot.

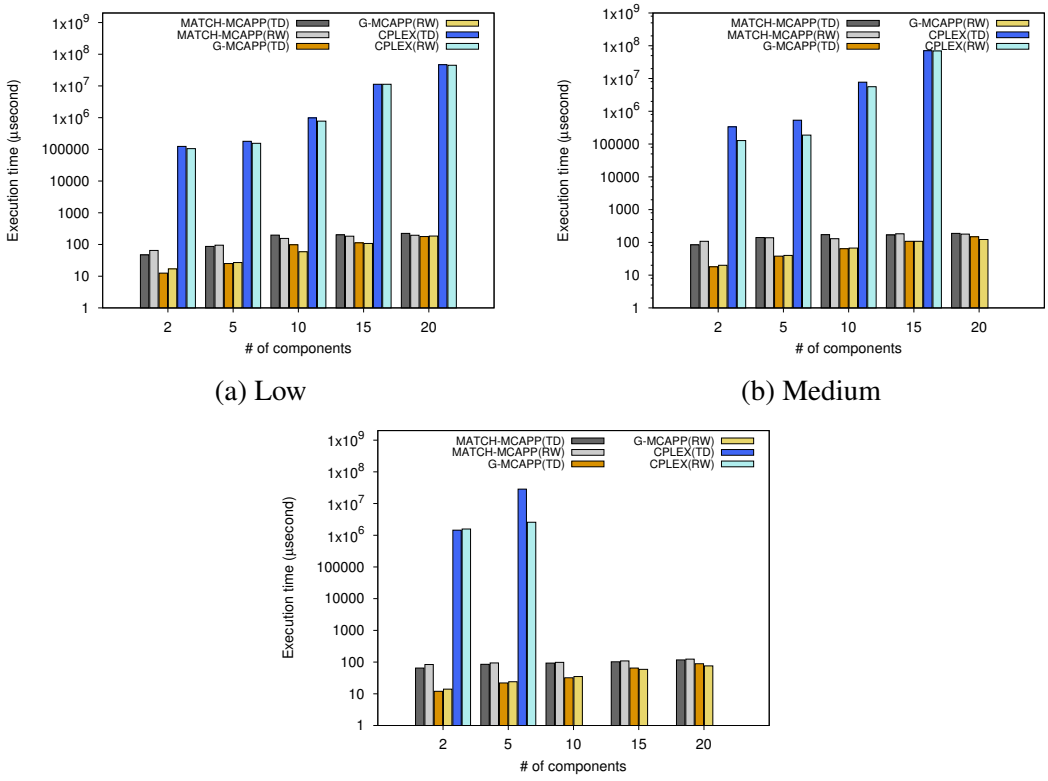
In Figure 3.6, we plot the performance ratio obtained by MATCH-MCAPP and G-MCAPP. In the case of low inter-component communication instances (Figure 3.6a), the performance ratio obtained by MATCH-MCAPP is very close to 1, thus this algorithm obtains optimal solutions or solutions that are very close to the optimal. The reason is that the inter-component communication is low, and therefore, the solution obtained by the Hungarian algorithm (inside the MATCH-MCAPP) is very close to the optimal solution. For those instances, G-MCAPP has a lower performance ratio. However, the performance ratio obtained by this algorithm is higher than 0.87 in all cases. Furthermore, by increasing the number of servers, the quality of solutions obtained by G-MCAPP improves. The reason is that by increasing the number of servers, a higher percentage of servers are available for components at each iteration of G-MCAPP. Therefore, G-MCAPP explores more

alternative placements and can make better decisions.

As the amount of inter-component communication increases, the performance ratios of MATCH-MCAPP and G-MCAPP increase (Figure 3.6b and 3.6c). We observe that G-MCAPP has a better performance compared to MATCH-MCAPP for higher inter-component communication cases. That means that G-MCAPP is able to obtain solutions that are closer to the optimal solution than those obtained by MATCH-MCAPP. As an example, for $m = 40$, for high inter-component communication case, the performance ratio of MATCH-MCAPP is 0.48, while that of G-MCAPP is 0.63. Also, we observe that the performance ratio decreases with the number of servers. Therefore, MATCH-MCAPP and G-MCAPP exhibit completely different behaviors compared to the low inter-component communication cases where the performance ratio increases by the increase of the number of servers. In fact, when the amount of inter-component communication is relatively high, each inefficient greedy decision can incur a significant amount of cost to the system. Thus, when the number of servers increases, there is a higher risk for local search/ greedy algorithms to make less efficient decisions.

Another important observation from Figure 3.6b and 3.6c is that the performance ratio obtained by MATCH-MCAPP decreases faster than the performance ratio obtained by G-MCAPP. Also, from Figure 3.6, we observe that the performance of MATCH-MCAPP and G-MCAPP under both trace-driven mobility data set and random walk model are consistent. In other words, the performance ratios obtained by the algorithms do not vary significantly under the considered mobility models. One reason for this consistency is the fact that our algorithms are relatively robust to the mobility behavior of users.

Performance with respect to the number of components. Next, we analyze the performance of MATCH-MCAPP and G-MCAPP by varying the number of components. We consider a set of instances that require running the application for 10 time slots ($T = 10$) and consist of 20 servers (we randomly choose 20 towers from the data set). We chose this type of instances in order to be able to solve them optimally using CPLEX and compare



(a) Low

(b) Medium

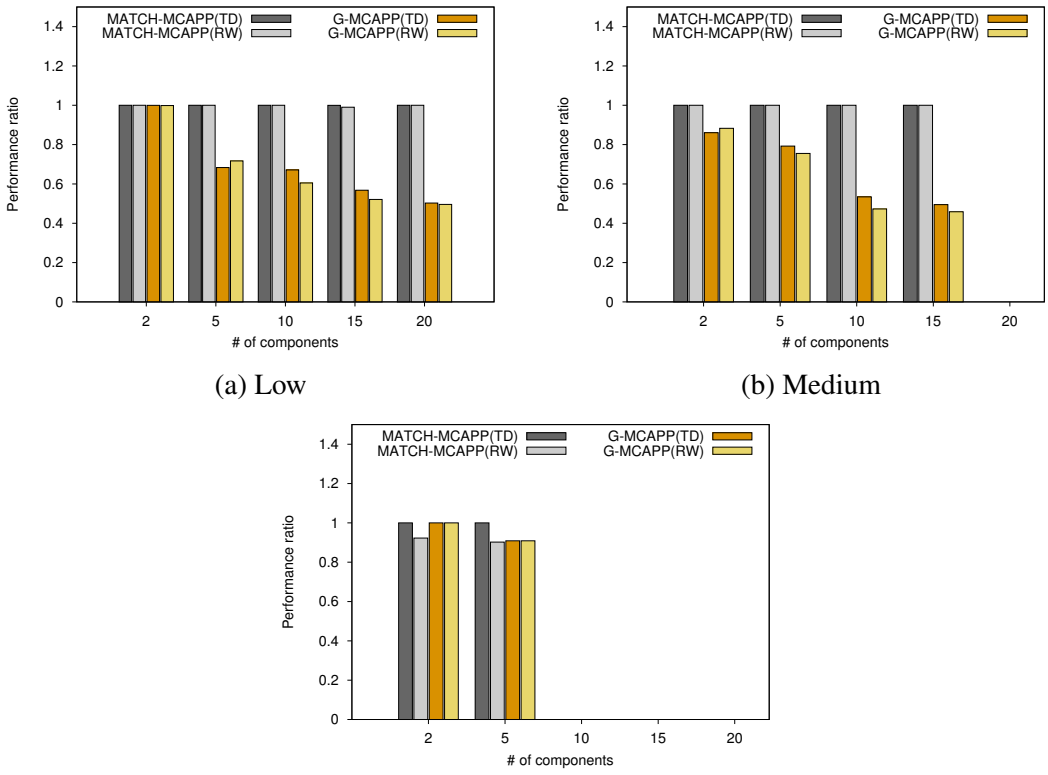
(c) High

Figure 3.7: Execution time (microseconds) vs. number of components for different level of inter-component communications.

(CPLEX was not able to determine the solutions for instances with 10, 15, and 20 components in feasible time, and thus, there are no bars in the plots for those cases)

the performance of our algorithms with that of the optimal solution. We vary the number of components from 2 to 20. CPLEX was not able to solve some of the instances in feasible time, and thus, we were not able to determine the performance ratios for MATCH-MCAPP and G-MCAPP algorithms. In those cases we do not display the bars in the corresponding plots.

In Figure 3.7, we plot the average execution time per time slot obtained by MATCH-MCAPP, G-MCAPP and CPLEX. The average execution time of the algorithms increases with the number of components. For example for instances with low inter-component communication and $n = 2$, the average execution time of MATCH-MCAPP is about 0.01 milliseconds under the trace-driven model, while for $n = 20$, it is about 0.1 milliseconds.



(a) Low

(b) Medium

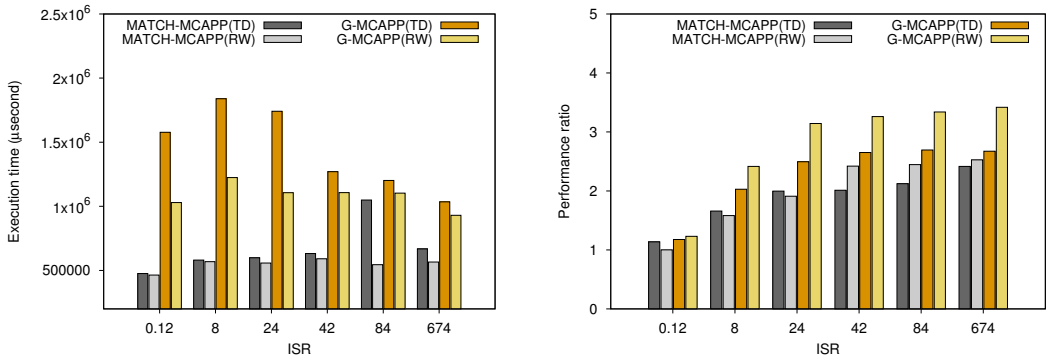
(c) High

Figure 3.8: Performance ratio vs. number of component for different level of inter-component communications.

(CPLEX was not able to determine the solutions for instances with 10, 15, and 20 components in feasible time, and thus, there are no bars in the plots for those cases)

Also, we observe that the execution time of G-MCAPP is much lower than the execution time of MATCH-MCAPP.

In Figure 3.8, we plot the performance ratio obtained by MATCH-MCAPP and G-MCAPP algorithms under the both mobility models. We observe that for instances with low or medium inter-component communication (Figure 3.8a, Figure 3.8b), MATCH-MCAPP outperforms G-MCAPP. Also, the performance ratio obtained by MATCH-MCAPP is very close to 1. For these instances, the performance ratio obtained by G-MCAPP decreases with the number of components. The reason is that, as the number of components increases, a smaller number of servers are available after each iteration of G-MCAPP. Therefore, the algorithm explores a smaller number of possible placements for each com-



(a) Average execution time (microseconds) per time slot.

(b) Performance ratio with respect to MATCH.

Figure 3.9: The effect of ISR (large-scale instances).

ponent.

Large-scale problem instances. We investigate the performance of the proposed algorithms for large-scale problem instances under the two mobility models. We consider large instances with a fixed number of components and servers ($n = 100$, $m = 200$) and fixed number of time slots ($T = 10$), and several values for ISR, ranging from 0.12 to 674. These instances with large number of components and servers are not expected to be encountered in practice, but we still consider them here to investigate the scalability of the algorithms. Since CPLEX is not feasible to use for solving such large instances, we will not compare the performance of our algorithms against the performance of the optimal solution obtained by CPLEX. Instead, we will compare the performance of our algorithm against that of MATCH (described in Section 3.5.1). In order to do this, we redefine the performance ratio as the ratio of the total cost obtained by MATCH and the total cost obtained by our proposed algorithms.

In Figure 3.9a, we plot the average execution time per time slot obtained by MATCH-MCAPP and G-MCAPP. As expected, the average execution time per time slot of MATCH-MCAPP and G-MCAPP is not very sensitive to the value of ISR. The execution time of MATCH-MCAPP increases slowly with ISR. This is because more swaps may be performed in the local search when the inter-component communication increases. However,

we observe that the execution time of G-MCAPP is not sensitive to the value of ISR. This is because the number of operations in the algorithm does not depend on the value of ISR. We also observe that, for large scale instances, the execution time of G-MCAPP is about two times greater than the execution time of MATCH-MCAPP; but it is still in a reasonable range compared to the duration of each time slot.

In Figure 3.9b, we plot the performance ratio of MATCH-MCAPP and G-MCAPP under the two mobility models. We observe that both algorithms obtain better solutions compared to the MATCH algorithm. MATCH-MCAPP obtains solutions with a total cost around 45% lower than MATCH for most cases. For small values of ISR, since there is almost no communication among the components, MATCH-MCAPP behaves similarly to the Hungarian algorithm, that is, the local search step is not actually able to improve the solution beyond that obtained by matching. By increasing the value of ISR, the performance ratio of MATCH-MCAPP increases. This means that, the local search improves the solution obtained by the Hungarian algorithm. Furthermore, we observe that the performance of G-MCAPP is much better than MATCH-MCAPP. In most cases, the total cost obtained by G-MCAPP is around 60% less than total cost obtained by MATCH algorithm. This means that G-MCAPP obtains a better performance for large scale problem instances while it has a reasonable execution time.

According to the experimental results, MATCH-MCAPP and G-MCAPP obtain solutions that are very close to the optimal and require very low execution time per slot for reasonably large instances. For the average size instances, the ones we expect to encounter in practice, the proposed algorithms perform very well with respect to both the quality of the solutions and the execution time per time slot. Also, the performance of both algorithms is consistent under both the trace-driven mobility data set and the random walk model which indicates that the proposed algorithms are relatively robust to the mobility behavior of the users. MATCH-MCAPP is more suitable for MEC systems with a relatively low number of servers and components, and applications with less intensive communica-

tion among components. On the other hand, G-MCAPP is more suitable for MEC systems with a large number of servers and components, as well as for applications with intensive communication among components.

3.6 Conclusion

We addressed the problem of placement of multi-component applications in MEC systems. We formulated the problem as a Mixed Integer Non-Linear Program (MINLP) and developed two efficient algorithms for solving it. We performed extensive experiments to investigate the performance of the proposed algorithms. The results of these experiments indicated that the proposed algorithms obtain very good performance and require very low execution time, making them very suitable for deployment on MEC systems.

CHAPTER 4

MECHANISMS FOR RESOURCE ALLOCATION AND PRICING IN MOBILE EDGE COMPUTING SYSTEMS

4.1 Introduction

In Mobile Cloud Computing (MCC), centralized cloud servers are employed as powerful resources for executing computational tasks of mobile applications [2]. These systems have been able to mitigate some of the existing limitations and challenges in mobile devices such as storage capacity, computation power, and battery life. The long distance between the centralized servers and end users results in high response latency which makes MCC systems unsuitable for many applications with low latency requirements. In recent years, Mobile Edge Computing (MEC) has been introduced to mitigate the existing challenges in MCC. In a MEC system, data, computation, and storage are migrated from mobile devices to the servers located at the edge of the network [99]. Despite the fact that MEC systems are in their early stages of development, a wide range of applications is expected to benefit from this technology. Autonomous driving [23, 100], healthcare [101, 102], and entertainment are just few examples of such applications. Compared to cloud data centers, edge systems have much more limited resources leading to increased competition among users who desire to acquire high quality services. Due to this challenge, the efficiency of MEC systems depends heavily on the utilized resource allocation mechanisms. Any inefficiencies in resource allocation might lead to low Quality of Service (QoS), high energy consumption, and increased operating costs.

Despite all the efforts focused on designing efficient resource allocation algorithms in MEC systems, monetization of services, that is, developing incentive schemes for mobile users and edge providers, is still a significant challenge in the development of MEC systems. A report by National Science Foundation (NSF) on grand challenges in edge computing [6], identified incentives and monetization as one of the five grand challenges in the development of edge computing systems. Decentralized distribution of MEC servers,

heterogeneity of resource requirements, and the competition between users to acquire high quality services make the resource allocation and pricing in MEC systems a challenging problem. Many auction mechanisms developed for MCC systems are not directly applicable to MEC settings. In MEC systems, resources are distributed over multiple levels, users typically request bundles of multiple types of resources, and they have different valuations for the services provided from different network levels (i.e., cloud and edge).

In this chapter, we address the monetization challenge in MEC systems by developing auction-based mechanisms for resource allocation and pricing. We consider a telecom-centric MEC system, where the edge resources are located at the first level of aggregation in the network. Such type of telecom-centric MEC architecture received considerable attention in the literature and significant support from industry and ETSI [90]. Deploying servers at the edge of the network in a telecom-centric MEC system is expensive, and therefore, a limited number of such servers are made available to mobile users. The scarcity of resources at the edge of the network creates a competitive environment for the mobile users, and therefore, there is an urgent need to develop incentive-based resource allocation and pricing mechanisms for MEC.

4.1.1 Our contributions

We propose two resource allocation and pricing mechanisms for MEC systems with heterogeneous servers and resources of different types. We consider dynamic provisioning of computing resources. Depending on the user demand, the computing resources are provisioned as Virtual Machines (VMs) instances. First, we develop a greedy resource allocation and pricing mechanism for edge computing systems. The mechanism combines features from both position [103] and combinatorial auctions [104] and handles heterogeneous resource requests from mobile users and heterogeneous types of resources. It determines *envy-free allocations* (i.e., allocations in which no user can improve her utility by exchanging bids with any user with the same request for resources) and prices that lead to close to optimal social welfare for the users. In addition, the mechanism requires very small

execution time even for very large problem instances with hundreds of users. The second algorithm, is a linear programming (LP) based approximation mechanism that does not guarantee envy-freeness, but it provides a solution that is guaranteed to be within a given distance from the optimal solution. We evaluate the performance of the proposed mechanisms through an extensive experimental analysis. For small-size instances, we compare the solutions obtained by our proposed mechanisms with the optimal solutions obtained by solving the Mixed-Integer Linear Programming (MILP) model of the problem using the CPLEX software. Since for large-size instances obtaining the optimal solution in a reasonable amount of time is not feasible, we compare the performance of the proposed mechanisms against each other. We also investigate the impact of dynamic provisioning by comparing the performance of the system under both static and dynamic provisioning.

4.1.2 Organization

The rest of the chapter is organized as follows. In Section 4.2, we formulate the problem of allocation and pricing. In Section 4.3, we introduce the proposed envy-free allocation and pricing mechanism. In Section 4.4, we introduce the proposed approximation mechanism. In Section 4.5, we describe the experimental setup and discuss the experimental results. In Section 4.6, we conclude the chapter and suggest possible directions for future work.

4.2 Edge resource allocation and pricing problem

We address the *Edge Resource Allocation Problem* (ERAP) for an edge computing system with one provider that offers computing resources in the form of Virtual Machines (VMs). In this network, servers are located either at the edge or on the clouds. Since the edge servers are closer to the users, they prefer to run their applications on the edge servers to obtain better performance. In fact, due to low latency requirements of users' applications, it is always more desirable for them to run their applications on edge servers that guarantee lower latency. However, the capacity of edge resources is more restricted than that of the

Table 4.1: Types of VM instances used in the experiments.

Type	vCPU	Memory (GB)	SSD Storage (GB)
medium	1	3.75 (1 unit)	1×4 (1 unit)
Large	2	7.5 (2 units)	1×32 (8 units)
Xlarge	4	15 (4 units)	2×40 (20 units)
2xlarge	8	30 (8 units)	2×80 (40 units)

cloud servers. Therefore, users have to compete to obtain computing resources on the edge servers. The provider uses an auction model to provide VM instances, where the price of resources is not pre-defined, but determined by employing a mechanism. The system allows dynamic provisioning of VMs, and does not require pre-provisioning the VMs. Therefore, the provider can fulfill dynamic market demands efficiently.

In this network, there are n users who are competing for resources situated at two levels, edge ($l = 1$), and cloud ($l = 2$), where l denotes the level. In both edge and cloud servers, m types of VM instances are available to serve users, where each type of VM instance is characterized by d types of resources. A VM instance of type j provides q_{jk} units of resource of type k. Here, we consider the aggregated resource capacity at each level, that is, the total capacity at the edge or cloud level. The total amount of resource of type k available at level l is denoted by C_{kl}. As an example, let us consider Amazon's Elastic Compute Cloud (EC2) M3 family of VM instances (see Table 4.1). In these instances, there are three types of resources ($d = 3$): vCPU ($k = 1$), memory ($k = 2$), and storage ($k = 3$). A 'large' VM instance of type $j = 2$ consists of 2 units of vCPU, 2 units of memory, and 8 units of storage. That is, the instance of type $j = 2$ is characterized by $q_{21} = 2$, $q_{22} = 2$, and $q_{23} = 8$.

User i submits a request for a bundle of VM instances which is denoted by $\theta_i = (b_i; \{r_{ij}\}) = (b_i; r_{i1}, \ldots, r_{im})$, where b_i is the bid and r_{ij} is the number of VM instances of type j requested by user i. The main reason for considering the requests as bundles of heterogeneous VMs is that users often require to execute tasks with different functionalities and roles on a set of VMs of different types [59]. It is safe to assume that the total request of a user for each type of resources is less than the available capacity for that type. The

requests are submitted to a mechanism that determines the provisioning and allocation of VM instances to users and the price they have to pay for their allocations.

A resource allocation and pricing mechanism for the above setting can be viewed as a hybrid of a multi-unit combinatorial auction [104] and a position auction [103]. It is a position auction because, (i) the auctioneer never allocates the bundle request of a user to more than one level, and (ii) users have different preferences for each level (position) of resources, with resources at the edge being more preferred. To characterize the user's preference for the edge and cloud level, we define α_l as the *preference factor* for level l which is determined based on the average distance between users and resources at level l (obviously, $\alpha_1 \geq \alpha_2$). Since the latency is determined by the distance between users and servers, the preference factor captures the service latency as well. Furthermore, since there are several VM instances of the same type in each level and users bid for a bundle of resources, the problem can also be viewed as a multi-unit combinatorial auction.

We assume that users are not interested in partially executing their tasks. Also, they do not obtain any value by partially executing on edge and partially on the cloud level (due to the undesirable variable latencies that may occur by executing their tasks on different levels of the network). In other words, we assume that users are single minded [105], that is, a user is only interested in a single bundle. The user values this bundle and any superset of it at the same amount, and all other bundles at zero. If the allocation function allocates the requested bundle of a user and any superset of it in the first level, then the user values the bundle and any superset of that bundle at the same amount. If the allocation function allocates the requested bundle of a user and any superset of it in the second level, then the user values it at the same amount (but this value is less than the value in the first case); otherwise, the user values the allocation at zero.

Let $a_i = \{a_i^1, a_i^2\}$ be the allocation for user i determined by the allocation mechanism, where $a_i^l = (a_{i1}^l, \ldots, a_{im}^l)$ is the allocation of resources at level l for user i, and a_{ij}^l is the number of VM instances of type j allocated to this user on level l. Thus, the valuation

function for user i is as follows,

$$v_i(a_i) = \begin{cases} \alpha_1 b_i & \text{if } r_i \preceq a_i^1 \\ \alpha_2 b_i & \text{if } r_i \preceq a_i^2 \text{ and } r_i \not\preceq a_i^1 \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where $r_i \preceq a_i^l$, if $r_{ij} \leq a_{ij}^l$, $\forall j \in \{1, \ldots, m\}$. The valuation represents the maximum price that a user is willing to pay for the requested bundle at each level. Since $\alpha_1 > \alpha_2$, users prefer the allocation at the edge level instead of that at the cloud level.

In our system, the goal is to maximize the social welfare. The social welfare, V, is the sum of users' valuations, $V = \sum_{l=1}^{2} \sum_{i=1}^{n} \alpha_l \cdot b_i \cdot x_{il}$, where x_{il} is a binary decision variable that is 1, if the bundle requested by user i is allocated at level l; and 0, otherwise. Maximizing social welfare helps resource providers increase their revenue, due to the fact that the system allocates the available VMs to the users who value them the most [105]. Table 4.2 shows the notation we use in the formulation of the problem. The ERAP can be formulated as a mixed-integer linear program (MILP) as follows,

ERAP-MILP:

$$\text{maximize} \quad V = \sum_{i=1}^{n} \sum_{l=1}^{2} \alpha_l \cdot b_i \cdot x_{il} \tag{4.2}$$

subject to:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \cdot q_{jk} \cdot x_{il} \leq C_{kl} \quad \forall k, \forall l \tag{4.3}$$

$$\sum_{l=1}^{2} x_{il} \leq 1 \quad \forall i \tag{4.4}$$

$$x_{il} \in \{0, 1\} \quad \forall i, \forall l \tag{4.5}$$

The objective function (4.2) is to maximize the *welfare*, V, where the welfare is the

Table 4.2: Notation

Notation	Description
n	Number of of users.
m	Types of VM instances.
d	Number of resource types.
α_l	Preference factor for level l.
b_i	Bid of user i.
r_{ij}	Number of VM instances of type j requested by user i.
C_{kl}	Capacity of resource of type k available at level l.
q_{jk}	Amount of resource of type k for a VM of type j.
w_k	Weight of resource of type k.
π_i	Base price for user i.
p_i	Payment of user i.

sum of users' valuations. Constraints (4.3) ensure that the total allocated requests of each type of resources to each level does not exceed the capacity of that level. Constraints (4.4) guarantee that the request of user i is not allocated to more than one level. Finally, constraints (4.5) guarantee the integrality of the decision variables.

A pricing mechanism $M = (A, P)$ consists of an allocation function $A = (a_1, \ldots, a_n)$ and a payment rule $P = (p_1, \ldots, p_n)$. The mechanism determines the allocation of requested bundles and the payments based on the users' bids and the available resources in the system. It determines a base price for each user i, denoted by π_i. Based on the base price, the price of a unit of resource of type k for user i is defined as $\pi_i \cdot w_k$, where w_k is the weight of the resource of type k. Here, w_k is used to differentiate the values of a unit of different types of resources. Therefore, the price of a VM instance of type j for user i is $\sum_{k=1}^{d} \pi_i \cdot w_k \cdot q_{jk}$.

We define the payment that user i has to pay to the provider as,

$$p_i = \sum_{j=1}^{m} \sum_{k=1}^{d} r_{ij} \cdot q_{jk} \cdot \pi_i \cdot w_k \tag{4.6}$$

We assume that users have quasi-linear utilities (i.e., $u_i = v_i - p_i$) and are rational, in the sense that their goal is to maximize their utility.

4.2.1 Complexity of ERAP

Here, we prove that the decision version (ERAP-D) of ERAP is NP-complete. This implies that ERAP is NP-hard. An instance of ERAP-D consists of: a set of users, users' requests $\theta_i = \{b_i, \{r_{ij}\}\}$, $i = 1, \ldots, n$, k computational resources of given capacities $\{C_{kl}\}$, at both levels (i.e., edge and cloud), preference factor α_l at level l, and a bound $B \in \mathbb{R}^+$. The decision problem is to determine whether there exists an allocation of users such that the social welfare of the assignment (Equation (4.2)) is at least B, and no capacity constraint is violated.

Theorem 2. *ERAP-D is NP-complete.*

Proof. We prove that ERAP-D is NP-complete by showing that: (i) ERAP-D belongs to NP, and, (ii) a well known NP-complete problem is reduced to ERAP-D in polynomial time. For any arbitrary allocation of requests, the feasibility check of the capacity constraints and computing the social welfare (Equation (4.2)) could be performed in polynomial time, which implies that ERAP-D is in NP.

For the second condition, we consider a special case of the Multiple Multi-dimensional Knapsack Problem, MMKP, with a fixed number of knapsacks (two knapsacks, one for each of the two network levels) and a fixed number of dimensions. For simplicity, we denote this special case by 2D-MMKP. The multi-dimensional knapsack problem for any fixed number of dimensions is NP-complete [106]. Also, it is well known that the multiple knapsack problem is NP-complete even when the number of knapsacks is two [107]. Therefore, 2D-MMKP is NP-complete.

Now, we show that 2D-MMKP can be reduced to ERAP-D in polynomial time. Let us define an arbitrary instance of 2D-MMKP with n' items. Each item i has a value v_i and a weight ω_{ij} in dimension j. The capacity of knapsack l in dimension k is C'_{kl}. The decision problem is to determine whether there is an assignment of items to each of the two knapsacks, such that the total value of the items is at least L, while the total weight on each dimension of each knapsack does not exceed the capacity.

We construct an instance of ERAP-D, called Q, based on an arbitrary instance of 2D-MMKP, called Q', such that the total social welfare of Q is at least B, if and only if, the total value of Q' is at least L. Instance Q consists of n items such that $n = n'$. The capacity of resources at level l for resources of type k is defined as $C_{kl} = C'_{kl}$. Let us assume $\alpha_1 = \alpha_2 = 1$. The bid of user i is defined as $b_i = v_i$. Also, we let $r_{ij} = \omega_{ij}$, and $q_{jk} = 1$. We claim that Q has a feasible assignment with social welfare greater than or equal to B, if and only if, Q' has a feasible assignment with the total value greater than or equal to L. Since we assume $q_{jk} = 1$, any feasible solution for Q is also a feasible solution for Q'. Also, since we assume that $\alpha_1 = \alpha_2 = 1$, the social welfare of the solution for Q is equivalent to the total value obtained for Q'. Conversely, suppose that there is an assignment in Q' with total value greater than or equal to L. We assign the corresponding users to the edge/cloud levels. Clearly, any feasible solution for Q' is also a feasible solution for Q and the total value of the solution for Q' is equivalent to the social welfare of the solution for Q. □

We proved that ERAP is NP-hard, that is, it is not possible to find an optimal solution in polynomial time, unless P = NP. On the other hand, incorporating pricing into a resource allocation model exacerbates the complexity of the problem. Given the fact that a resource allocation mechanism has to be used efficiently for instances with a large number of requests and in order to solve the problem in reasonable time, we have to leverage heuristic methods instead of using commercial solvers. In this chapter, we design two mechanisms for edge resource allocation and pricing problem, G-ERAP and APX-ERAP. G-ERAP is a greedy mechanism for resource allocation and pricing in edge systems, that is individually-rational and envy-free. APX-ERAP is an LP-based $\left(\frac{1}{2d+1}\right)$-approximation mechanism for resource allocation and pricing in edge systems. In the following sections, we describe the proposed mechanisms and discuss their properties.

4.3 Envy-free resource allocation and pricing mechanism

In this section, we design a greedy mechanism for resource allocation and pricing, called G-ERAP. G-ERAP, which is given in Algorithm 3, considers uniform base prices for each type of VM in the same level. In other words, the price per unit of each VM type is the same for winners at the same level of the system, but winners at different levels should pay differently for the same type of VM instance. The mechanism is invoked periodically at time intervals of a specified duration. The allocation and price determined by the mechanism is valid for the current time interval.

The mechanism collects the requests of users and prioritize them based on their *bid densities*, that is, the average bid per unit of resource. The bid density of user i is defined as,

$$B_i = \frac{b_i}{\sum_{j=1}^{m} \sum_{k=1}^{d} r_{ij} \cdot q_{jk} \cdot w_k} \tag{4.7}$$

The mechanism starts the allocation from the edge level for the requests with relatively high density bid. Once it reaches to a request which is not fitted to the edge level (according to the size of request), the mechanism starts the allocation on the cloud level.

The input to G-ERAP consists of a vector of requests θ_i, $i = 1, \ldots, n$ from users, and a vector of resource capacities, C. G-ERAP determines how these resources are allocated to users. The output of the mechanism consists of the allocation matrix X, where $X = [x_{il}]$, $i = 1, \ldots, n$, and $l = 1, 2$, the social welfare V, and the payment vector $P = \{p_i\}$. First, the mechanism determines the bid density for each user (line 3). Then, users are sorted in non-increasing order of their bid densities (line 4), and VM instances are allocated to users starting from the first level (i.e., the edge level). For the current user, the mechanism checks if there are enough resources at the current level. If so, the requested VM instances are allocated to the user, and the social welfare and the capacity are updated (lines 7-11). If there are not enough resources to allocate the requested bundle at the first level, then

Algorithm 3 G-ERAP Mechanism

Input: Vector of requests: $\theta_i = (b_i; \{r_{ij}\})$
Vector of resources' capacities: $C = \{C_{lk}\}$
Output: Allocation matrix: $X = \{x_{il}\}$
Total welfare: V
Payment vector: $P = \{p_i\}$

$V \leftarrow 0$
$X \leftarrow 0$
$B_i \leftarrow \frac{b_i}{\sum_{j=1}^{m}\sum_{k=1}^{d} r_{ij} \cdot q_{jk} \cdot w_k} \quad \forall i \in \{1, \ldots, n\}$
Sort users in non-increasing order of their B_i
$u \leftarrow 0$, $u' \leftarrow 0$, $l \leftarrow 1$, $i \leftarrow 1$
while $i \leq n$ **do**
    if $C_{lk} \geq \sum_{j=1}^{m} r_{ij} \cdot q_{jk} \quad \forall k \in \{1, \ldots, d\}$ **then**
        $C_{lk} \leftarrow C_{lk} - \sum_{j=1}^{m} r_{ij} \cdot q_{jk} \quad \forall k \in \{1, \ldots, d\}$
        $V \leftarrow V + \alpha_l \cdot b_i$
        $x_{il} \leftarrow 1$
        $i \leftarrow i + 1$
    else
        if $l = 1$ **then**
            $u \leftarrow i - 1$
            $l \leftarrow l + 1$
        else
            $u' \leftarrow i$
            break
        end if
    end if
end while
if $u' < n$ **then**
    $B^* \leftarrow B_{u'+1}$
else
    $B^* \leftarrow B_{u'} - \epsilon$
end if
for each $i \in \{1, \ldots, n\}$ **do**
    if $x_{i2} = 1$ **then**
        $\pi_i \leftarrow \alpha_2 \cdot B^* + \frac{(\alpha_1 - \alpha_2)}{2}(B_u + B_{u+1})$
    else
        if $x_{i1} = 1$ **then**
            $\pi_i \leftarrow \alpha_2 \cdot B^*$
        else
            $\pi_i \leftarrow 0$
        end if
    end if
    $p_i \leftarrow \sum_{j=1}^{m}\sum_{k=1}^{d} r_{ij} \cdot q_{jk} \cdot \pi_i \cdot w_k$
end for

the index of the level is increased by one (i.e., starting allocation of VM instances on the second level), and the index of the user is stored as the first allocated user in the second level. This user is denoted by u (lines 12-15). The allocation stops once it reaches a user (denoted by u') for which there are not enough resources to satisfy the requested bundle in the second level (lines 16-18).

Next, G-ERAP determines the payments for each user (lines 19-31). According to the mechanism, user u is the last user in the sorted order which is allocated to the first level. Therefore, user $u+1$ is the first user in the list that is to be allocated on the second level. The base price for each user i allocated at edge level is determined as follows,

$$\pi_i = \alpha_2 B^* + \frac{(\alpha_1 - \alpha_2)}{2}(B_u + B_{u+1}) \tag{4.8}$$

The base price for each user i allocated at cloud level is determined as follows,

$$\pi_i = \alpha_2 B^* \tag{4.9}$$

where

$$B^* = \begin{cases} B_{u'+1} & \text{if } u' < n \\ B_{u'} - \epsilon & \text{otherwise} \end{cases} \tag{4.10}$$

and u' is the last allocated user at the cloud level ($l = 2$). If some users in the sorted list cannot be allocated at the cloud level, then B^* is defined based on the bid density of the first unallocated user in the sorted list. But, if all the remaining users from the first level are allocated at the second level, then B^* has a value a bit less than the weighted bid of the last allocated user (i.e., $B_{u'} - \epsilon$, where ϵ is a very small positive real number). Note that if a user is not allocated the requested bundle, then the base payment is 0 (line 30). After determining the base payments, G-ERAP determines the payment of users according to Equation 4.6 (line 31).

4.3.1 Properties of G-ERAP

G-ERAP combines features from both position [103] and combinatorial auctions [104] and is not truthful. Here, we do not target truthfulness for our mechanism but instead, we guarantee that it produces envy-free allocations. Truthful mechanisms are desirable from the user perspective because truth-telling is the dominant strategy. Thus, users know that by submitting their true valuation they maximize their utilities independent of the bids of the other users. However, truthful mechanisms are not necessarily the most desirable mechanisms from the auctioneer's perspective. For example, search engines, such as Google prefer to employ the Generalized Second Price (GSP) mechanism to sell online advertising [72]. GSP is an envy-free mechanism and is not truthful, but it generates more revenue than truthful mechanisms such as Vickrey-Clarke-Groves (VCG). In addition, GSP has a simple design and is very fast.

Envy freeness allows us to design a computationally efficient mechanism suitable for providing services in a two-level edge computing systems, where the edge resources are at the first level and the cloud resources at the second. In the following, we prove that G-ERAP mechanism is *individually-rational* and produces *envy-free allocations* which are two important and desirable properties of a mechanism [70]. The first property guarantees that users are willing to participate in the mechanism, while the second guarantees that when the auction is terminated, no user would be happier with the outcome of another user.

Definition 1 (**Individual Rationality**). *A mechanism is individually rational if for each user i bidding her true valuation for the bundle, $v_i - p_i \geq 0$ (i.e., a user reporting her true valuation for the bundle never incurs a loss).*

Lemma 3. *The base price of a user allocated at the edge level, satisfies $\pi_i \leq \alpha_1 B_u$, where u is the last user allocated at the edge level ($l = 1$).*

Proof. The base price of a user i allocated at the edge level is given by:

$$\begin{aligned}\pi_i &= \alpha_2 B^* + \frac{(\alpha_1 - \alpha_2)}{2}(B_u + B_{u+1}) \\ &\leq \alpha_2 B^* + \frac{(\alpha_1 - \alpha_2)}{2} 2B_u \leq \alpha_2 B^* + (\alpha_1 - \alpha_2) B_u \\ &\leq \alpha_1 B_u + \alpha_2 (B^* - B_u)\end{aligned}$$

Because user u is the last user allocated at the edge level, $B^* \leq B_u$, and thus, $\pi_i \leq \alpha_1 B_u$. □

Theorem 4. *G-ERAP is individually-rational.*

Proof. To prove this theorem, we need to show that the utility of a user reporting (bidding) her true valuation for the requested bundle is non-negative. There are three possible outcomes for a participant user denoted by i:

Case I. User i is allocated to the edge level ($B_i \geq B_u$).

$$v_i - p_i = \alpha_1 B_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk} - \pi_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk}$$

$$\geq \alpha_1 B_u \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk} - \pi_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk}$$

Using the result of Lemma 3, we have, $v_i - p_i \geq 0$.

Case II. User i is allocated to the cloud level ($B_u \geq B_i \geq B^*$).

$$v_i - p_i = \alpha_2 B_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk} - \pi_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk}$$

$$\geq \alpha_2 B^* \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk} - \pi_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk}$$

According to Equation (4.9), $\pi_i = \alpha_2 B^*$. Thus, $v_i - p_i \geq 0$.

Case III. User i loses the auction ($B^* \geq B_i$). According to Equation (4.1), $v_i = 0$. Because user i is not allocated any bundle of VMs, $p_i = 0$. Thus, $v_i - p_i = 0$. □

Definition 2 (**Envy-Freeness**). *An allocation is envy-free if no user can improve her utility by exchanging bids with any user with the same request for VM instances.*

Theorem 5. *G-ERAP produces envy-free allocations.*

Proof. Let us assume that user i is allocated to the first level (edge), user i' is allocated to the second level (cloud), and user i'' loses the auction, and all of them have the same request for VM instances. It is obvious that a user cannot improve her utility by exchanging bids with another user with an identical request for VM instances that is allocated to the same level. Therefore, we only need to show three cases.

Case I. Users i and i' cannot improve their utilities by exchanging their bids. By exchanging their bids, user i' is allocated to the first level (edge) and user i is allocated to the second level (cloud). In this case, we need to show that,

$$v_i(a_i) - p_i \geq v_i(a_{i'}) - p_{i'} \tag{4.11}$$

and,

$$v_{i'}(a_{i'}) - p_{i'} \geq v_{i'}(a_i) - p_i \tag{4.12}$$

These equations are equivalent to,

$$\alpha_1 b_i - \pi_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk} \geq \alpha_2 b_i - \pi_{i'} \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk} \tag{4.13}$$

and,

$$\alpha_2 b_{i'} - \pi_{i'} \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{i'j} \cdot q_{jk} \geq$$
$$\alpha_1 b_{i'} - \pi_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{i'j} \cdot q_{jk} \tag{4.14}$$

According to the definition of B_i (Equation (4.7)), we rewrite Equation (4.13) as,

$$\alpha_1 B_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk} - \pi_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk} \geq$$
$$\alpha_2 B_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk} - \pi_{i'} \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk}$$
$$\implies \alpha_1 B_i - \pi_i \geq \alpha_2 B_i - \pi_{i'} \tag{4.15}$$

Similarly, Equation 4.14 is equivalent to,

$$\alpha_2 B_{i'} - \pi_{i'} \geq \alpha_1 B_{i'} - \pi_i \tag{4.16}$$

Based on Equations (4.15) and (4.16), we only need to show that,

$$B_{i'}(\alpha_1 - \alpha_2) \leq \pi_i - \pi_{i'} \leq B_i(\alpha_1 - \alpha_2) \tag{4.17}$$

Using the definition of B_u and B^* from Equations (4.8) and (4.10) we obtain, $B^* \leq B_{i'} \leq B_{u+1} \leq B_u \leq B_i$. Also, based on Equations (4.8) and (4.9),

$$\pi_i - \pi_{i'} = \frac{(\alpha_1 - \alpha_2)}{2}(B_u + B_{u+1}) \tag{4.18}$$

Thus,

$$\frac{(\alpha_1 - \alpha_2)}{2}(B_{i'} + B_{i'}) \leq \pi_i - \pi_{i'} \leq \frac{(\alpha_1 - \alpha_2)}{2}(B_i + B_i)$$

$$(\alpha_1 - \alpha_2)B_{i'} \leq \pi_i - \pi_{i'} \leq (\alpha_1 - \alpha_2)B_i \tag{4.19}$$

Therefore, for this case, G-ERAP produces envy-free allocations.

Case II. Users i and i'' cannot improve their utilities by exchanging their bids. In this case, user i loses the auction and user i'' is allocated to the first level (edge). It is obvious that the utility of user i does not improve, thus, we only need to show that the utility of user i'' does not improve. In this case, we need to show that, $v_{i''}(a_{i''}) - p_{i''} \geq v_{i''}(a_i) - p_i$. This is equivalent to show that,

$$0 \geq \alpha_1 B_{i''} \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{i''j} \cdot q_{jk} - \pi_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{i''j} \cdot q_{jk} \tag{4.20}$$

which is equivalent to show that,

$$0 \geq \alpha_1 B_{i''} - \pi_i \tag{4.21}$$

Also, based on Equation (4.8),

$$\pi_i \geq \alpha_2 B^* + \frac{(\alpha_1 - \alpha_2)}{2}(2B^*) \Longrightarrow \pi_i \geq \alpha_1 B^* \geq \alpha_1 B_{i''} \tag{4.22}$$

which satisfies Equation (4.21).

Case III. User i' and user i'' cannot improve their utility by exchanging their bids. By exchanging their bids, user i' loses the auction and user i'' is allocated to the second level (i.e., cloud). In this case, the utility of user i' does not improve, thus, we only need to show that the utility of user i'' does not improve. For this purpose, we need to show that, $v_{i''}(a_{i''}) - p_{i''} \geq v_{i''}(a_{i'}) - p_{i'}$. This is equivalent to show that,

$$0 \geq \alpha_2 B_{i''} \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{i''j} \cdot q_{jk} - \pi_{i'} \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{i''j} \cdot q_{jk} \tag{4.23}$$

which is equivalent to show that,

$$0 \geq \alpha_2 B_{i''} - \pi_{i'} \tag{4.24}$$

According to Equation (4.10), $B^* \geq B_{i''}$. Therefore, based on Equation (4.9), $\pi_{i'} = \alpha_2 B^* \geq \alpha_2 B_{i''}$ which satisfies Equation (4.24). □

Now, we investigate the time complexity of G-ERAP. The time complexity of computing the bid densities (line 3) is $O(nmd)$. Sorting the users (line 4) takes $O(n \log n)$. The main part of the algorithm consists of the loop in lines 7-19 which executes n times. In each iteration, the available capacity for each type of resources is compared with the total request of the user for that resource type (lines 8-9), which takes $O(md)$. Therefore, the time complexity of G-ERAP is $O(n \log n + nmd)$.

4.4 LP-based approximation mechanism for resource allocation and pricing

In this section, we focus on designing a mechanism that provides guarantees with respect to how far from the optimal is the solution obtained by the mechanism. We develop an LP-based approximation mechanism, APX-ERAP, for solving the ERAP. APX-ERAP is an extension of a $(\frac{1}{d+1})$ - approximation algorithm [108] for the d-dimensional knapsack problem, MDKP. In fact, ERAP can be viewed as a weighted multi-dimensional multiple knapsack problem composed of two knapsacks (two levels of resources). Each knapsack has d dimensions (d types of resources) with a limited capacity, C_{lk}, for each dimension k. There are n items (n users) that are to be assigned to the knapsacks. The size of item i in dimension k is $\sum_{j=1}^{m} r_{ij} \cdot q_{jk}$, the profit of item i in knapsack l is $\alpha_l \cdot b_i$, and the objective is to maximize the total profit which is $\sum_{i=1}^{n} \alpha_l \cdot b_i \cdot x_{il}$.

The original algorithm [108] first solves the LP relaxation of MDKP which considers only one d-dimensional knapsack. The LP relaxation solution contains a set of completely assigned items, I, and a set of at most d fractionally assigned items, F. Then, the algorithm considers two feasible solutions for the knapsack: assigning all items in I to the knapsack

or assigning the most profitable item in F to the knapsack. The algorithm selects the solution that maximizes the profit. In APX-ERAP, we extend this idea to solve the ERAP, which considers two d-dimensional knapsacks. The LP relaxation solution of ERAP-MILP contains a set of completely assigned users at the edge level, I_1, a set of completely assigned users at the cloud level, I_2, a set of at most d fractionally assigned items at the edge level, F_1, and a set of at most d fractionally assigned items at the cloud level, F_2. Then, based on these sets, the algorithm determines the most valuable allocations for the edge level and the cloud level.

APX-ERAP is given in Algorithm 4. The input to APX-ERAP consists of a vector of requests from users, θ_i, and a vector of resource capacities, C. The output of the mechanism consists of the allocation matrix, $X = \{x_{il}\}$, the social welfare V, and the vector of payments of users, $P = \{p_i\}$. The mechanism first sorts the users in non-increasing order of their bid densities (lines 3-4). This order is used when the algorithm determines the payment of users. Then, the algorithm solves the LP relaxation of ERAP-MILP by calling LP-SOLVE(ERAP) and saves the solution in matrix $\bar{X} = \{\bar{x_{il}}\}$ (line 5). Then, it defines $F_l = \{i : 0 < \bar{x_{il}} < 1\}$ as the set of users that have fractional allocations at level l, $I_l\{i \mid \bar{x_{il}} = 1\}$, as the set of users that are completely allocated on level l, and ν_l as the total bids of users who are completely allocated on level l (lines 6-7). One feasible solution for the problem is to allocate users in I_1 at the edge level and users in I_2 at the cloud level. An alternative solution is to select two users from $F_1 \cup F_2$ and allocate one of them at the edge level and the other one at the cloud level. This solution is also feasible, because we assume that the size of each user's request is less than the capacity. Based on these facts, the mechanism determines the allocation at the edge level and the cloud level.

To obtain the maximum social welfare, the mechanism compares the two highest bids of users in $F_1 \cup F_2$ with the sum of the bids of users in I_1, ν_1, and the sum of the bids of users in I_2, ν_2. For this purpose, it determines the index of the two highest bidders in $F_1 \cup F_2$, denoted by max_1 and max_2 (lines 9-10). Then, it considers four possible cases

Algorithm 4 APX-ERAP

Input: Vector of requests: $\theta_i = (b_i; \{r_{ij}\})$
Vector of resources' capacities: $C = \{C_{lk}\}$
Output: Allocation matrix: $X = \{x_{il}\}$
Total welfare: V
Payment vector: $P = \{p_i\}$

$V \leftarrow 0$
$X \leftarrow \{0\}$
$B_i \leftarrow \frac{b_i}{\sum_{j=1}^{m}\sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk}} \quad \forall i \in \{1, \ldots, n\}$
Sort users in non-increasing order of B_i
$\bar{X} \leftarrow$ LP-SOLVE(ERAP)
$F_l \leftarrow \{i : \ 0 < \bar{x}_{il} < 1\}, \ l = 1, 2$
$I_l \leftarrow \{i : \ \bar{x}_{il} = 1\}, \ l = 1, 2$
$\nu_l \leftarrow \sum_{i \in I_l} b_i, \ l = 1, 2$
$max_1 \leftarrow \text{argmax}_{i \in F_1 \cup F_2} \, b_i$
$max_2 \leftarrow \text{argmax}_{i \in F_1 \cup F_2 \setminus \{max_1\}} \, b_i$
case1: $\nu_1 < b_{max_1}$ **and** $\nu_2 < b_{max_2}$
    $V \leftarrow \alpha_1 \cdot b_{max_1} + \alpha_2 \cdot b_{max_2}$
    $x_{max_1 1} \leftarrow 1$
    $x_{max_2 2} \leftarrow 1$
case 2: $\nu_1 \geq b_{max_1}$ **and** $\nu_2 \geq b_{max_2}$
    $V \leftarrow \alpha_1 \cdot \nu_1 + \alpha_2 \cdot \nu_2$
    $x_{i1} \leftarrow 1 \quad \forall i \in I_1$
    $x_{i2} \leftarrow 1 \quad \forall i \in I_2$
case3: $\nu_1 < b_{max_1}$ **and** $\nu_2 \geq b_{max_2}$
    $V \leftarrow \alpha_1 \cdot b_{max_1} + \alpha_2 \cdot \nu_2$
    $x_{max_1 1} \leftarrow 1$
    $x_{i2} \leftarrow 1 \quad \forall i \in I_2$
case4: $\nu_1 \geq b_{max_1}$ **and** $\nu_2 < b_{max_1}$
    $V \leftarrow \alpha_1 \nu_1 + \alpha_2 \cdot b_{max_1}$
    $x_{i1} \leftarrow 1 \quad \forall i \in I_1$
    $x_{max_1 2} \leftarrow 1$
for each $i \in \{1, \ldots, n\}$ **do**
    if $x_{i1} = 1$ **then**
        $\pi_i \leftarrow \alpha_1 B_{i+1}$
    else
        if $x_{i2} = 1$ **then**
            $\pi_i \leftarrow \alpha_2 B_{i+1}$
        else
            $\pi_i \leftarrow 0$
        end if
    end if
    $p_i \leftarrow \sum_{j=1}^{m}\sum_{k=1}^{d} r_{ij} \cdot q_{jk} \cdot \pi_i \cdot w_k$
end for

for values of ν_1, ν_2, b_{max_1}, and b_{max_2} (lines 11-26). In each case, the mechanism allocates users in such a way that the social welfare is maximized. In the first case, since $\nu_1 < b_{max_1}$ and $\nu_2 < b_{max_2}$, the mechanism allocates the user with index max_1 at the edge level and user with index max_2 at the cloud level. In the second case, since $\nu_1 \geq b_{max_1}$ and $\nu_2 \geq b_{max_2}$, the mechanism allocates users in I_1 at the edge level and users in I_2 at the cloud level. In the third case, since $\nu_1 < b_{max_1}$ and $\nu_2 \geq b_{max_2}$, it allocates the user with index max_1 at the edge level and users in I_2 at the cloud level. Finally, in the last case, since $\nu_1 \geq b_{max_1}$ and $\nu_2 < b_{max_2}$, the mechanism assigns users in I_1 at the edge level and user with index max_1 at the cloud level.

Next, the APX-ERAP determines the payments for users (lines 27-35). The algorithm considers variable base prices in which winners of the same level may have different base prices. For user i allocated at level l, the base price is calculated as,

$$\pi_i = \alpha_l B_{i+1} \tag{4.25}$$

where B_{i+1} is the highest bid density of the user after user i in the non-increasing order of bids.

4.4.1 Properties of APX-ERAP

Here, we prove that APX-ERAP is a $\left(\frac{1}{2d+1}\right)$-approximation mechanism, where d is the number of types of physical resources. We also show that the mechanism is *individually-rational*.

Lemma 6. *In the LP relaxation of ERAP, where $0 \leq x_{il} \leq 1$, there are at most d users that are fractionally allocated at the edge level and at most $2d$ users that are fractionally allocated at the cloud level.*

Proof. Let $\bar{x} = \{\bar{x}_i^l\}$ be the solution obtained by LP-relaxation, and $F_l = \{i : 0 < \bar{x}_{il} < 1\}$ be the set of users that are fractionally allocated at level l. For user‘i that is fractionally allocated at the edge level ($i \in F_1$), Constraint (4.4) and Constraint ($0 \leq x_{il} \leq 1$) are re-

dundant. The reason is that, since $\alpha_1 > \alpha_2$, the only constraint that does not allow a higher fraction of allocation for user i at the edge level is the limited capacity (Constraint (4.3)). Since Constraint (4.3) is the only set of constraints in the LP relaxation model, the total number of constraints related to the edge level is d. Therefore, any basic feasible solution of this relaxation has at most d fractional variables at the edge level (i.e., $|F_1| \leq d$).

For the set of users that are fractionally allocated at the cloud level ($i \in F_2$), there are two possible subsets: (i) the set of users that are fractionally allocated at both edge level and cloud level (i.e., $F_1 \cap F_2$); (ii) the set of users that are fractionally allocated at the cloud level only (i.e., $F_2 \setminus F_1$). It is obvious that the number of users in the first set is not more than d. In the second set, the only constraint that does not allow a higher allocation is the capacity constraint (Constraint (4.3)). Thus, Constraint (4.4) and Constraint ($0 \leq x_{il} \leq 1$) are redundant for users in this set. Therefore, there are at most d fractional variables associated with the users in this set. Therefore, the total number of users fractionally allocated at the cloud level is at most $2d$. □

Theorem 7. *APX-ERAP is a* $\left(\frac{1}{2d+1}\right)$*-approximation mechanism for ERAP.*

Proof. Let X^* be the optimal allocation matrix and OPT be the total social welfare of the optimal solution. The solution to the LP relaxation is an upper bound on the optimal solution,

$$OPT \leq LP \leq \alpha_1 \cdot (\sum_{i \in F_1} b_i + \sum_{i \in I_1} b_i) + \alpha_2 \cdot (\sum_{i \in F_2} b_i + \sum_{i \in I_2} b_i) \tag{4.26}$$

For the four possible cases mentioned in Algorithm 4, we can easily show that,

$$V \geq \alpha_1 \cdot b_i + \alpha_2 \cdot b_j \quad \forall i \in F_1, j \in F_2 \tag{4.27}$$

According to Lemma 6, there are at most d items in F_1 and at most $2d$ items in F_2. There-

fore,

$$2d \cdot V \geq \alpha_1 \cdot \sum_{i \in F_1} b_i + \alpha_2 \cdot \sum_{i \in F_2} b_i \tag{4.28}$$

Also, $V \geq \alpha_1 \cdot \nu_1 + \alpha_2 \cdot \nu_2$. Thus,

$$(2d+1) \cdot V \geq \alpha_1 \sum_{i \in F_1} b_i + \alpha_2 \sum_{i \in F_2} b_i + \alpha_1 \nu_1 + \alpha_2 \nu_2 \geq OPT \tag{4.29}$$

Therefore, $V \geq \frac{OPT}{2d+1}$. □

Theorem 8. *APX-ERAP is individually-rational.*

Proof. To prove this theorem, we need to show that the utility of a user reporting her true valuation for the requested bundle is non-negative. There are two possible outcomes of each user i:

Case I. User i is allocated to level l.

$$v_i - p_i = \alpha_l B_i \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk} - \alpha_l B_{i+1} \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot r_{ij} \cdot q_{jk}$$

Therefore, since $B_i \geq B_{i+1}$, we have, $v_i - p_i \geq 0$.

Case II. User i loses the auction. According to Equation (4.1), $v_i = 0$. Because user i is not allocated any bundle of VMs, $p_i = 0$. Thus, $v_i - p_i = 0$. □

Now, we investigate the time complexity of APX-ERAP. The most time consuming part of the algorithm is solving the LP relaxation of ERAP-MILP, which takes polynomial time [109]. The other parts of the algorithm also have polynomial time complexity. Therefore, the time complexity of the algorithm is polynomial.

4.5 Experimental Analysis

We perform an extensive experimental analysis to evaluate the performance of the proposed mechanisms, G-ERAP and APX-ERAP with respect to several key metrics. First,

we compare the performance of the mechanisms with that of the optimal solution obtained by solving small-size instances of the ERAP-MILP problem using the CPLEX solver [98]. Second, we compare the performance of the two proposed mechanisms for large-size problem instances. Since for the large-size instances, obtaining the optimal solution for ERAP-MILP within a reasonable amount of time is not possible, we compare the performance of the proposed mechanisms with that of the LP relaxation of ERAP-MILP. Furthermore, to investigate the impact of dynamic provisioning, we compare the performance of the mechanisms under both dynamic provisioning and static pre-provisioning. In the following, we describe the experimental setup and analyze the experimental results.

4.5.1 Experimental setup

We generate several problem instances with different number of users, demands, and capacities for the edge and cloud. In our experiments, the provider offers four types of VM instances as shown in Table 4.1. These types of VM instances are based on Amazon's Elastic Compute Cloud (EC2) M3 family of instances. In this family, four types of VM instances are defined, medium, large, xlarge, and 2xlarge. Each type of VM instance provides a combination of three types of resources, vCPU, memory, and storage. We assume that the same types of VMs can be provided at both the edge and the cloud levels. However, depending on the capacity of resources, during the dynamic provisioning, some types of VMs may not be provisioned at the edge or cloud level. We define 3.75 GB of memory as one unit of memory and every 4GB of storage as one unit of storage. Therefore, in the EC2 VM instances, the size of CPU and memory varies from 1 unit to 8 units, while the size of storage varies from 1 unit to 40 units. The distribution of the other parameters that are used to generate problem instances in our simulation experiments are shown in Table 4.3. In this table, we denote by $U[a, b]$, the uniform distribution within interval $[a, b]$. The bid b_i of user i must be proportional to the total amount of request of users i. Therefore, to generate bid b_i, we first draw from $U[1, 10]$, the bid per unit of resource. Then, we multiply this value by the total size of the request of user i, $R_i = \sum_{j=1}^{m} \sum_{k=1}^{d} w_k \cdot q_{jk} \cdot r_{ij}$.

Table 4.3: Simulation parameters for small instances

Parameter	Distribution
C_1, C_2	small: 25000 large: 250000
C_3	small: 200000 large: 2000000
r_{ik}	$U[0, 10]$
w	$[1, 1, 1]$

We also use the uniform distribution to generate r_{ik}, the number of VM instances of each type that are requested by user i. We evaluate the performance of the mechanisms for both small-size instances and large-size instances. For small-size instances, the number of users varies from 100 to 500. Since the number of VM instances of each type of VM is drawn from $U[0, 10]$, the total request of users for each type of VM varies from $100 \cdot U[0, 10]$ to $500 \cdot U[0, 10]$. Thus, to cover the requests of a reasonable number of users, the available capacity of each type of resources must be proportional to these values. For CPU and memory, the available capacity is 25000, while for the storage the available capacity we use 200000. For large-size instances, the number of users varies from 5000 to 50000. Thus, to cover the requests of a reasonable number of users, the available capacity of CPU and memory is 250000, while for the storage the available capacity we use 2000000. The vector of resource weights w, for the three types of resource is given in the last row of the table. For each size of instances, we execute G-ERAP and APX-ERAP, and CPLEX for 10 randomly generated instances and perform our analysis based on the average values.

One of the important factors on the performance of the mechanisms is the amount of available resources at the edge level and the cloud level. Therefore, we define a parameter ρ_{EC}, called *edge-cloud resource capacity ratio*, which is the sum of the ratios of the capacity for each type of resources at the edge and the capacity at the cloud level, $\rho_{EC} = \sum_{k=1}^{d} (\frac{C_{k1}}{C_{k2}})$.

Social welfare and revenue are two important measures for the efficiency of resource allocation mechanisms. To characterize the welfare and revenue obtained by G-ERAP and APX-ERAP, we define two relative metrics: (i) the *social welfare ratio*, $\mathcal{V}^r = \frac{V}{V^*}$, where

V is the social welfare obtained by G-ERAP or APX-ERAP, and V^* is the social welfare of the optimal solution obtained by CPLEX; and, (ii) the *revenue ratio*, $\mathcal{R}^r = \frac{R}{R^*}$, where R is the revenue obtained by G-ERAP or APX-ERAP, and R^* is the revenue obtained by the optimal solution obtained by CPLEX. R^* and R are calculated using Equation (4.6). Note that, to determine the revenue of the CPLEX solution, we use the pricing rule defined for APX-ERAP.

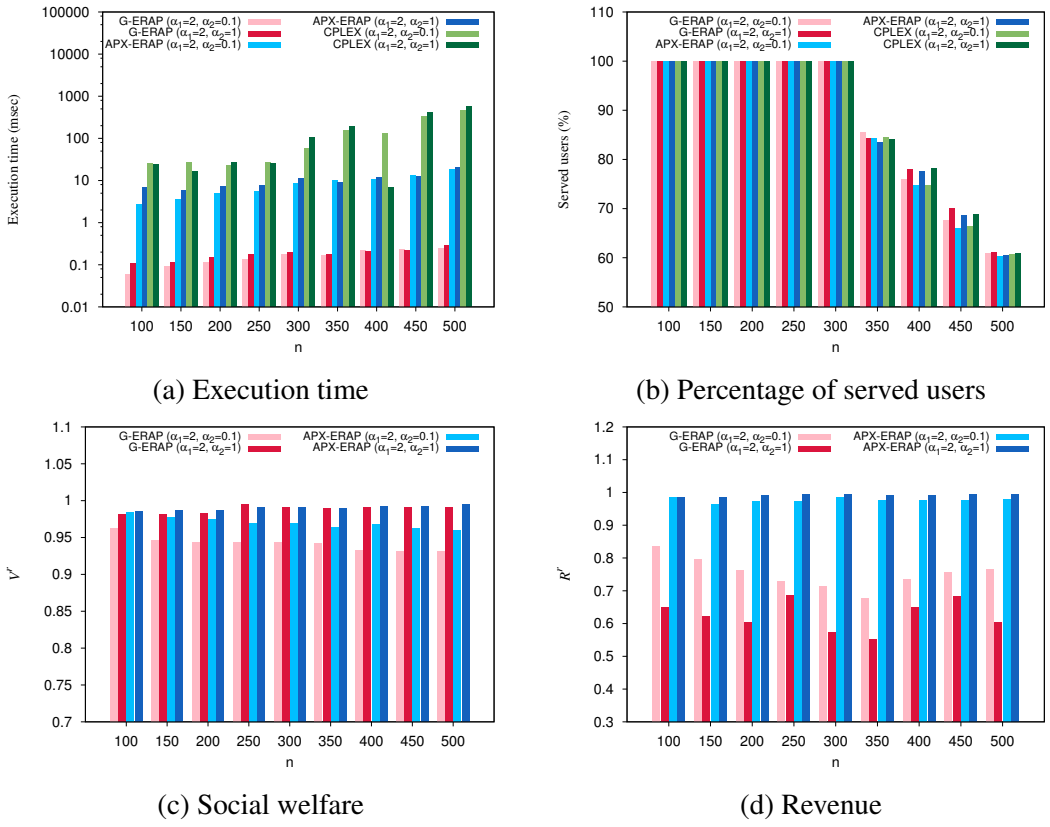
To investigate the impact of the dynamic provisioning on the performance of the mechanisms, we compare their performance considering two types of MEC systems: one that allows dynamic provisioning, and another one in which the VMs are pre-provisioned. Both systems have the same amount of physical resources. In the system with pre-provisioning, the resources are equally provisioned into four types of VMs. In fact, since we assume the same range of demand for each type of VMs (the number of requests for each type of VMs for each user is drawn from the same distribution), we equally distribute the resources among the four types of VMs.

The mechanisms are implemented in C++ and the experiments are conducted on an Intel 1.6GHz Core i5 system with 8 GB RAM. For solving ERAP-MILP, we use the CPLEX 12 solver provided by IBM ILOG CPLEX optimization studio for academics initiative [98].

4.5.2 Analysis of results

In this section, we compare the performance and the scalability of the mechanisms for different types of problem instances. First, we investigate the performance of the mechanisms for small-size instances by comparing with the CPLEX solution. Then, we investigate the scalability and performance of the mechanisms for large-size instances. Finally, we investigate the performance of the mechanisms on systems with different edge-cloud resource capacity ratio values.

Performance with respect to the number of users (small-size instances). First, we investigate the effects of the number of users on the performance of the mechanisms. In this experiment, we compare the performance of the mechanisms against the optimal solu-



(a) Execution time

(b) Percentage of served users

(c) Social welfare

(d) Revenue

Figure 4.1: The effect of the total number of users, n, on the performance (small-size instances).

tions obtained by solving ERAP-MILP using CPLEX. For this purpose, we only consider small-size problem instances. We consider a fixed amount of total capacity for each type of physical resources at both edge and cloud levels. We set ρ_{EC} for each type of resource instance to $1/9$, that is, 90% of the total capacity of each type of resources is at the cloud level, and 10% at the edge level. We perform experiments with two sets of problem instances. In order to investigate the performance of the proposed mechanisms for applications with different latency requirements, we perform our experiments using two sets of values for (α_1, α_2). For the first set, we consider the preference factors $\alpha_1 = 2$, and $\alpha_2 = 1$, while for the second set, we consider $\alpha_1 = 2$, $\alpha_2 = 0.1$. We consider different number of users, n, ranging from 100 to 500. Figure 4.1a shows the execution time of G-ERAP, APX-ERAP, and CPLEX for the two sets of problem instances. In all cases, the execution time of G-ERAP is less than one millisecond, significantly smaller than the time required by CPLEX

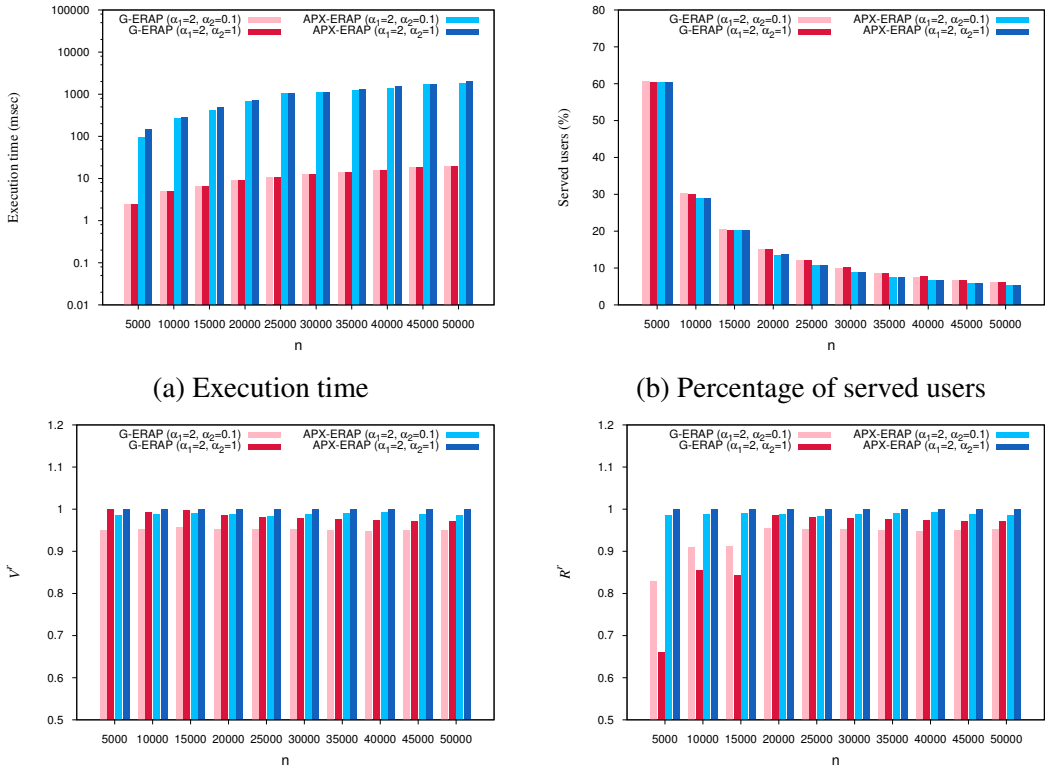
to obtain the solution. Also, the execution time of APX-ERAP is less than 11 milliseconds. We observe an increase in the execution time of G-ERAP with the increase in the number of users. The reason is that the running time of the mechanims grows linearly with the number of users. Overall, both G-ERAP and APX-ERAP are not very sensitive to the number of users. In contrast, the execution time of CPLEX is very sensitive to the number of users and it increases significantly as the number of users in the system increases. Figure 4.1b shows the percentage of users served by the provider when employing the solution determined by G-ERAP, APX-ERAP, and CPLEX. For instances with a small number of users, there is no difference between the solution of the mechanisms in terms of the number of served users. With an increase in the number of users we observe that G-ERAP serves more users. The reason is that G-ERAP greedily allocates users that have high bid density. Therefore, it may assign users that have relatively higher bid and smaller request size.

Figure 4.1c shows the social welfare ratio for the two sets of problem instances. The results obtained by G-ERAP and APX-ERAP for both sets of problem instances are fairly close to those of CPLEX. Furthermore, the value of social welfare ratio is above 0.93. In most cases, the performance of APX-ERAP is better than that of G-ERAP, specially for the second set of problem instances ($\alpha_1 = 2$, $\alpha_2 = 0.1$) in which the the edge level is more preferred. In fact, both G-ERAP and APX-ERAP are more sensitive to the number of users for the second set of instances. The reason is that when applications have significantly low latency requirements, $\alpha_1 \gg \alpha_2$, any inappropriate allocation has significant effects on the social welfare. Figure 4.1d shows the revenue ratios. We observe no significant gap between the performance of APX-ERAP and that of CPLEX. The reason is that APX-ERAP and CPLEX use the same pricing rule. However, G-ERAP obtains lower revenue than the other mechanism. The reason is that G-ERAP considers a lower payment for the allocated users. Furthermore, we observe that for $n \leq 350$, the revenue obtained by G-ERAP decreases by increasing the number of users, while for $n > 350$, the revenue increases by increasing the number of users. The reason is that for $n \leq 350$ the provider

has enough capacity to allocate most of the users. Thus, by increasing the number of users, more users are allocated. This results in a higher revenue for CPLEX which is obtained by adding the bids of the winners. However, the revenue of G-ERAP which is obtained based on the bids of the last allocated bidder and the first losing bidder on the cloud and the edge level, does not grow at the same rate. For $n > 350$ when the number of losers increases (see Figure 4.1b), the ratio between the highest bidder that loses and the average bids of winners decreases. Thus, the revenue of G-ERAP, which is affected by the bids of losers, gets closer to the revenue of CPLEX which is based on the bids of the winners. Another observation is that the revenue obtained by G-ERAP is closer to that of CPLEX for the case $\alpha_1 \gg \alpha_2$. The reason is that for these settings the low revenue obtained by G-ERAP from the cloud side does not affect the total revenue of the system (due to the small value of α_2).

Performance and scalability with respect to the number of users (large-size instances). We now evaluate the performance of G-ERAP and APX-ERAP by varying the number of users. We assume a fixed capacity at both edge and cloud level and vary the number of requests from 5000 to 50000. In this set of experiments, we assume that 90% of resources are at the cloud level while only 10% of resources are available at the edge level. The values of other parameters are given in Table 4.3. Since CPLEX is not feasible to be used for solving such large instances, we will not compare the performance of our mechanisms against the performance of the optimal solution obtained by solving ERAP-MILP. Instead, we will compare the performance of our mechanisms against the LP relaxation of ERAP-MILP (which gives the upper bound on the optimal solution). In order to do this, we redefine the *social welfare ratio* as the ratio between the social welfare obtained by G-ERAP or APX-ERAP, and the social welfare obtained by the LP relaxation solution. We also redefine the *revenue ratio*, as the ratio between the revenue obtained by G-ERAP or APX-ERAP, and the revenue obtained by the LP relaxation solution.

Figure 4.2a shows the average execution time of both G-ERAP and APX-ERAP, for



(a) Execution time

(b) Percentage of served users

(c) Social welfare

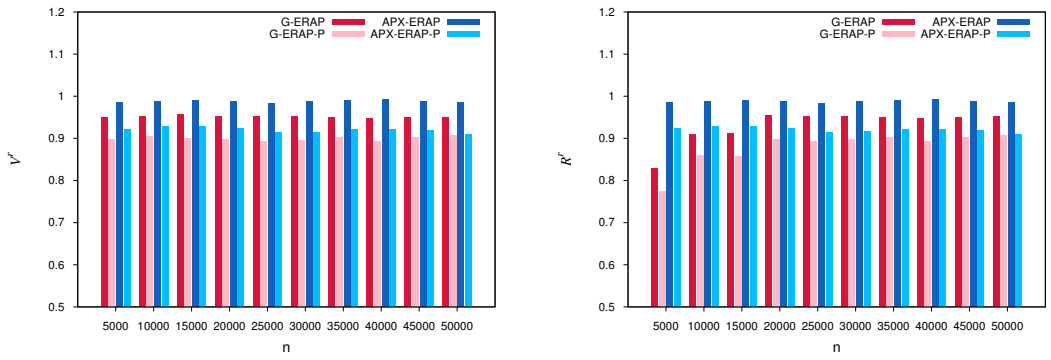
(d) Revenue

Figure 4.2: The effect of the total number of users, n, on the performance (large-size instances).

the two sets of problem instances. We observe that by increasing the number of users, the execution time of both G-ERAP and APX-ERAP increases linearly. For example, the average execution time of G-ERAP for $n = 5000$ is about 2.5 milliseconds and for $n = 50000$, the average execution time is about 18 milliseconds. We observe a similar behavior for APX-ERAP. The average execution time of APX-ERAP for $n = 5000$ is about 120 milliseconds and for $n = 50000$, the average execution time is about 1790 milliseconds. Figure 4.2b shows the average percentage of served users for various numbers of users. We observe that by increasing the number of users, a lower percentage of users are served. This is because the total capacity is fixed and the system is not able to serve all the requests. Again, we observe that by increasing the number of users, G-ERAP serves more users. The reason is that G-ERAP may greedily assign users that have relatively higher bids and smaller request sizes. Figure 4.2c shows the effect of the number of users on the social

welfare for the two sets of problem instances. Both G-ERAP and APX-ERAP are able to maintain a welfare ratio above 0.95. A similar behavior is observed for APX-ERAP when we consider the effects of the number of users on the revenue (Figure 4.2d), where the revenue ratio is kept above 0.95. However, G-ERAP obtains a lower revenue compared to APX-ERAP. The reason is that the LP relaxation and APX-ERAP use the same pricing rule to determine the revenue while G-ERAP considers a lower payment for the allocated users. We also observe that by increasing the number of users, the revenue of G-ERAP gets closer to that obtained by the LP relaxation. The reason is that by increasing the number of users, the number of losers increases. Thus, with a high probability, the ratio between the bid of the highest bidder that loses and the average bid of the winners decreases. Therefore, the revenue of G-ERAP, which is affected by the bids of losers, becomes closer to the revenue obtained by the LP relaxation, which is based on the bids of the winners. The experimental results on large-size instances show that G-ERAP can solve fairly large-size problem instances within 20 milliseconds while keeping the social welfare and revenue within an acceptable distance from those obtained by the LP relaxation solution. These results show that G-ERAP can be employed efficiently in real world systems with a large number of users.

Performance with respect to dynamic provisioning. Here, we investigate the effects of the dynamic provisioning on the performance of the system. For this purpose, we consider a system in which the number of VM types at each level is pre-determined (as explained in subsection 4.5.1). In this set of experiments, we consider the same setup as that for large size instances. We vary the number of requests from 5000 to 50000. For more readability, in Figure 4.3 we show the performance of the mechanisms in the pre-provisioning (denoted by G-ERAP-P and APX-ERAP-P) and dynamic provisioning cases only for preference factors $\alpha_1 = 2,\ \alpha_2 = 1$. Figure 4.3a shows that the social welfare of the system in the pre-provisioning case is lower than in the case of dynamic provisioning for both G-ERAP and APX-ERAP. Figure 4.3b shows a similar behavior of the mechanisms in terms of the



(a) Social welfare

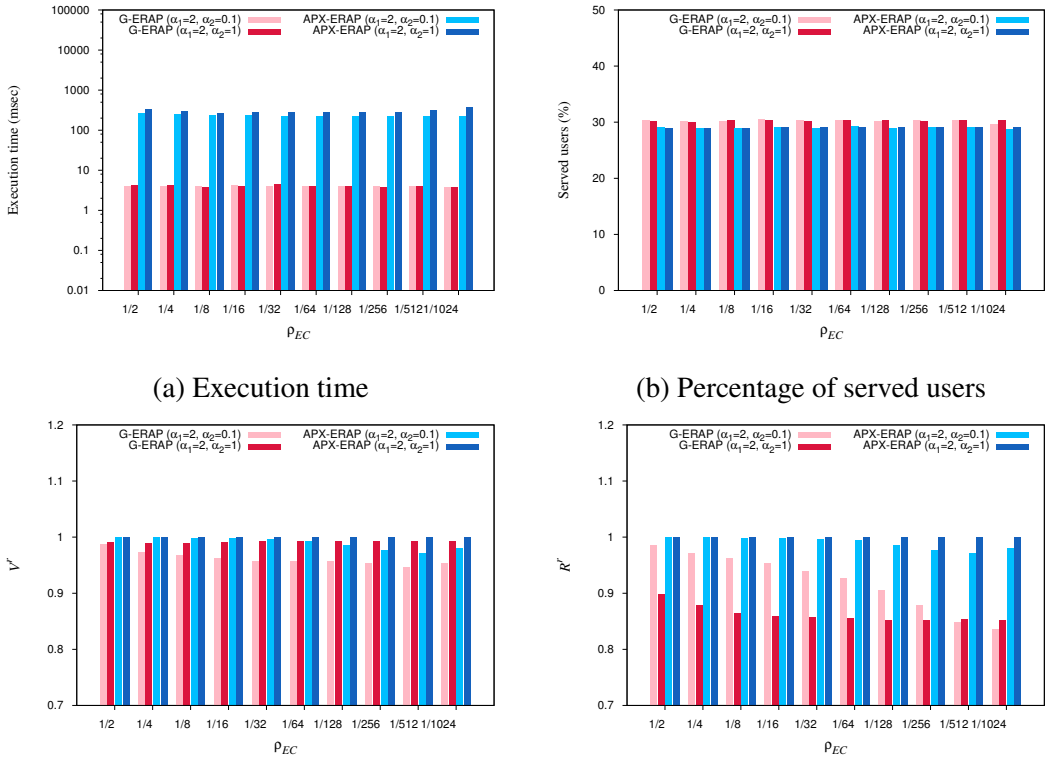
(b) Revenue

Figure 4.3: The effect of the dynamic provisioning on the social welfare and the revenue ($\alpha_1 = 2, \alpha_2 = 1$).

revenue. The reason behind this observation is that in a system with pre-provisioning, some VMs that are statically provisioned may not be used as they are the surplus to the demand, while some other VMs that are more wanted are scarce. Therefore, the resource manager is not able to utilize its resources according to the demand, and the social welfare and the revenue of the system decreases.

Performance with respect to the edge-cloud resource capacity ratio (large-size instances). In this set of experiments, we investigate the effects of the edge-cloud resource capacity ratio, ρ_{EC} on large-size problem instances with 10000 users. We draw the total capacity of each type of physical resources from the distributions given in Table 4.3 and keep it fixed. We allocate the capacity to each level according to the following edge-cloud resource capacity ratios, $\rho_{EC} = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}, \frac{1}{128}, \frac{1}{256}, \frac{1}{512}$, and $\frac{1}{1024}$. These ratios will allow us to investigate the performance of G-ERAP and APX-ERAP on systems with plenty of available resources at the edge level ($\rho_{EC} = \frac{1}{2}$), and systems with very few resources at the edge level ($\rho_{EC} = \frac{1}{1024}$). We perform our experiments with two sets of problem instances. In the first set, we consider the preference factors $\alpha_1 = 2$, and $\alpha_2 = 1$, while in the second set, we consider $\alpha_1 = 2$, $\alpha_2 = 0.1$.

Figure 4.4a shows the effects of ρ_{EC} on the average execution time of both G-ERAP and APX-ERAP, for the two sets of problem instances. For all values of ρ_{EC}, the execution



(a) Execution time

(b) Percentage of served users

(c) Social welfare

(d) Revenue

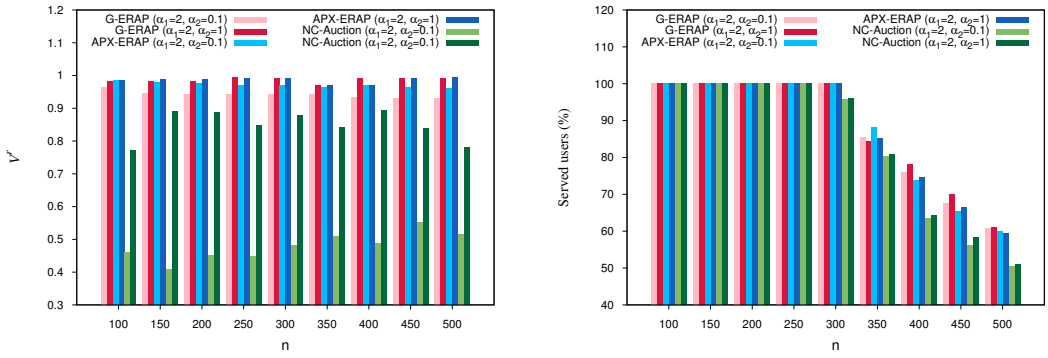
Figure 4.4: The effect of the capacity ratio on the performance (large-size instances).

time of G-ERAP is less than 10 milliseconds, while the execution time of APX-ERAP is less than 300 millisecond. Also, the execution time of both G-ERAP and APX-ERAP are not significantly sensitive to variations of ρ_{EC}. Figure 4.4b shows the effects of ρ_{EC} on the percentage of served users when considering the solution obtained by G-ERAP and APX-ERAP. Similarly to the execution time, we observe that ρ_{EC} does not have significant effects on the number of served users for both G-ERAP and APX-ERAP solutions. Figure 4.4c shows the effect of ρ_{EC} on the social welfare for the two sets of problem instances. Both G-ERAP and APX-ERAP are able to maintain a welfare ratio above 0.9 even in the worst cases, where resources are very scarce at the edge level ($\rho_{EC} = \frac{1}{1024}$). A similar behavior is observed for APX-ERAP when we consider the effects of ρ_{EC} on the revenue (Figure 4.4d), where the revenue ratio is kept above 0.9 even when the resources at the edge level are very scarce. The reason is that the same pricing rule is used to determine

the revenue of the LP solution and that of APX-ERAP. However, G-ERAP obtains a lower revenue than the other algorithms due to considering a lower payment for allocated users. We also observe that when the edge level is more preferred ($\alpha_1 \gg \alpha_2$), the efficiency of the proposed mechanisms is affected by the distribution of the resources among the levels. This is because the valuation for the edge level resources is higher and those resources are scarcer.

Performance comparison with non-combinatorial auctions. In order to investigate the impact of non-combinatorial bidding on the performance metrics, such as that of the bidding considered by Kiani et al. [66], we compare the performance of G-ERAP and APX-ERAP algorithms against that of a class of non-combinatorial auctions (called NC-Auction). In a non-combinatorial auction the user who needs a bundle of VM instances composed of VM instances of various types, submits a separate bid for each individual type of VM instances in the bundle. This is in contrast with combinatorial auctions (G-ERAP and APX-ERAP) in which a user submits a single bid for the whole bundle of VM instances. Kiani et al. [66] employed such a type of non-combinatorial auction. In their approach, a user submits a bid for each type of VM and the users' requests are ordered in descending order of their bids and are allocated accordingly to the edge level and the cloud level. Furthermore, their approach requires static provisioning in which VMs are provisioned in advance. We generate the bids of users for the whole bundle of the request as described in Subsection 4.5.1. Thus, the valuation of user i for the whole bundle is obtained by multiplying the bid per unit of resource and the total size of the request of user i. We assume that the VMs are complementary goods (i.e., the valuation for two VMs A and B is greater than or equal to the sum of individual valuations of A and B), and thus in the case of NC-Auction, the valuation per unit of resource of a partially allocated bundle is reduced between 5-15% (randomly drawn) compared to the case in which the whole bundle is allocated.

Figure 4.5a shows the social welfare ratio obtained by G-ERAP, APX-ERAP, and NC-Auction for the two sets of problem instances with ($\alpha_1 = 2$, $\alpha_2 = 1$) and ($\alpha_1 = 2$,



(a) Social welfare

(b) Percentage of served users

Figure 4.5: Comparison with NC-Auction (small-size instances).

$\alpha_2 = 0.1$). In all instances, the social welfare ratio obtained by G-ERAP and APX-ERAP is higher than that obtained by NC-Auction. The reason is that NC-Auction allocates VMs based on the individual bids, while G-ERAP and APX-ERAP consider the bid for the whole bundle. Thus, NC-Auction may greedily allocate a VM instance to a user with a relatively high individual bid for the VM, while the bids for the other VMs needed by the user are low. If we consider this as a bundle the sum of the bids in the bundle is low. On the other hand the G-ERAP and APX-ERAP allocate higher value bundles and will obtain a higher social welfare. Furthermore, the valuation of a user for the whole bundle of the request is higher than the sum of the individual bids. Thus, there might be a case in which user i has a higher individual bid for each type of VM compared to user i'; but the bid for the whole bundle of user i is lower than that of user i'. In this case, NC-Auction greedily allocates user i, while G-ERAP and APX-ERAP allocate resources to user i' and obtains a higher social welfare.

Figure 4.5b shows the percentage of the served users for the two sets of problem instances. We observe that when the number of users is relatively low ($n \leq 250$), G-ERAP, APX-ERAP, and NC-Auction serve the same percentage of users. The reason is that there is enough capacity at both edge and cloud levels to serve the requests. As the number of users increases, NC-Auction allocates a lower percentage of users compared to G-ERAP and APX-ERAP. NC-Auction may not able to utilize resources according to the demand,

thus the number of served users decreases. NC-Auction allocates VMs one by one based on their individual bids. Thus, a relatively high percentage of users may receive a partial allocation, while G-ERAP and APX-ERAP only allocate whole bundles.

Our experimental results showed that for small problem instances, both G-ERAP and APX-ERAP yield solutions close to those obtained by the CPLEX optimal solution. Compared to APX-ERAP, G-ERAP has a very small execution time. However, in systems where users have high preference for the edge level ($\alpha_1 \gg \alpha_2$), APX-ERAP is more efficient than G-ERAP in terms of social welfare and revenue. Overall, the low execution time and the acceptable distance from the optimal solution make G-ERAP mechanism suitable for edge computing systems with a large number of users.

4.6 Conclusion

Monetization of services is one of the grand challenges in edge computing systems. In this chapter, we proposed two resource allocation and pricing mechanisms in edge computing systems, where users have heterogeneous requests and compete for high quality services. We proved the properties of the proposed mechanisms and evaluated their efficiency by performing an extensive experimental analysis. For small-size instances, we compared the solutions obtained by the proposed mechanisms with the optimal solutions obtained by the CPLEX solver with respect to execution time, percentage of served users, social welfare and revenue. For large-size instances, we compared the performance of the two proposed mechanisms with respect to the same metrics used for the analysis on small-size instances. The experimental results showed that the resource allocation obtained by the proposed mechanisms yield near optimal solutions. In addition to the quality of solutions, the small execution time makes the proposed mechanisms promising for deployment in edge computing systems.

CHAPTER 5

VECMAN: A FRAMEWORK FOR ENERGY-AWARE RESOURCE MANAGEMENT IN VEHICULAR EDGE COMPUTING SYSTEMS

5.1 Introduction

Electric Connected Autonomous Vehicles (eCAVs), the future of our transportation system, have two main requirements: processing massive amount of data with minimum latency and having long driving ranges. To address the first requirement computational nodes must be placed closer to the eCAVs at the *edge* of the cloud [13]. Thus, in these so called *Vehicular Edge Computing (VEC)* systems, computational nodes are placed in Road-Side Units (RSUs) and exploit the Dedicated Short Range Communication (DSRC) technology [110] for vehicle-to-vehicle and vehicle-to-RSU communications. However, the scalability of these systems may be hindered by the amount of vehicles and workloads in RSU coverage areas. Indeed, due to the limited capacity of RSUs, some computational tasks may experience poor Quality of Service (QoS) or even failure. In order to improve reliability, computing resources (e.g., Nvidia Drive Px 2) are installed on each vehicle for local workload execution and minimized latency. On the other hand, having local computing resources might affect the eCAV's driving range. For example, a preliminary study by Lin et al. [111] shows that a computing node (including computing, storage, and cooling hardware) can reduce the driving range of a Chevy Bolt by up to 11.5%. However, the authors only considered three executing tasks on various computing configurations and did not consider the effect of the eCAV speed on the driving range. To conduct a more general analysis, we exploit the driving range vs speed data available for the Tesla Model S [112], which can be used to easily estimate the total power consumption of the eCAV at different speeds given the battery pack size. Figure 5.1 shows the results for the addition of a computing system of power consumption varying from 0W to 3kW. We observe that the impact of the computing power on the driving range varies from 10% to 40% for a 2kW computing power. This is computed for the eCAV traveling at average speeds from 60mph to 20mph,

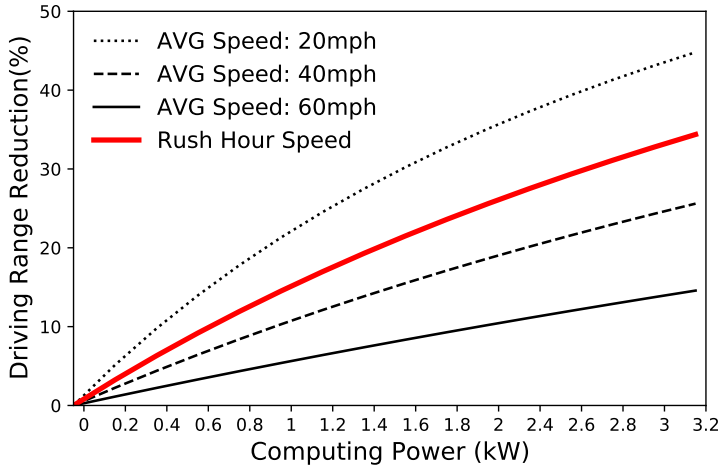


Figure 5.1: Effect of computing power on eCAV's driving range.

respectively. Considering that the rush-hour (i.e., 4-5pm) speed in 20 US cities is 33mph on average [113], a 2kW computing system can cause a 25% reduction in driving range on modern electric vehicles. As a result, because most eCAVs on the road during weekdays travel at rush hours and because during this period eCAVs spend a considerable amount of time near each other due to traffic, it is desired to design new strategies that allow to reduce the energy consumption of eCAVs computing systems for longer driving ranges.

In order to achieve high QoS, reliability, and increased driving range, eCAVs in a VEC system could share their resources. Rather than relying only on the limited capacity of RSU nodes, the computing resources of the vehicles can be coordinated to achieve energy savings. The key-intuition for achieving energy savings is to 1) exploit the *computational slack* caused by discrete power/QoS settings of computing resources (e.g., CPUs and GPUs), and 2) exploit the non-linear relationship between these settings and the computational power consumption [114]. First, CPUs commonly have a fixed number of selectable configurations for voltage-frequency levels and number of cores to trade-off power consumption and QoS. Each configuration leads to a maximum number of instructions that can be executed within a certain time period. If the local workload exceeds that maximum, the system must select a new configuration that increases the QoS at the cost of a higher power consumption,

e.g., activate more cores or increase the voltage-frequency level. However, this selected *default* configuration may not be fully utilized by the local workload, i.e., there exist a slack of computational capacity that can be used to execute extra workload at the same default configuration. Second, there exist a non-linear relationship between the computational power consumption and the frequency settings [114]. Thus, a taskset executing on an eCAV operating at a high CPU frequency may consume much more power than the same taskset partially executing on an eCAV operating at a lower CPU frequency with similar overall performance. Thus, the system can achieve energy savings by offloading part of *requester* vehicle workload (currently operating at a high frequency) to *provider* vehicles (currently operating at a low frequency) that can exploit the providers' computational slack without changing their default configuration, which limits the provider vehicles power overhead. Then, at a later time, providers can become requesters to achieve higher energy savings.

Enabling resource sharing among eCAVs in a VEC system requires the dynamic determination of two important parameters: (a) the set of participating vehicles in resource sharing, and (b) the time duration of resource sharing. However, the uncertainties in the future location of vehicles make it hard to decide these parameters so that all participants benefit from resource sharing. In this chapter, we overcome these challenges by proposing VECMAN, a framework for energy-aware resource management, which consists of two algorithms: (i) a resource selector algorithm that runs periodically on the local RSU and determines the set of participating vehicles as well as the duration of resource sharing; and (ii) an energy manager algorithm that runs periodically at a finer time grain within each resource sharing time period and determines the state of each participating vehicle, i.e., requester or provider, the number of replicas for the requesters' workloads, and the amount of requester's workload to offload so that energy consumption is minimized for all the participating vehicles.

5.1.1 Our contributions

Our main contributions are as follows:

- We formulate the resource selection problem and the energy manager problem in the VECMAN framework. The objective of the resource selector is to dynamically determine the set of participating vehicles and the duration of the resource sharing period for maximized computing resources and reliability. The objective of resource manager is to maximize the total energy saving of vehicles while minimizing the risk of failure. The VECMAN framework allows partial offloading of vehicles' workload.

- VECMAN is robust to uncertainties in the future location of vehicles. However, the optimization problem solved by VECMAN is a chance-constrained problem, and thus, obtaining the optimal solution in a reasonable amount of time is not freasible. Therefore, VECMAN is designed based on an iterative algorithm called I-Selector to solve the resource selection problem, and a greedy algorithm called G-ERMP to solve the energy-aware resource management problem.

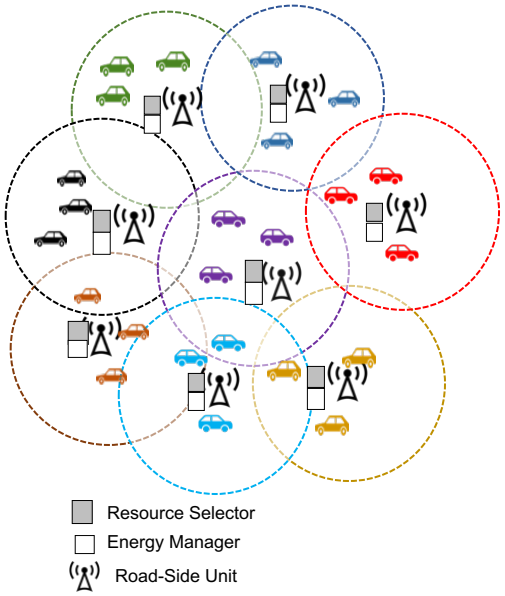
- We test VECMAN using a real-world dataset of vehicular mobility collected in the city of Cologne, Germany [115]. The results show that VECMAN achieves between 7% and 18% energy savings compared to a baseline that executes workload locally, and an average of 13% energy savings compared to a baseline that offloads workloads only to RSUs.

5.1.2 Organization

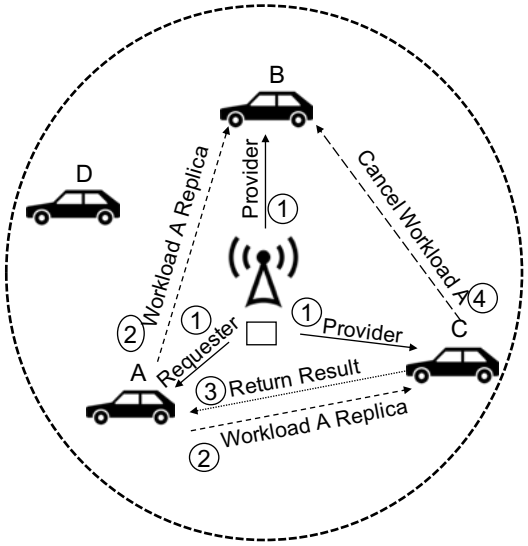
This chapter is organized as follows. Section 5.2 provides the framework problem formulation. Section 5.3 presents the proposed algorithms that solve the framework problem formulation. Section 5.4 describes the experimental results and Section 5.5 concludes the chapter.

5.2 VECMAN problem formulation

VECMAN is characterized by two main components, the *resource selector* and the *energy manager*. The resource selector takes the vehicles' information (i.e., location and computing resources) into consideration periodically (at a coarse time scale) and, based on



(a) An area with several vehicles and RSUs.



(b) Each RSU runs the resource selector and the energy manager locally.

Figure 5.2: A schematic set up of a VEC system.

the history of the traffic data in the coverage area and the movement behavior of vehicles, (a) it selects the set of vehicles that can participate in resource sharing, (b) it determines the length of the period so that each selected vehicle remains in the coverage area with a high probability during the resource selection period. We assume that the real-time location information of a vehicle is obtained by the on-board global position system (GPS). The resource selector, based on the history of traffic data and the current location of a vehicle, determines the probability that the vehicle stays in the coverage area to decide whether or not to engage that vehicle in the resource sharing. At a finer time scale than the resource selector, the energy manager, decides the vehicles' state (i.e., requester or provider), the number of replicas for each requester's workload, and the replicas' allocation on provider vehicles. The resource selector and the energy manager run at different time scales, and thus the problems they solve can be treated as separate.

5.2.1 Illustrative example

Figure 5.2a shows the distributed architecture of a VEC system implementing VEC-MAN. Each vehicle is assigned to the nearest RSU. Each RSU runs the resource selector

and the energy manager locally. Figure 5.2b shows the coverage area of an RSU, with four assigned vehicles. In this example, the resource selector selects vehicles A, B, and C to participate in resource sharing for T units of time. Due to the high probability of leaving the coverage area, vehicle D is not selected to participate in resource sharing. At a finer grain, the energy manager is executed on the selected vehicles for T/T_{em} periods, where T_{em} is the length of each energy manager period. In the first period, the energy manager (1) selects vehicle A as a service requester and vehicles B and C as service providers. It also determines two replicas for the request of vehicle A to be assigned on vehicles B and C. (2) The energy manager coordinates the replicas deployment. (3) When one of the providers returns the computation result to A (e.g., vehicle C), (4) the other providers stop the computation and wait for other workloads. In the following energy manager periods, vehicle A will host the workloads of vehicles B and C to help them lower their energy consumption. Thus, all the vehicles can save energy over the resource selector period T.

In the following, we formulate the resource selection problem (RSP) and the energy-aware resource management problem (ERMP) in VEC systems.

5.2.2 RSP formulation

The inputs of the resource selector are a fixed set V of vehicles that are currently located in the coverage area, their available computing resources, and the historical information about locations of vehicles until the current time. Based on these parameters, the resource selector determines (a) the length of resource selection period T, and (b) the subset of vehicles $\mathcal{V}$ that with a risk factor less than α stay in the coverage area for T units of time. In fact, the goal of the resource selector is to engage as many computing resources of vehicles as possible but at the same time guarantee that each vehicle, by participating in sharing computing resources, can save some energy before the next invocation of the resource selector algorithm. At that point, a new period length is determined based on the current vehicle motion information.

We consider the number of instructions that can be executed by the CPU in a resource

selection period as the amount of computing resources of a vehicle. Thus, increasing the length of the resource selection period increases the amount of computing resources of each vehicle. However, increasing the length of the period may decrease the probability that the vehicle remains in the RSU coverage area, which leads to a lower expected value of the total computing resources. Thus, to maximize the amount of computing resources, the resource selector should consider a trade off between the length of the period and the probability that the vehicles stay in the coverage area during the resource selector period. Thus, the *objective of the resource selector* is to find a subset of vehicles $\mathcal{V}$ over all possible subsets of vehicles Λ so that the total amount of computing resources for $T_{\mathcal{V}}$ units of time is maximized:

$$\max_{\mathcal{V}\in\Lambda} \sum_{i\in\mathcal{V}} M_i \cdot T_{\mathcal{V}} \tag{5.1}$$

where M_i is the Millions Instructions Per Second (MIPS) for the CPU of vehicle i, and $T_{\mathcal{V}}$ is the maximum number of units of time that vehicles in $\mathcal{V}$ stay in the coverage area with a risk factor less than α. In other words, $T_{\mathcal{V}}$ is the maximum number of units of time that satisfies the following risk factor constraint for all vehicles in $\mathcal{V}$:

$$p\left\{\bigcap_{t=1}^{T_{\mathcal{V}}}(\ell_i^t \leq \rho)\right\} \geq 1-\alpha, \;\; \forall i \in \mathcal{V} \tag{5.2}$$

where ρ is the coverage range of the RSU and ℓ_i^t is a probabilistic parameter indicating the distance of vehicle i from the RSU at time unit t. The probability distribution of ℓ_i^t is obtained based on the possible scenarios for the location of the vehicle in time unit t. Note that, to increase the readability, in the rest of the chapter, we use T instead of $T_{\mathcal{V}}$ when we refer to the currently selected time duration of the resource selector.

5.2.3 ERMP formulation

The inputs of the energy manager are T, the length of resource selector period, $\mathcal{V}$, a fixed set of vehicles that with a risk factor less than α stay in the coverage area, the history of vehicle locations until the current time, and the workload characteristics. We assume that the RSU runs the energy manager for F_e periods within T units of time. The length of each period is fixed and is denoted by T_{em}, which can be set to a value that satisfies the Quality of Service (QoS) for the workloads execution, i.e., every workload should complete its execution within T_{em} units of time. Thus, the energy manager, at each one of the F_e invocations, decides the vehicle's state and the number of replicas for each selected requester so that, after F_e periods, the vehicles' energy consumption is minimized without violating the QoS requirements. In other words, we want to ensure that each vehicle, by participating in sharing computing resources, can save some energy before the end of the current resource selector period. Next, we identify two important constraints, i.e., limited capacity and risk factor, define the energy consumption model of the vehicles, and finally, formulate the ERMP.

Capacity Constraints. We characterize the capacity of vehicle i by $C_i = \{C_{1i}, \ldots, C_{Qi}\}$, where Q is the number of resource types. We consider three resource types (i.e., $Q = 3$) indexed by h: CPU ($h = 1$), memory ($h = 2$), and storage ($h = 3$). Each vehicle i has a limited capacity C_{hi} for each resource type h. In addition, we characterize the workload of vehicle i by $r_i = \{r_{1i}, \ldots, r_{Qi}\}$, where r_{hi} is the amount of resource of type h needed to complete the execution of the workload. Thus, r_{1i} is the number of CPU instructions (in millions), r_{2i} is the amount of memory, and r_{3i} is the amount of storage needed by the workload of vehicle i. For each vehicle i selected as a provider, the total amount of resources requested cannot exceed its available capacity:

$$\sum_{j \in S_i} {r'}_{hj} \leq C_{hi}, \quad \forall i \in P, \forall h \tag{5.3}$$

where r'_{hj} is the amount of resource of type h of vehicle j partially offloaded to provider i ($r'_{hj} \leq r_{hj}$), P is the set of providers, S_i is the set of vehicles that have a replica of their workload assigned to vehicle i, and C_{hi} is the available capacity of vehicle i for resource type h during each T_{em} units of time.

A challenge to overcome is how to calculate the CPU capacity C_{1i} in Constraint (5.3). We define the CPU capacity based on the Millions of Instructions (M_i) that can be executed within an energy manager period minus the number of CPU instructions required for the local workload:

$$C_{1i} = M_i \cdot T_{em} - r_{1i} \tag{5.4}$$

For a simple single-core CPU, the value of M_i is approximated by a function of the CPU frequency as follows:

$$M_i = \vartheta_i \cdot f_i + \theta_i \tag{5.5}$$

where ϑ_i and θ_i are estimated parameters, and f_i is the CPU frequency of each CPU. In order to ensure a good QoS, the required time to run the local workload r_{1i} must be shorter than the energy manager period duration T_{em}, which can be set as desired:

$$\frac{r_{1i}}{\vartheta_i \cdot f_i + \theta_i} \leq T_{em} \tag{5.6}$$

Given the discrete frequency levels available in every CPU, each vehicle i, to achieve the required QoS at minimum energy consumption, must set (as default) the minimum frequency level f_i that satisfies the following constraint:

$$f_i \geq \frac{r_{1i}}{\vartheta_i \cdot T_{em}} - \frac{\theta_i}{\vartheta_i} \tag{5.7}$$

As a result, the total CPU capacity C_{1i} in Equation (5.3) can be calculated as follows:

$$C_{1i} = (\vartheta_i \cdot f_i + \theta_i) \cdot T_{em} - r_{1i} \tag{5.8}$$

As a result, the capacity constraints in Equation (5.3) enable the energy manager to place extra workload on provider vehicles without affecting their default CPU frequency.

Risk Factor Constraint. Despite the resource selector efforts to provide a fixed set of vehicles within each resource selector period, it may happen that some vehicles change their location in any of the F_e energy manager periods. Thus, because the future vehicle locations can only be predicted, we need to ensure that, with some level of confidence defined by a risk factor, each requester has a good connection with at least one provider during each period. To formulate this constraint, we first need to find the minimum distance between each requester and providers. In particular, for every selected requester, we want to have at least one provider within a reliable distance $\delta > 0$. On the other hand, the location of vehicles is non-deterministic and thus it may be affected by estimation errors. As a result, we must make sure that the probability of having at least one provider within a reliable distance is greater than a satisfaction factor $(1 - \beta)$. This constraint can be expressed as follows:

$$p\left\{\min_{j \in P | i \in S_j} l_{ij} \leq \delta\right\} \geq 1 - \beta, \quad \forall i \in \mathcal{V} \setminus P \tag{5.9}$$

where l_{ij} is the average distance between vehicle i and vehicle j in the current period, and P is the set of providers.

Energy Consumption Model. The computing system energy consumption for vehicle i is mainly characterized by two components, i.e., the computational and the transmission energy consumption. The computational energy consumption includes dynamic energy consumption and the idle energy consumption. The idle energy consumption is the basic power consumption in T_{em} units of time. The dynamic energy consumption is the power

consumption for executing the requests which is proportional to the execution time of the requests and the third power of the CPU frequency [116]. According to Equation (5.6) and assuming that that all instructions are running at the same frequency, the execution time of request ${r'}_j$ on provider i is $\frac{{r'}_{1j}}{\vartheta_i \cdot f_i + \theta_i}$. Thus, the extra energy consumption by executing request ${r'}_j$ on provider i is $\lambda_i \cdot \frac{{r'}_{1j}}{\vartheta_i \cdot f_i + \theta_i} \cdot f_i^3$, where λ_i is an estimated parameter. Thus, the total extra computational energy consumption on provider i is:

$$E_i^{extra} = \lambda_i \cdot f_i^3 \cdot \frac{\sum_{j \in S_i} {r'}_{1j}}{\vartheta_i \cdot f_i + \theta_i}, \quad \forall i \in P \tag{5.10}$$

On the other hand, if vehicle i is selected as a requester, by offloading ${r'}_i$, the default frequency of vehicle i may change from f_i to ${f'}_i$, with $f_i \geq {f'}_i$. Thus, the energy saving of the vehicle is equivalent to:

$$E_i^{save} = \lambda_i \cdot f_i^3 \cdot \frac{r_{1i}}{\vartheta_j \cdot f_i + \theta_i} - \lambda_i \cdot {f'}_i^3 \cdot \frac{r_{1i} - r'_{1i}}{\vartheta_i \cdot {f'}_i + \theta_i}, \quad \forall i \in \mathcal{V} \setminus P \tag{5.11}$$

The transmission energy consumption is proportional to the transmission latency which depends on the ratio between the request size and the data rate between the requester and the provider [117]. On the other hand, the data rate is proportional to the bandwidth between the requester and the provider. Thus, the transmission energy consumption is proportional to the ratio of the request size and the bandwidth between the requester and the provider. Thus, the transmission energy consumption of vehicle i to receive a request ${r'}_j$ from vehicle j is calculated as the ratio of the request size d_j and the average bandwidth b_{ij}, i.e., $\omega_i \cdot \frac{d_j}{b_{ij}}$. The parameter ω_i is the energy consumption of vehicle i to receive one unit of data. Therefore, the total energy consumption of provider i to receive requests from other

vehicles is:

$$E_i^{rec} = \sum_{j \in S_i} \omega_i \cdot \frac{d_j}{b_{ij}}, \quad \forall i \in P \tag{5.12}$$

Similarly, the energy consumption of vehicle i to send its request to other vehicles is:

$$E_i^{send} = \sum_{j | i \in S_j} \psi_i \cdot \frac{d_i}{b_{ij}}, \quad \forall i \in \mathcal{V} \setminus P \tag{5.13}$$

where ψ_i is the energy consumption of vehicle i to send one unit of data to the network.

In order to keep track of the total energy saved and the extra energy spent by each vehicle when selected as requesters or providers, respectively, we define the *energy balance*. In each energy manager period, the energy balance of vehicle i, E_i^{blnc}, is calculated based on the energy balance $E_i^{blnc'}$ obtained from the previous periods, the transmission energy, and the energy savings/extra energy consumption in the current period. In practice, *a negative energy balance means energy savings compared to the case of always executing the workload locally*. Given the above models, the energy balance of a provider in the current energy manager period is calculated as follows:

$$E_i^{blnc} = E_i^{blnc'} + \sum_{j \in S_i} \omega_i \cdot \frac{d_j}{b_{ij}} + \lambda_i \cdot f_i^3 \cdot \frac{\sum_{j \in S_i} {r'}_{1j}}{\vartheta_i \cdot f_i + \theta_i}, \quad \forall j \in P \tag{5.14}$$

Similarly, the energy balance of a requester in the current energy manager period is calculated as follows:

$$\begin{aligned} E_i^{blnc} = E_i^{blnc'} + \sum_{j | i \in S_j} \psi_i \cdot \frac{d_i}{b_{ij}} - \Bigg(& \lambda_i \cdot f_i^3 \cdot \frac{r_{1i}}{\vartheta_i \cdot f_i + \theta_i} \\ & -\lambda_i \cdot {f'}_i^3 \cdot \frac{r_{1i} - r'_{1i}}{\vartheta_i \cdot {f'}_i + \theta_i} \Bigg), \quad \forall i \in \mathcal{V} \setminus P \end{aligned} \tag{5.15}$$

Table 5.1: Notation

Notation	Description
T	Duration of the resource selection period
F_e	Number of energy manager periods
T_{em}	Duration of the energy manager period
V	Set of vehicles in the coverage area
$\mathcal{V}$	Set of participating vehicles.
α, β	Risk factor
r'_{hi}	Amount of type h resource requested by vehicle i
l_{ij}	Average distance between vehicle i and j
C_{hi}	Available capacity of resource of type h
$E_i^{blnc'}$	Energy balance of vehicle i from previous period
E_i^{idle}	Idle energy consumption of vehicle i in an energy manager period
f_i	Default CPU frequency of vehicle i
d_i	Size of the request of vehicle i
b_{ij}	Average bandwidth between vehicle i and vehicle i
ψ_i	Transmission energy to send one unit of data
ω_i	Transmission energy to receive one unit of data
$\lambda_i, \theta_i, \gamma_i, \vartheta_i$	Estimated parameters

Formulation. The objective of the energy manager is to find a set of providers P over all possible set of providers Π and the set of replicas S_i assigned to each provider $i \in P$ over all possible replica assignments Γ_i so that the maximum energy balance over all vehicles is minimized:

$$\min_{P \in \Pi, S_i \in \Gamma_i | i \in P} \max_{j \in \mathcal{V}} \left\{ E_j^{blnc} \right\} \tag{5.16}$$

where E_j^{blnc} is obtained based on Equation (5.14) and Equation (5.15), subject to the capacity and risk factor constraints in Equations (5.3) and (5.9), respectively. Table 5.1 summarizes the notation that we use in the chapter.

5.3 VECMAN algorithms

Because of Constraints (5.2) and Constraints (5.9) RSP and ERMP are both chance-constrained optimization problems. As a result, they are robust to location uncertainties of the vehicles. However, solving chance-constrained optimization problems optimally

Algorithm 5 VECMAN Framework

Input: ID of the RSU executing VECMAN: RSU_ID

1: **while** true **do**
2: Update set of vehicles allocated to RSU_ID and their history: V, H
3: Calculate the probability vector p based on V and H
4: $\mathcal{V}, T \leftarrow$ I-Selector(V, p)
5: **for** each $k \in [1, \frac{T}{T_{em}}]$ **do**
6: Update current locations of vehicles $\mathcal{V}$ and generate scenarios ξ based on H and T
7: $S, P \leftarrow$ G-ERMP$(\mathcal{V}, \xi)$
8: Communicate S and P to vehicles
9: **end for**
10: **end while**

usually requires computationally expensive algorithms due to the large number of scenarios on the movement of the vehicles. To tackle this complexity, we only consider a sample of scenarios, where each scenario represents a potential sequence of vehicles location over consecutive energy manager periods. Then, we develop an iterative algorithm called I-Selector to find a solution for RSP and a greedy algorithm called G-ERMP to find a solution for ERMP in polynomial time.

An algorithmic description of the proposed VECMAN framework is given in Algorithm 5. The VECMAN framework running on each RSU (each one uniquely identified by an RSU_ID) starts each vehicle management loop (Lines 2-8) by updating the set of vehicles V currently within the RSU coverage area and by getting their location history H (Line 2). The RSU coverage area is divided into multiple cells. VECMAN calculates the vector p that gives, for each vehicle, the probability of leaving the RSU coverage area from each one of these cells for various candidate lengths of resource selection periods (Line 3). We refer the reader to Section 5.4.1 for more details on how p is obtained from a real dataset. Based on V and p, the I-Selector determines the length of the next resource selection period T and, at the same time, the subset of vehicles $\mathcal{V}$ that with a high probability remain in the RSU coverage area over that period (Line 4). Then, VECMAN starts executing the energy manager (Lines 5-8).

At the beginning of each one of the F_e energy manager periods, with $F_e = T/T_{em}$, the G-ERMP algorithm updates the current locations of the participating vehicles and generates a set of possible scenarios to predict where each vehicle may be by the end of the current energy manager period (Line 6). We refer the reader to Section 5.4.1 for more details on how the scenarios are obtained using a real dataset. Based on these scenarios, the G-ERMP algorithm executes to decide the vehicles' state, the number of workload replicas, and their allocations to the selected providers, for minimized energy consumption while ensuring a low risk of failure and good QoS (Line 7). Then, the computed allocations for the current energy manager period are communicated to the vehicles (Line 8). These steps (Lines 6-8) are repeated every T_{em} units of time for F_e times, after which VECMAN starts a new vehicle management loop (Lines 2-8).

5.3.1 I-Selector algorithm

The I-Selector algorithm is given in Algorithm 6. The inputs of the algorithm are the set of vehicles V with their initial location in the coverage area, and the vector p calculated as described in the previous section. The output of the algorithm consists of the set of participating vehicles $\mathcal{V}$ and the length of the resource selection period T.

I-Selector starts with a small value of T corresponding to the minimum energy manager period T_{em} (Line 1). It increases this value iteratively until it obtains a length of period that maximizes the amount of available computing resources (Lines 5-14). Since the objective function of RSP (i.e., Equation 5.1) is a bitonic function of T (the total available capacity first increases by the increase of T, and then decreases) we can guarantee that I-Selector finds the optimal length for resource selection period for the given sample of scenarios. In each iteration, for the current value of time period T, I-Selector calls find-satisfied-set procedure to obtain the set of vehicles $\mathcal{V}_{new}$ that with a risk factor less than α stay in the coverage area for T units of time (Line 6). The input of find-satisfied-set procedure is the set of vehicles V with their initial locations and the vector $p^T = \{p_a^T\}$ that gives the probability of leaving the coverage area from each cell a within T units of

Algorithm 6 I-Selector

Input: Set of vehicles: V
Vector of probabilities of leaving the coverage area: p

1: $T \leftarrow T_{em}$
2: $R \leftarrow 0$
3: $\mathcal{V} \leftarrow \emptyset$
4: $stop \leftarrow$ false
5: **while not** $stop$ **do**
6: $\mathcal{V}_{new} \leftarrow$find-satisfied-set$(V, p^T)$
7: $R_{new} = \sum_{i \in \mathcal{V}_{new}} M_i \cdot T$
8: **if** $R_{new} \geq R$ **then**
9: $R \leftarrow R_{new}$
10: $\mathcal{V} \leftarrow \mathcal{V}_{new}$
11: $T \leftarrow T + T_{em}$
12: **else**
13: $stop \leftarrow$ true
14: $T \leftarrow T - T_{em}$
15: **end if**
16: **end while**

Output: $\mathcal{V}, T$

time. As the output, the procedure returns the set of vehicles $\mathcal{V}_{new}$, where for each vehicle in $\mathcal{V}_{new}$ with an initial location a, the probability of leaving the coverage area is less than α (i.e., $p_a^T < \alpha$). Then, I-Selector computes R_{new}, the computing resources of vehicles in $\mathcal{V}$ (Line 7). If there is no improvement in the amount of available resources, the algorithm stops; otherwise, it increases the value of T and continues this procedure as long as the amount of available resources for the new time period T is higher than that for the previous time period.

Complexity Analysis. The time complexity of I-Selector is $O(V \cdot F_e \cdot |\xi^T|)$. The main part of I-Selector consists of the loop in Lines 5-14, which executes $\frac{T}{T_{em}} = F_e$ times. In each iteration, we call satisfied-set which takes $O(|\xi^T| \cdot V)$ time. Therefore, the total time complexity of I-Selector is $O(V \cdot F_e \cdot |\xi^T|)$.

5.3.2 G-ERMP algorithm

G-ERMP operates in two phases using various *scenarios* to handle the chance constraint. Each scenario assumes a deterministic (i.e., known) location for vehicles. In the

first phase, the algorithm picks a random sample of scenarios generated based on a set of probable locations for the vehicles. Therefore, the algorithm solves the deterministic version of ERMP to obtain a solution for each scenario. Because the location within a scenario is known, this solution provides a single replica for each requester vehicle. Then, in the second phase, based on the assignments obtained for each scenario, the algorithm determines the number of replicas for each vehicle as well as their replica assignment so that Constraint (5.9) is satisfied with a probability higher than $(1 - \beta)$. However, the problem solved in the first phase of G-ERMP belongs to the class of packing problems, which are known to be NP-hard. Therefore, it is not solvable in polynomial time, unless P=NP. Thus, we first develop a greedy algorithm called GD-ERMP, that solves the problem associated with the first phase of G-ERMP for a selected scenario. Then, we describe the complete G-ERMP algorithm, which examines the solutions provided by GD-ERMP to finalize the selection of the providers and the number of replicas.

GD-ERMP algorithm

In order to minimize the maximum energy balance of vehicles, GD-ERMP analyzes the energy balance of each vehicle at the beginning of each period: vehicles with a low energy balance are more likely to be selected as the providers for the current period. However, if the decision is made only based on the energy balance, we might obtain a solution in which providers are distributed very irregularly. Thus, the decision maker needs to consider a high number of providers in the system so that all the selected requesters are covered within a reliable distance. In other words, making decisions only based on the energy balance may increase the number of providers. Thus, the leftover CPU capacity may not be used efficiently. To solve this problem, our algorithm considers both the location of the vehicles and their energy balance.

The algorithm defines a set of providers P, which is updated in an iterative manner. The algorithm starts with a minimum possible number of providers, i.e., $|P| = 1$. This value increases in the next iteration if the current providers are not able to provide a good

Algorithm 7 GD-ERMP

Input: Set of vehicles: $\mathcal{V} = \{(C_i, r_i, E_i^{blnc'})\}$
Scenario for the location of vehicles: ε

1: $i \leftarrow \text{argmin}_{j \in \mathcal{V}} E_j^{blnc'}$
2: $P \leftarrow \{i\}$
3: $stop \leftarrow \text{false}$
4: **while not** $stop$ **do**
5: $\quad stop \leftarrow \text{true}$
6: $\quad S_i \leftarrow \emptyset \quad \forall i \in P$
7: $\quad$ **for** $j \in \mathcal{V} - P$ **do**
8: $\quad\quad i \leftarrow$ find-provider(j, P, ε)
9: $\quad\quad$ **if** $i > 0$ **then**
10: $\quad\quad\quad S_i \leftarrow S_i \cup \{j\}$
11: $\quad\quad$ **else**
12: $\quad\quad\quad i \leftarrow \text{argmax}_{j \in \mathcal{V}-P}(\frac{l(j,P,\varepsilon)}{\bar{l}} - \frac{E_j^{blnc'}}{\bar{E}})$
13: $\quad\quad\quad P \leftarrow P \cup \{i\}$
14: $\quad\quad\quad stop \leftarrow \text{false}$
15: $\quad\quad\quad$ **break**
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end while**

Output: S, P

connectivity for all the requesters or they do not have enough resources to process the workloads. GD-ERMP stops when all requesters are allocated within a reliable distance.

Algorithm 7 shows the pseudo-code of GD-ERMP. It considers both the energy balance and the location of vehicles to decide the set of providers P and the assignment of the replicas. The inputs are the vector of vehicles with their request type r_i, capacity C_i, and their current energy balance $E_i^{blnc'}$, and a scenario ε for the location of vehicles in the current energy manager period. The outputs are the set of providers P and the set of replicas $\{S_i\}$ allocated to each provider i.

In order to determine the providers, GD-ERMP first picks a vehicle with the minimum energy balance, and puts it in the set of providers P (Lines 1-2). Then, in an iterative manner, other providers are added to P. For each vehicle j that is not selected as a provider, the algorithm calls find-provider to find the nearest provider i that (i) has *enough capacity* to serve the vehicle; (ii) is within a reliable distance, i.e., $l_{ij} \leq \delta$; (iii) works at a lower

frequency; and (iv) by offloading a part of workload of vehicle j to it the system achieves energy savings (Line 8). To determine the amount of offloading for vehicle j, find-provider considers the maximum possible amount of workload r'_j that can be processed on the remaining capacity of provider i. Then, based on Equations (5.10-5.13), it obtains the amount of energy saving for the system by this offloading. The positive energy saving means that provider i can serve the request of vehicle j; otherwise, the algorithm considers the possibility of offloading to other providers. If this procedure does not find such a provider, it returns a negative value; otherwise it returns the index of the provider. Thus, if find-provider finds such a provider, the requester is assigned to that provider and the replica set of that provider is updated (Lines 9-10). If the algorithm cannot allocate a request within a reliable distance, the current set of providers is not enough to satisfy all the requests. Hence, the algorithm needs to add another vehicle to the set of providers. The next provider is chosen so that it has a relatively low energy balance and it is far away from the already selected providers, which helps covering more requesters with a minimum number of providers. This strategy is implemented in Lines 12-13, where the algorithm picks a vehicle that has the maximum value of $\left(\frac{l(j,P,\varepsilon)}{\bar{l}} - \frac{E_j^{blnc'}}{\bar{E}}\right)$, where $l(j,P,\varepsilon)$ is the minimum distance of vehicle j from the set of selected providers under the scenario ε. $\bar{l}$ and $\bar{E}$ are the average distance over vehicles and the average energy balance over vehicles, respectively. The above procedure is repeated until all requests are allocated to providers that are within a reliable distance.

Complexity Analysis. The time complexity of GD-ERMP is $O(|\mathcal{V}|^3)$. The main part of GD-ERMP consists of the loop in Lines 4-14, which executes $|P|$ times. In each iteration, for each non-provider vehicle, finding the nearest provider among j providers will take $O(j)$ time. Therefore, the total time complexity of GD-ERMP is $\sum_{j=1}^{|P|}(|\mathcal{V}| - j) \cdot j = O(|\mathcal{V}|^3)$

G-ERMP algorithm

Algorithm 8 shows the pseudo-code of G-ERMP. The algorithm has as input the set of scenarios, ξ, the vector of vehicles with their request size, r_i, and their capacity, C_i. The output consists of the set of providers P and the set of replica's assignments S for the current energy manager period. The main idea of G-ERMP is to create a graph based on the replica allocations obtained for each scenario by the GD-ERMP algorithm. Each vertex of this graph represents a vehicle; each edge of the graph indicates a requester j to provider i assignment, weighted by the number of scenarios in which a request of j has been allocated to i by the GD-ERMP algorithm. Then, the algorithm partitions this graph into the set of providers and the set of requesters, and determines the number of replicas for each requester. This partitioning is done so that, for each vehicle, Constraint (5.9) is satisfied for more than $(1 - \beta) \cdot |\xi|$ scenarios.

G-ERMP starts with an empty set of providers P and empty set of replicas' assignments (Lines 1-2). In each iteration of the algorithm, these sets will be updated. Also, we define vector $\sigma = \{\sigma_i\}$ to store the number of scenarios in which Constraints (5.9) are satisfied for each vehicle i (Line 3). We define Γ as a set of vehicles for which Constraints (5.9) are satisfied. G-ERMP initializes Γ with the empty set (Line 4). Sets $\tilde{P}$ and $\tilde{S} = \{\tilde{S}_i\}$ are used to save the set of providers and the set of replicas obtained for each scenario by GD-ERMP (Lines 5-6). G-ERMP creates a graph with $|\mathcal{V}|$ vertices. Each vertex represents a vehicle and each edge indicates a request-provider assignment. The weight of an edge from vertex j to vertex i is denoted by w_{ji} and is defined as the number of scenarios in which vehicle j is assigned to vehicle i. The indegree of vertex i, i.e., the total weight of edges adjacent to vertex i, is stored in vector $indeg = \{indeg_i\}$ (Lines 7-10).

In order to find the minimum number of providers, in each iteration, G-ERMP selects the vehicle as the provider that has received the maximum number of requests from the various scenarios. Therefore, it choses the vertex with the maximum indegree as a provider (Line 12). Then, it updates the set of providers (Line 13). When a vehicle is selected as

Algorithm 8 G-ERMP

Input: Set of vehicles: $\mathcal{V} = \{(C_i, r_i)\}$
Set of scenarios :ξ

$P \leftarrow \emptyset$
$S_i \leftarrow \emptyset \quad \forall i \in \mathcal{V}$
$\sigma_i \leftarrow 0 \quad \forall i \in \mathcal{V}$
$\Gamma \leftarrow \emptyset$
for each $\varepsilon \in \xi$ **do**
    $(\tilde{S}, \tilde{P}) \leftarrow$GD-ERMP($\mathcal{V}$,$\varepsilon$)
    for each $i \in \tilde{P}$ **do**
        for each $j \in \tilde{S}_i$ **do**
            $w_{ji} \leftarrow w_{ji} + 1$
            $indeg_i \leftarrow indeg_j + 1$
        end for
    end for
end for
while $|\Gamma| < \mathcal{V}$ **do**
    $j \leftarrow \text{argmax}_{i \in \mathcal{V} \setminus P}\, indeg_i$
    $P \leftarrow P \cup \{j\}$
    $\Gamma \leftarrow \Gamma \cup \{j\}$
    for each $i \in P$ **do**
        if $j \in S_i$ **then**
            $S_i \leftarrow S_i - \{j\}$
            for each $k \in \mathcal{V} \setminus \Gamma$ **do**
                if $w_{ki} > 0$ **and** available(k,i,S) **then**
                $S_i \leftarrow S_i \cup \{k\}$
                $\sigma_k \leftarrow \sigma_k + w_{ki}$
                if $\sigma_k > (1 - \beta) \cdot |\xi|$ **then**
                $\Gamma \leftarrow \Gamma \cup \{k\}$
                end if
                end if
            end for
        end if
    end for
    for each $g \in \mathcal{V} \setminus \Gamma$ **do**
        if $w_{gj} > 0$ **and** available(g,j,S) **then**
            $S_j \leftarrow S_j \cup \{g\}$
            $\sigma_g \leftarrow \sigma_g + w_{gj}$
            if $\sigma_g > (1 - \beta) \cdot |\xi|$ **then**
                $\Gamma \leftarrow \Gamma \cup \{g\}$
            end if
        end if
    end for
end while

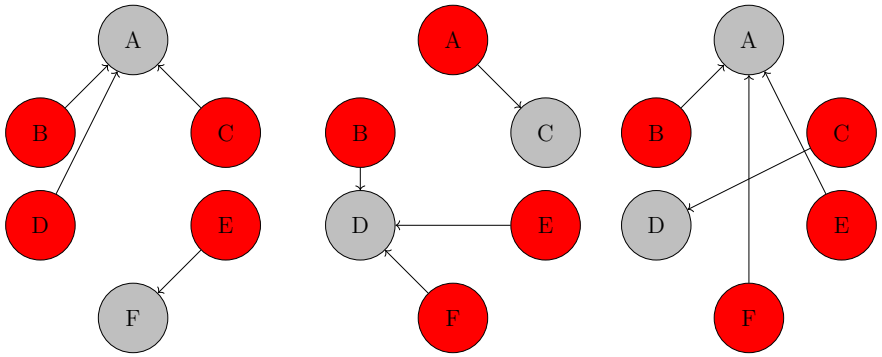
Output: S, P

a provider, it runs its requests locally, which means that, for this vehicle, Constraint (5.9) is automatically satisfied. Thus, it adds the current provider to the set Γ (Line 14). The algorithm then updates the replica assignment of vehicles in two steps. In the first step, since vehicle j is selected as a provider, the algorithm removes all the previous assignments from vehicle j on any provider. In fact, the algorithm checks if a request from vehicle j has been assigned to a provider i, it removes that assignment (Lines 16-17). Furthermore, since the remaining capacity of vehicle i is increased, it might be able to serve more requests. Thus, for each vehicle $k \in V - \Gamma$ willing to be assigned to vehicle i, the algorithm updates the assignment if vehicle i has enough capacity. It also updates σ for vehicle k. If σ_k is greater than $(1 - \beta) \cdot |\xi|$, the algorithm adds vehicle k to Γ. Therefore, the algorithm will not generate any further replica for that vehicle (Lines 18-23). In the second step, the algorithm assigns requests from each vehicle g willing to be assigned to vehicle j. It updates the assignment if vehicle i has enough capacity. It also updates σ for vehicle g. If σ_g is greater than $(1 - \beta) \cdot |\xi|$, the algorithm adds vehicle g to Γ (Lines 24-29). The algorithm continues this procedure until all the vehicles are added to set Γ.

Complexity Analysis. To investigate the time complexity of G-ERMP, we analyze the time complexity of the two main parts of the algorithm. In the first part, G-ERMP calls GD-ERMP for each scenario. Therefore, as analyzed in the previous section, the time complexity of the first part is $O(|\xi| \cdot |\mathcal{V}|^3)$. In the second part, G-ERMP builds a graph based on the solution obtained in the first part. The time complexity of the second part mainly depends on the loop in Lines 11-29, which executes $O(|\mathcal{V}|)$ times. The main part of the loop consists of the loop in Lines 15-23 which executes $O(|P| \cdot (|\mathcal{V} \setminus \Gamma|))$ times. Therefore, the time complexity of the second part is $O(|\mathcal{V}|^3)$. As a result, the total time complexity of G-ERMP is $O(|\xi| \cdot |\mathcal{V}|^3 + |\mathcal{V}|^3) = O(|\xi| \cdot |\mathcal{V}|^3)$

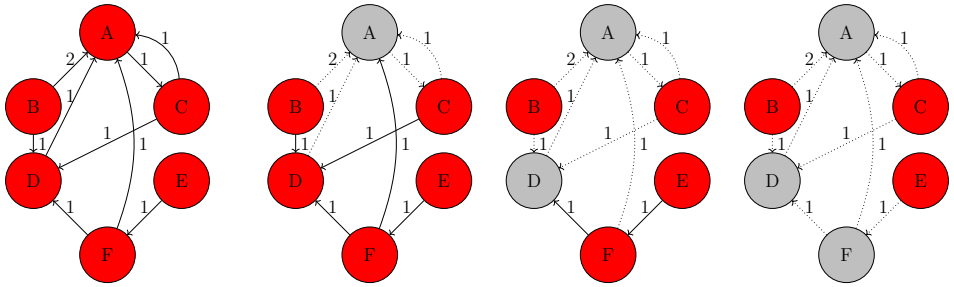
5.3.3 G-ERMP execution: example

Here, we provide an example to show how the G-ERMP algorithm works. We consider an area with six vehicles. We assume that there are three scenarios for the predicted



(a) (b) (c)

Figure 5.3: Example: A problem instance with six vehicles and three scenarios.



(a) (b) (c) (d)

Figure 5.4: Example: Replica placement obtained by G-ERMP

locations of the vehicles. In this example, we set $\beta = 0$. Therefore, a feasible solution is obtained when for each vehicle Constrain (5.9) is satisfied for all scenarios. In Figure 5.3, we show the three solutions obtained by GD-ERMP for each scenario. In the first scenario (Figure 5.3a), the request from vehicles B, C, and D are assigned to A and the request from E is assigned to F. In the second scenario (Figure 5.3b), requests from B, E, and F are assigned to D and the request from A is assigned to C. In the third scenario (Figure 5.3c), requests from B, F, and E are assigned to A and request from C is assigned to D. Now, we show how G-ERMP determines the set of requesters, the set of providers, and the replica assignment based on these solutions. Figure 5.4a shows the graph obtained by the solutions for each scenario. The weight on an edge from i to j indicates the number of scenarios that recommend an assignment from vehicle i to j. For example, the weight

on edge (B, A) is two because in two scenarios (Figures 5.3a and 5.3c), the request from vehicle A is assigned to B.

Figure 5.4b shows the first iteration of the algorithm. In this iteration, vehicle A, is selected as a provider because it has the maximum indegree. Replicas from B, D, and C are respectively assigned to A. The dotted arrows in the figures are used to highlight the edges that are not used anymore to compute the nodes' indegree. Note that due to the limited capacity, replicas from F are not assigned to A. Figure 5.4c shows the next iteration of the algorithm. In this iteration, vehicle D, which has the second-highest indegree, is selected as a provider. Thus, a replica from B and C is allocated to D. Again, due to the the limited capacity, vehicle F is not assigned to D. However, since D is a provider now, its previous assignment on A is removed and the capacity of A is updated. Now, there is enough capacity for A to serve replicas from F. Therefore, its corresponding edges are marked with dotted arrows. In the last iteration (Figure 5.4d), the last unsatissfied request from E is served by F, which is marked as provider since its indegree is higher than that of E.

5.4 Experimental analysis

In this section, we describe our experimental setup and then analyze the experimental results.

5.4.1 Experimental setup

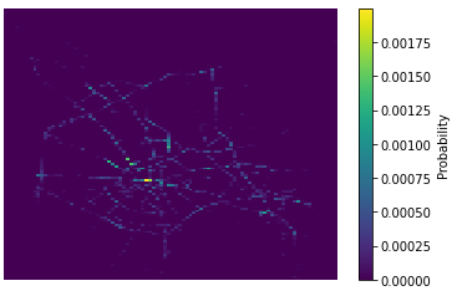
Computing Setup. Table 5.2 shows the parameters used to generate instances in our analysis. $U[x, y]$ indicates the uniform distribution within the interval $[x, y]$, and $N(x, y)$ indicates the normal distribution with mean x and variance y. We have tested VECMAN for various values of T_{em} but observed little variation on its general behavior. Thus, due to limited space, in the rest of the section we show the result only for $T_{em} = 10$ seconds. We assume that for each type of resources, the capacity of vehicles is in the same range and does not vary significantly. Therefore, we use the normal distribution for the memory and

Table 5.2: Distribution of parameters

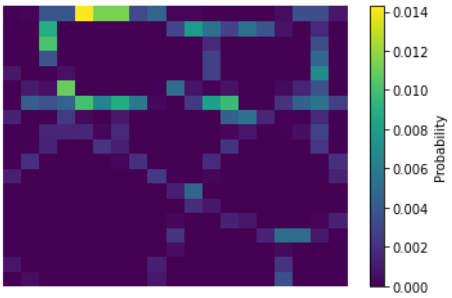
Parameter	Distribution/Value	Parameter	Value
f_i	light: $U[700, 1000]$	r_{2i}	$U[100, 1000]$
	moderate: $U[1100, 1400]$	r_{3i}	$U[100, 1000]$
	heavy: $U[1500, 1900]$	μ	3375
α, β	0.1	λ_i	0.00125
ω_i	0.2	ϕ_i	0.2
d_i	light: $U[500, 1000]$	θ_i	-4558.52
	moderate: $U[1000, 5000]$	ϑ_i	7.683
	heavy: $U[5000, 9000]$	γ_i	-0.741625
C_{2i}	$N[5000, 500]$	T_{em}	10
C_{3i}	$N[5000, 500]$		

storage capacity of vehicles. According to Equation (5.8), the CPU capacity of a vehicle depends on its frequency and the size of the local workload. We use as an example CPU ARM Cortex A57 that has 13 frequency levels from 700 MHz to $1,900$ MHz. We profile the Cortex A57 in terms of MIPS and power consumption for each frequency level to get the model parameters in Equations (5.8) and (5.10). We define three types of instances with light, moderate, and heavy workloads. For light workloads, the default frequency varies from 700 to 1000, while for moderate and heavy cases, it varies from 1100 to 1400 and 1500 to 1900, respectively. For all instances the required time to execute workloads is uniformly drawn from $[1, 10000]$ milliseconds. Thus, by using Equation (5.6), the number of CPU instructions for each instance is $r_{1i} = (\vartheta_i \cdot f_i + \theta_i) \cdot \frac{U[1,10000]}{1000}$ and the CPU capacity is $C_{1i} = (\vartheta_i \cdot f_i + \theta_i) \cdot T_{em} - (\vartheta_i \cdot f_i + \theta_i) \cdot \frac{U[1,10000]}{1000}$. According to this equation and since the execution time of all types of workloads is in the same range, the expected CPU capacity of vehicles with heavy workloads is higher than that of lighter workloads.

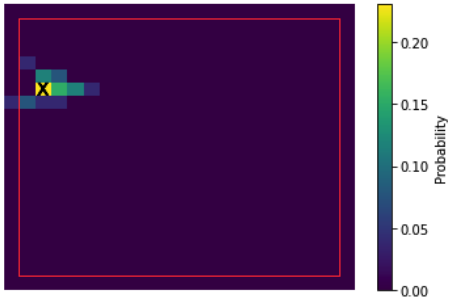
We estimate the transmission energy parameters ω_i and ϕ_i based on the analysis provided in [118]. We also set the value of μ to 27Mb/s (equivalent to 3.375MB/s) which is the maximum data transmission in DSRC networks [119]. Since the length of energy manager period is 10 seconds, the maximum size of data that can be transmitted within an energy manager period in a unit of distance is 33.75MB. Considering the computational



(a) Distribution of vehicles over a region of Cologne city.



(b) Distribution of vehicles over the most congested area.



(c) An example of vehicles' mobility model.

Figure 5.5: Scenario generation model.

time needed to execute workload on vehicles within each T_{em}, to guarantee QoS, we define three problem instances with light, moderate, and heavy amount of data. In these instances the size of data varies from 0.5MB to 9MB.

RSU Coverage Area Setup. We use the dataset of vehicular mobility in the city of Cologne, Germany [115]. The dataset contains the traces of vehicles over a region of 400 square kilometers during the 24 hours of a working day with a granularity of one second. Due to the large amount of data in the dataset, we generate vehicle mobility scenarios using only data during the rush hour (i.e., 7:30am-8:30am) to stress-test VECMAN when the maximum number of vehicles are in traffic. We assume that the coverage range of the RSU is 1 kilometer, which is similar to the radius of the DSRC [110]. Thus, we consider a 2 kilometers by 2 kilometers area of Cologne that has the heaviest traffic during the selected rush hour and assume the RSU is at the center of this area. For simplicity, we consider the area as a two-dimensional grid of 20×20 cells in which the size of each cell is 100 meters

by 100 meters. Figure 5.5a shows the traffic over the city of Cologne during rush hour and Figure 5.5b shows the traffic of the most congested area selected for our tests. We observe that on average, in every second, there are 720 vehicles in the RSU coverage area. Thus, we set the number of vehicles V that are initially in the coverage area to 720. For each cell of the RSU coverage area, we obtain the average number of vehicles that are located in that cell. We use this number as a probabilistic parameter to initialize the distribution of vehicles across the coverage area.

I-Selector Probability Vector. According to the I-Selector Algorithm, we need to consider various lengths of resource selection periods to determine the length that maximizes the objective function (i.e., Equation 5.1). For each candidate length T, for each cell in the RSU coverage area, and given the vehicles initial locations when the I-Selector is invoked, we obtain the probability for each vehicle to leave the RSU coverage area before T seconds. For this purpose, we consider the vehicles' location records for each T seconds time intervals in the dataset. Based on these records, we obtain the total number of vehicles that are initially located in each cell, but they leave the coverage area by time slot T. For each cell, we then divide this number to the total number of vehicles that are initially located in the cell, which is the probability for each vehicle in each cell of leaving the RSU coverage area by the end of T.

Scenario Generation for G-ERMP. We generate scenarios for the future locations of vehicles based on the current location of vehicles and the movement probabilities of vehicles between the cells of the grid area. To determine the probability that a vehicle moves from one cell A to cell B in each energy manager period, we consider the number of movements from cell A to cell B divided by the total number of departures from cell A. Figure 5.5c shows the movement probability from the cell marked with X to other cells as a heat map. The red square shows the RSU coverage area. In this example there is a low probability a vehicle will leave the RSU coverage area. These probabilities are then used to generate the

various possible scenarios.

Performance Metrics. The performance of VECMAN is evaluated by computing the percentage of total energy savings, which is defined as the ratio between the total energy savings of vehicles and the total baseline energy consumption (i.e., vehicles run their requests locally).

$$\text{ES}(\%) = 100 \cdot \frac{\sum_{i \in \mathcal{V} \setminus P} (\vartheta_i \cdot f_i + \theta_i) \cdot n_i}{\sum_{i \in \mathcal{V}} (\vartheta_i \cdot f_i + \theta_i) \cdot n_i} \tag{5.17}$$

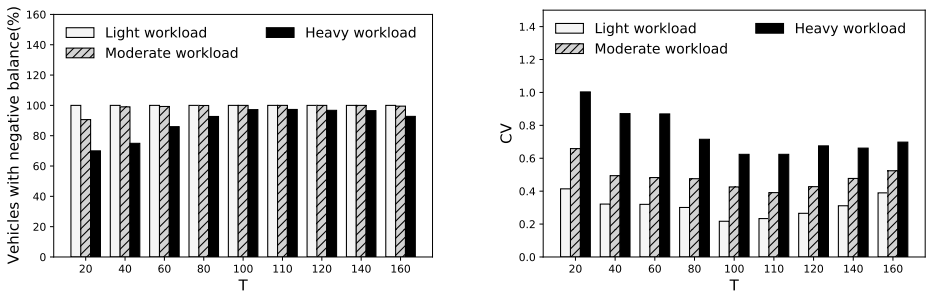
To evaluate the fairness of VECMAN, we determine the Coefficient of Variation (CV) over energy balance of the vehicles. A lower value of CV means a more fair distribution of requests. CV is defined as the ratio of the standard deviation of E_i^{blnc} over the average energy balance across vehicles $\bar{E}$,

$$CV = \frac{\sqrt{\frac{1}{\mathcal{V}} \sum_{i=1}^{V} (E_i^{blnc} - \bar{E})^2}}{\bar{E}} \tag{5.18}$$

G-ERMP and I-Selector are implemented in C++ and executed on an Intel 1.6GHz Core i5 with 8 GB RAM.

5.4.2 Experimental results

In this section, we first investigate the impact of the length of resource selection period on the fairness and the energy balance of vehicles. We show that the length obtained by I-Selector algorithm yields a fair distribution of workloads and a low energy balance among the vehicles compared to other possible lengths of resource selection period. Then, we investigate the performance of the VECMAN compared to the baseline that executes workload of each vehicle locally for the light, moderate, and heavy workload instances. Next, we investigate the performance of VECMAN compared to the baseline for instances with light, moderate, and heavy data transmission sizes. Finally, we investigate the scalability of VECMAN compared to a baseline that only offloads vehicles' workload to the



(a) Percentage of vehicles with a negative energy balance.

(b) Coefficient of Variation (CV) of energy balance.

Figure 5.6: Performance for various lengths of the resource selection period.

local RSU while changing the number of vehicles .

Impact of the resource selector

We run the I-Selector algorithm on the set of vehicles that are initially located in the coverage area. To investigate the efficiency of the resource selection, we compare the fairness and energy balance of the participating vehicles obtained for various lengths of resource selection period. We vary the value of T from 20 seconds to 160 seconds. For each length T, we obtain the participating vehicles who for the next T units of time stay in the area with probability not less than $1 - \alpha$ and periodically execute our G-ERMP algorithm using the selected $T_{em} = 10$ seconds. For fairness of comparison, we run all the experiments over a fixed time interval (200 seconds) and then, for each baseline, average the results across resource-selection intervals.

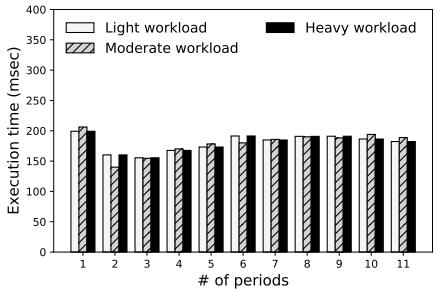
Figures 5.6a and 5.6b show the average percentage of vehicles with negative energy balance and average coefficient of variation at the end of each resource selection period, respectively. As the figures show, the optimal resource selection period that ensures both fairness and high energy savings for most participating vehicles is $T = 110$ seconds, which is also the period selected by the I-Selector algorithm of VECMAN. The corresponding number of vehicles selected for participation is 374. A higher energy unbalance across participating vehicles is observed for $T < 110$ and $T > 110$, specially for heavy workload type. Specifically, when $T < 110$ the energy manager runs fewer times during each

resource selection period, therefore giving no time to some participating vehicles to become both requester and provider before the period expiration. As a result, during each resource selection interval some vehicles may have been only in requester mode, thus enjoying energy savings, while some others may have been only in provider mode, which leads to no energy savings, i.e., unfairness. On the other hand, when $T > 110$ the number of vehicles that are likely to remain in the RSU coverage area and are therefore selected for participation decreases, which leads to a lower amount of shared resources during each resource selection period. This low resource availability leads to a higher imbalance in energy savings across vehicles because a fewer number of workloads can be placed on the lower number of available providers.

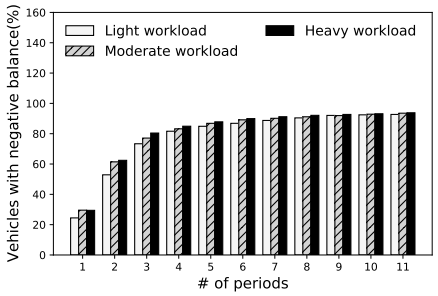
These experiments show that the I-Selector algorithm helps G-ERMP provide fair energy savings among the participating vehicles.

Performance vs. workload types

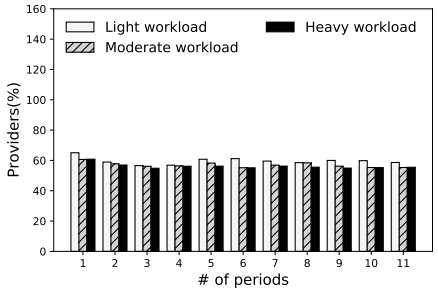
As we discussed in previous section, the resource selector chooses 374 vehicles to participate in resource sharing with optimal period $T = 110$ seconds. Here we analyze the performance of VECMAN by considering these vehicles and run the G-ERMP in 11 energy-manager periods. We consider three sets of workload instances: light, moderate, and heavy. In these instances, the size of transmission data of vehicles is moderate. Figure 5.7a shows that the average execution time of G-ERMP for each problem instance is less than 0.2 second, which is negligible compared to the execution period of the requests ($T_{em} = 10$ seconds). The execution times for the three types of workloads are almost the same. For some of the periods, the execution time of G-ERMP for problem instances with moderate workload is slightly higher than that for problem instances with heavy workload. The reason is that for some periods, in the case of moderate workload instances, the percentage of providers that do not execute only their local workload is slightly higher than in the case of high workload instances. Thus, the graph generated by G-ERMP for problem instances with moderate workload is slightly more complex compared to the graph gener-



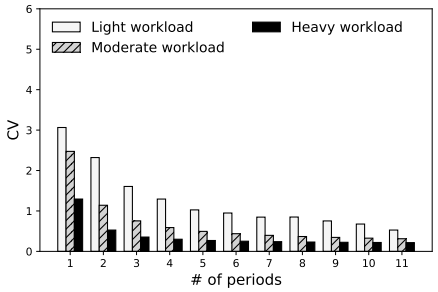
(a) Execution time of G-ERMP.



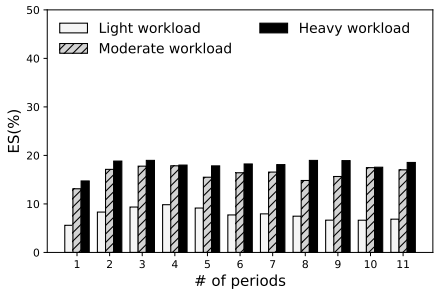
(b) Percentage of vehicles with negative balance.



(c) Percentage of vehicles selected as providers.



(d) Coefficient of Variation (CV) of energy balance.



(e) Percentage of energy savings.

Figure 5.7: Performance with respect to the workload types.

ated for instances with heavy workload. As a result, the time needed to partition the graph is slightly higher in the case of instances with moderate workloads.

As Figure 5.7b shows, the percentage of vehicles with a negative balance increases over the periods. For all problem instances, after all periods, more than 92% of vehicles obtain energy savings. However, for problem instances with heavy workload, we observe a higher percentage of vehicles achieving energy savings compared to the moderate and light workloads. The reason is that in average, the CPU capacity of instances with heavy workload is

higher than that with lighter workloads (see section 5.4.1). Thus, as Figure 5.7c shows, the percentage of providers decreases compared to the light and moderate workloads. Consequently, more energy savings are achieved for these instances, as Figure 5.7e shows; and a higher percentage of vehicles achieve energy savings. Furthermore, for problem instances with heavy workload, as Figure 5.7d shows, the CV value is generally less than that of the lighter workloads because fewer vehicles have to process their requests locally. This leads to having higher vehicles achieving energy savings. On the other hand, as Figure 5.7e shows, even for the light-workload case the vehicles can achieve about 7% more energy savings compared to the baseline, while the moderate and heavy workloads achieve 16% and 18% energy savings, respectively.

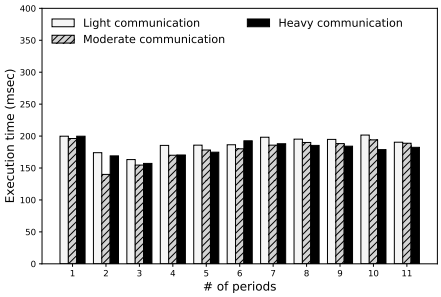
Note that some vehicles, e.g., 8% in the above experiments, may not achieve energy savings in the current resource sharing period. However, they may achieve energy savings in the next sharing periods. It is in our future work to design an algorithm that considers the initial energy balance of each vehicle to ensure that all vehicles can achieve energy savings over multiple resource selection periods.

These experiments show that VECMAN enables vehicles to achieve energy savings for various workload instances.

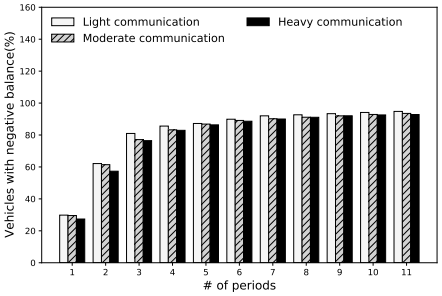
Performance vs. data size

In this experiment, we investigate the effect of the size of transmitted data on the performance of VECMAN. We consider three types of problem instances, light, moderate, and heavy communication. We assume that the workload of the vehicles is moderate.

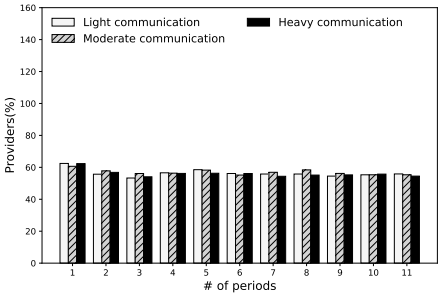
Figure 5.8a shows the execution time of G-ERMP for the three types of instances. As we observe, the size of data does not affect the execution time of the algorithm. For all instances the execution time of the algorithm is less than 0.2 seconds. Figure 5.8b shows the percentage of vehicles with negative balance. We observe that for problem instances with light communication more vehicles achieve energy savings compared to the moderate and heavy communication. Figure 5.8c shows the percentage of vehicles that are selected



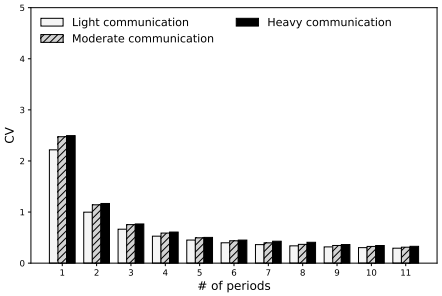
(a) Execution time of G-ERMP.



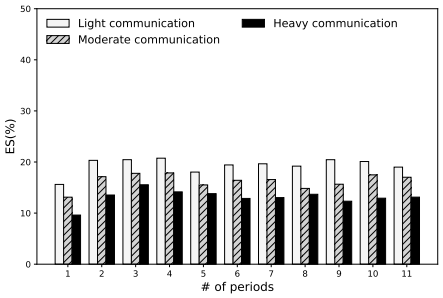
(b) Percentage of vehicles with negative balance.



(c) Percentage of vehicles selected as providers.



(d) Coefficient of Variation (CV) of energy balance.



(e) Percentage of energy savings.

Figure 5.8: The effect of data transmission on the performance.

as providers over the energy manager periods. As the figure shows, the percentage of providers does not change much with the increase in the data size. The reason is that the number of providers does not depend on the size of the data transmitted by the vehicles, but it depends on how the transmitted data size varies among the vehicles. Since in all types of problem instances, the size of the data transmitted follows the uniform distribution, the percentage of providers will not change. Figure 5.8d shows the CV of the energy balance over energy manager periods. Due to the fact that, in the case of heavy communication

vehicles achieve energy savings in a slower manner, the CV value is higher than the CV value for the light and moderate communication instances. However, as Figure 5.8e shows, even for the heavy communication the vehicles achieve an average of 13% energy savings compared to the baseline.

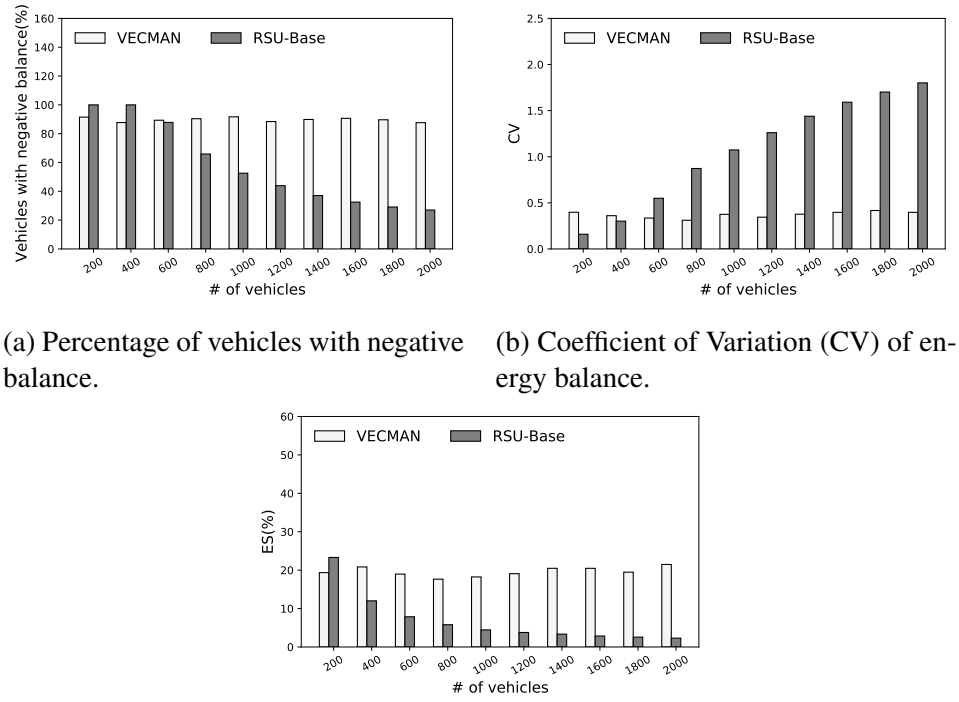
These experiments show that VECMAN enables vehicles to achieve energy savings for various instances with different data sizes.

Performance vs. number of vehicles

Here, we investigate the scalability of VECMAN with respect to the number of vehicles and compare it to a baseline called RSU-Base, which uses the local RSU to run the requests of the vehicles. RSU-Base orders the vehicle requests in descending order based on the vehicles' energy balance and then, starting from the one with highest balance, it allocates as many requests as possible on the RSU's resources.

As discussed in Section 5.4.1, the estimated number of vehicles that are initially in the coverage area is 720. But, here we vary the number of vehicles from 200 to 2000 to investigate the scalability of the algorithm. We use the scenario generation model and resource selection setup described in Section 5.4.1 to generate problem instances. The vehicles' workload and the size of data is moderate. As Figure 5.9a shows, VECMAN enables about 90% of the vehicles to obtain energy savings while RSU-Base, because of its limited resources, cannot achieve energy savings for more than 50% of the vehicles when there are more than 1000 vehicles. As a result, as Figure 5.9b shows, VECMAN provides a fair distribution of the energy savings (decreasing CV value) while RSU-Base has an unbalanced savings distribution (increasing CV value) due to the limited number of vehicles that can offload their workload. This behavior, as Figure 5.9c shows, translates in a stable 19% energy savings with VECMAN and a decreasing amount of savings for RSU-Base with an increasing number of vehicles.

These experiments show that VECMAN enables vehicles to achieve energy savings independently from the number of vehicles it manages.



(a) Percentage of vehicles with negative balance.

(b) Coefficient of Variation (CV) of energy balance.

(c) Percentage of energy savings.

Figure 5.9: The effect of number of vehicles on the performance.

5.5 Conclusion

In this chapter, we proposed VECMAN, an energy-aware resource management framework for VEC systems with the aim of minimizing the energy consumption of the participant vehicles. We evaluated VECMAN by performing an extensive experimental analysis on several problem instances. The results showed that the proposed framework allows vehicles to achieve between 7% and 18% computational energy savings compared to a baseline that executes workload locally and 13% energy savings compared to a baseline that offloads vehicles' workloads only to RSUs.

CHAPTER 6

CONCLUSION AND FUTURE RESEARCH

The widespread usage of mobile devices generates an unprecedented amount of data that often requires real-time processing. Mobile Edge Computing (MEC) is one of the promising paradigm to provide the required infrastructure for low latency computing services through running mobile applications at the edge of the network, where an edge can be any computing resources of the network. In MEC, the edge nodes are widely distributed in the network and available to all mobile users. However, compared to the cloud data centers, edge nodes have more restricted capacity. Therefore, it might not be feasible to run large size applications on a single edge node, and finding an efficient placement for the components of an application on the multiple nodes is a major challenge. Another challenge stems from the fact that mobile users change their locations dynamically and the current assignment of an application to the edge nodes might not be the best in terms of the costs involved. In addition to the mobility of users, the resource availability and network conditions may also change dynamically. Therefore, in order to provide high quality services with the minimum costs, the application may need to migrate from one edge/core node to another, dynamically. When it comes to leveraging MEC for intensive computations in Electric Connected Autonomous Vehicles (eCAVs), due to the limited capacity of RSUs, some computational tasks may experience poor Quality of Service (QoS) or even failure. Monetization of services, that is, developing incentive schemes for mobile users and edge providers, is another significant challenge in the development of MEC systems. Decentralized distribution of MEC servers, heterogeneity of resource requirements, and the competition between users to acquire high quality services make the resource allocation and pricing in MEC systems a challenging problem.

6.1 Our contributions

To tackle the mentioned challenges and based on the current gap in the literature, we made several contributions in the area of resource management in MEC systems. Our

contributions consist of designing efficient resource allocation algorithms that are suitable for real world edge systems while considering major performance metrics such as energy consumption and QoS. One of the problems addressed in this dissertation was the multi-component application placement problem in MEC systems. We considered an area managed by an edge provider with heterogeneous servers that periodically runs a resource manager. The user requests to offload an application with a set of components. Each component is characterized by its processing requirement and its communication with user/other components. In this problem, the *objective* is to find an assignment between components and servers, such that the total placement cost of the application is minimized. The total placement cost is composed of four types of costs: (i) the cost of running a component on a server; (ii) the relocation cost; (iii) the component-user communication cost; and, (iv) the inter-component communication cost. We proved that the problem is NP-hard. In order to solve the problem efficiently, we developed two algorithms, one based on matching and local search and one based on a greedy approach. We evaluated the performance of the proposed algorithms by an extensive experimental analysis. The experiments were driven by two types of user mobility models, one derived from real-life mobility traces and the other one based on the random-walk model. Our experimental results showed that the proposed algorithms obtain solutions that are very close to the optimal and require very low execution time (less than a millisecond). Also, the performance of both algorithms was consistent under both the trace-driven mobility data set and the random walk model which indicates that the proposed algorithms are relatively robust to the mobility behavior of users. The results of this research are published in Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC-2017) [16], IEEE Transactions on Cloud Computing (TCC) [17], and the Proceedings of the International Conference on Edge Computing (EDGE-2019) [18].

We also addressed Allocation and Pricing of Resources in MEC Systems. We developed two resource allocation and pricing mechanisms in MEC systems, where users have

heterogeneous requests and compete for high quality services. First, we developed a novel auction-based mechanism that combines features from both position and combinatorial auctions and handles heterogeneous resource requests from mobile users and heterogeneous types of resources. It determines envy-free allocations (i.e., allocations in which no user can improve her utility by exchanging bids with any user with the same request for resources) and prices that lead to close to optimal social welfare for the users. We also proposed an LP-based approximation mechanism that does not guarantee envy-freeness, but it provides solutions that are guaranteed to be within a given distance from the optimal solution. We evaluated the efficiency of the algorithms by performing an extensive experimental analysis. For small-size instances, we compared the solutions obtained by the proposed mechanisms with the optimal solutions obtained by the CPLEX solver with respect to execution time, percentage of served users, social welfare and revenue. For large-size instances, we compared the performance of the two proposed mechanisms with respect to the same metrics used for the analysis on small-size instances. The experimental results showed that the resource allocation obtained by the proposed mechanisms yield near optimal solutions. In addition to the quality of solutions, the small execution time makes the proposed mechanisms promising for deployment in EC systems. The results of this research are published in the Proceedings of the Third ACM/IEEE Symposium on Edge Computing (SEC-2018) [19] and in the IEEE Transactions on Parallel and Distributed Systems [20].

Another contribution of this dissertation is designing VECMAN, an energy-aware resource management framework for (VEC) systems with the aim of minimizing the energy consumption of the participant vehicles. In this study, we proposed VECMAN that manages the computing nodes of connected electric vehicles for minimized energy consumption. VECMAN consists of two algorithms: (i) a resource selector algorithm that runs periodically on the local RSU and determines the set of participating vehicles as well as the duration of resource sharing; and (ii) an energy manager algorithm that runs periodically at

a finer time grain within each resource sharing time period and determines the state of each participating vehicle, i.e., requester or provider, the number of replicas for the requesters' workloads, and the amount of requester's workload to offload so that energy consumption is minimized for all the participating vehicles. We evaluated VECMAN by performing an extensive experimental analysis on several problem instances. The results showed that the proposed framework allows vehicles to achieve between 7% and 18% computational energy savings compared to a baseline that executes workload locally and 13% energy savings compared to a baseline that offloads vehicles' workloads only to RSUs. The results of this research are published in the Proceedings of the International Conference on Cloud Engineering (IC2E-2020) [21], IEEE Transactions on Mobile Computing [22], and the Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking (EdgeSys-2019) [23].

6.2 Future research

As a future research on application placement, we plan to design placement algorithms that take into account both the users' and providers' economic incentives when making placement decisions. A direct extension of this work is to consider settings in which a subset of the components of a single application are offloaded to a single server. In the area of the resource allocation and pricing mechanisms, we plan to design and implement resource allocation and pricing mechanisms for edge computing systems with different network structures. In this research, we assumed that a user is allocated either at the cloud or edge level, but not at both. One direction for future research is to allow allocation of a user request at both the edge and the cloud level. Another possible direction for future research is considering a setting with multiple edge providers. We also plan to improve VECMAN to (i) theoretically guarantee the minimum computational energy consumption for any data transmission size, (ii) explore the possibility of collaboration between RSUs, and (iii) allow provider vehicles to temporarily increase their default computing frequency level.

REFERENCES

[1] *Mobile cloud computing forum*, `www.mobilecloudcomputingforum.com`, (accessed October 5, 2021).

[2] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[3] M. Satyanarayanan, "A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets," *GetMobile: Mobile Comp. & Comm.*, vol. 18, no. 4, pp. 19–23, 2015.

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

[5] S. Wang, G.-H. Tu, R. Ganti, T. He, K. Leung, H. Tripp, K. Warr, and M. Zafer, "Mobile micro-cloud: Application classification, mapping, and deployment," in *Annual Fall Meeting of ITA (AMITA)*, 2013.

[6] NSF, "NSF workshop report on grand challenges in edge computing," 2016.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. 1st Edition of the MCC Workshop on Mobile Cloud Comp.*, ACM, 2012, pp. 13–16.

[8] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[9] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[10] M. Satyanarayanan, "The emergence of edge computing," *IEEE Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[11] M. T. Beck, M. Werner, S. Feld, and S Schimper, "Mobile edge computing: A taxonomy," in *Proceedings of the Sixth International Conference on Advances in Future Internet*, 2014, pp. 48 –54.

[12] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, 2002.

[13] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang, "Toward cloud-based vehicular networks with efficient resource management," *IEEE Network*, vol. 27, no. 5, pp. 48–55, 2013.

[14] E. Farhangi Maleki, L. Mashayekhy, and S. M. Nabavinejad, "Mobility-aware computation offloading in edge computing using machine learning," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[15] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 909–922, 2020.

[16] T. Bahreini and D. Grosu, "Efficient placement of multi-component applications in edge computing systems," in *Proc. 2nd ACM/IEEE Symp. on Edge Computing*, 2017, 5:1–5:11.

[17] T. Bahreini and D. Grosu, "Efficient algorithms for multi-component application placement in mobile edge computing," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2020.

[18] T. Bahreini, H. Badri, and D. Grosu, "Energy-aware capacity provisioning and resource allocation in edge computing systems," in *International Conference on Edge Computing*, Springer, 2019, pp. 31–45.

[19] ——, "An envy-free auction mechanism for resource allocation in edge computing systems," in *ACM/IEEE Symp. on Edge Computing*, 2018, pp. 313–322.

[20] ——, "Mechanisms for resource allocation and pricing in mobile edge computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 667–682, 2022.

[21] T. Bahreini, M. Brocanelli, and D. Grosu, "Energy-aware resource management in vehicular edge computing systems," in *2020 IEEE International Conference on Cloud Engineering (IC2E)*, 2020, pp. 49–58.

[22] ——, "VECMAN: A framework for energy-aware resource management in vehicular edge computing systems," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[23] ——, "Energy-aware speculative execution in vehicular edge computing systems," in *Proc. 2nd International Workshop on Edge Systems, Analytics and Networking*, 2019, pp. 18–23.

[24] D. Dutta, M. Kapralov, I. Post, and R. Shinde, "Embedding paths into trees: Vm placement to minimize congestion," in *European Symposium on Algorithms*, Springer, 2012, pp. 431–442.

[25] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. Networking*, vol. 20, no. 1, pp. 206–219, 2012.

[26] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, 2012.

[27] F. Berg, F. Dürr, and K. Rothermel, "Optimal predictive code offloading," in *Proc. 11th Int. Conf. on Mobile and Ubiquitous Syst.: Computing, Networking and Services*, ICST, 2014, pp. 1–10.

[28] J. Kwak, Y. Kim, J. Lee, and S. Chong, "Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE J. on Selected Areas in Comm.*, vol. 33, no. 12, pp. 2510–2523, 2015.

[29] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "A sample average approximation-based parallel algorithm for application placement in edge computing systems," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 2018, pp. 198–203.

[30] M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," *arXiv preprint arXiv:1704.06749*, 2017.

[31] R. Kaewpuang, D. Niyato, P. Wang, and E. Hossain, "A framework for cooperative resource management in mobile cloud computing," *IEEE J. Selected Areas in Comm.*, vol. 31, no. 12, pp. 2685–2700, 2013.

[32] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Trans. Vehicular Technology*, vol. 66, no. 4, pp. 3435–3447, 2017.

[33] A. Ksentini, T. Taleb, and M. Chen, "A markov decision process-based service migration procedure for follow me cloud," in *Proc. IEEE Int. Conf. on Comm.*, 2014, pp. 1350–1354.

[34] T. Taleb and A. Ksentini, "Follow me cloud: Interworking federated clouds and distributed mobile networks," *IEEE Network*, vol. 27, no. 5, pp. 12–19, 2013.

[35] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205 –228, 2015.

[36] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. IFIP Networking Conf.*, IEEE, 2015, pp. 1–9.

[37] S. Wang, M. Zafer, and K. K. Leung, "Online placement of multi-component applications in edge computing environments," *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[38] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system," *IEEE Trans. Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2179–2192, 2018.

[39] Z. Zhou, Q. Wu, and X. Chen, "Online orchestration of cross-edge service function chaining for cost-efficient edge computing," *IEEE J. Selected Areas in Comm.*, vol. 37, no. 8, pp. 1866–1880, 2019.

[40] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Computing*, vol. 5, no. 4, pp. 725–737, 2017.

[41] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Trans. Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2866–2880, 2016.

[42] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Trans. Networking*, vol. 25, no. 3, pp. 1818–1831, 2017.

[43] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *J. Parallel and Distributed Computing*, vol. 127, pp. 160 –168, 2019.

[44] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundancy scheduling for microservice-based applications at the edge," *IEEE Trans. on Services Computing*, 2020.

[45] S. Deng, Z. Xiang, P. Zhao, J. Taheri, H. Gao, J. Yin, and A. Y. Zomaya, "Dynamical resource allocation in edge for trustable internet-of-things systems: A reinforcement learning method," *IEEE Trans. on Industrial Informatics*, vol. 16, no. 9, pp. 6103–6113, 2020.

[46] S. Deng, Z. Xiang, J. Taheri, K. A. Mohammad, J. Yin, A. Zomaya, and S. Dustdar, "Optimal application deployment in resource constrained distributed edges," *IEEE Trans. on Mobile Computing*, 2020.

[47] Z. Xiang, S. Deng, J. Taheri, and A. Zomaya, "Dynamical service deployment and replacement in resource-constrained edges," *Mobile Networks and Applications*, vol. 25, no. 2, pp. 674–689, 2020.

[48] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *IEEE Wireless Comm. and Netw. Conf.*, 2017, pp. 1–6.

[49] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[50] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ACM, 2010, pp. 49–62.

[51] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. on Vehicular Techn.*, vol. 68, no. 1, pp. 856–868, 2018.

[52] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ACM, 2011, pp. 43–56.

[53] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Trans. on Communications*, vol. 67, no. 6, pp. 4132–4150, 2019.

[54] Y. Qian, L. Hu, J. Chen, X. Guan, M. M. Hassan, and A. Alelaiwi, "Privacy-aware service placement for mobile edge computing via federated learning," *Information Sci.*, vol. 505, pp. 562–570, 2019.

[55] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[56] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE Conf. on Computer Communications*, 2019, pp. 10–18.

[57] L. Mashayekhy, M. M. Nejad, and D. Grosu, "Physical machine resource management in clouds: A mechanism design approach," *IEEE Trans. Cloud Computing*, vol. 3, no. 3, pp. 247–260, 2015.

[58] S. Zaman and D. Grosu, "Combinatorial auction-based allocation of virtual machine instances in clouds," *Journal of Parallel and Distributed Computing*, vol. 73, no. 4, pp. 495–508, 2013.

[59] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: A randomized auction approach," in *IEEE Conf. on Computer Communications*, 2014, pp. 433–441.

[60] Z. Xiong, S. Feng, D. Niyato, P. Wang, and Z. Han, "Edge computing resource management and pricing for mobile blockchain," *arXiv preprint arXiv:1710.01567*, 2017.

[61] Y. Jiao, P. Wang, D. Niyato, and Z. Xiong, "Social welfare maximization auction in edge computing resource allocation for mobile blockchain," in *Proc. IEEE Int. Conf. on Comm.*, 2018, pp. 1–6.

[62] N. C. Luong, Z. Xiong, P. Wang, and D. Niyato, "Optimal auction for edge computing resource management in mobile blockchain networks: A deep learning approach," in *Proc. IEEE Int. Conf. on Communications*, 2018, pp. 1–6.

[63] L. Li, M. Siew, T. Q. Quek, J. Ren, Z. Chen, and Y. Zhang, "Learning-based priority pricing for job offloading in mobile edge computing," *arXiv preprint arXiv:1905.07749*, 2019.

[64] B. Baek, J. Lee, Y. Peng, and S. Park, "Three dynamic pricing schemes for resource allocation of edge computing for iot environment," *IEEE Internet of Things Journal*, 2020.

[65] Y. Chen, Z. Li, B. Yang, K. Nai, and K. Li, "A stackelberg game approach to multiple resources allocation and pricing in mobile edge computing," *Future Generation Computer Systems*, 2020.

[66] A. Kiani and N. Ansari, "Toward hierarchical mobile edge computing: An auction-based profit maximization approach," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2082–2091, 2017.

[67] Z. Kong, C.-Z. Xu, and M. Guo, "Mechanism design for stochastic virtual resource allocation in non-cooperative cloud systems," in *IEEE Int. Conf. Cloud Computing*, 2011, pp. 614–621.

[68] L. Mashayekhy, M. M. Nejad, and D. Grosu, "A ptas mechanism for provisioning and allocation of heterogeneous cloud resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 9, pp. 2386–2399, 2015.

[69] Y. Zhang, M. Li, K. Bai, M. Yu, and W. Zang, "Incentive compatible moving target defense against vm-colocation attacks in clouds," in *IFIP Int. Information Security Conf.*, Springer, 2012, pp. 388–399.

[70] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. New York, NY, USA: Cambridge University Press, 2007.

[71] S. Lahaie, D. M. Pennock, A. Saberi, and R. V. Vohra, "Sponsored search auctions," *Algorithmic game theory*, vol. 1, pp. 699–716, 2007.

[72] B. Edelman, M. Ostrovsky, and M. Schwarz, "Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords," *American Economic Review*, vol. 97, no. 1, pp. 242–259, 2007.

[73] K. Zheng, H. Meng, P. Chatzimisios, L. Lei, and X. Shen, "An smdp-based resource allocation in vehicular cloud computing systems," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 12, pp. 7920–7928, 2015.

[74] C. Yang, Y. Liu, X. Chen, W. Zhong, and S. Xie, "Efficient mobility-aware task offloading for vehicular edge computing networks," *IEEE Access*, vol. 7, pp. 26 652–26 664, 2019.

[75] Z. Jiang, S. Zhou, X. Guo, and Z. Niu, "Task replication for deadline-constrained vehicular cloud computing: Optimal policy, performance analysis, and implications on road traffic," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 93–107, 2018.

[76] 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 *IEEE International Conference on Sensing, Communication and Networking*, 2018.

[77] Y. Sun, J. Song, S. Zhou, X. Guo, and Z. Niu, "Task replication for vehicular edge computing: A combinatorial multi-armed bandit based approach," *arXiv preprint arXiv:1807.05718*, 2018.

[78] J. Zhou, D. Tian, Y. Wang, Z. Sheng, X. Duan, and V. C. Leung, "Reliability-optimal cooperative communication and computing in connected vehicle systems," *IEEE Transactions on Mobile Computing*, vol. 19, no. 5, pp. 1216–1232, 2019.

[79] X. Hou, Z. Ren, J. Wang, W. Cheng, Y. Ren, K.-C. Chen, and H. Zhang, "Reliable computation offloading for edge-computing-enabled software-defined iov," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7097–7111, 2020.

[80] L. Li, X. Zhang, K. Liu, F. Jiang, and J. Peng, "An energy-aware task offloading mechanism in multiuser mobile-edge cloud computing," *Mobile Information Systems*, vol. 2018, pp. 3–15, 2018.

[81] O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 10, pp. 4738–4755, 2015.

[82] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[83] H. Trinh, D. Chemodanov, S. Yao, Q. Lei, B. Zhang, F. Gao, P. Calyam, and K. Palaniappan, "Energy-aware mobile edge computing for low-latency visual data processing," in *The 5th International Conference on Future Internet of Things and Cloud*, 2017.

[84] Y. Jang, J. Na, S. Jeong, and J. Kang, "Energy-efficient task offloading for vehicular edge computing: Joint optimization of offloading and bit allocation," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, IEEE, 2020, pp. 1–5.

[85] H. Viswanathan, E. K. Lee, I. Rodero, and D. Pompili, "Uncertainty-aware autonomic resource provisioning for mobile cloud computing," *IEEE Trans. on Parallel and Distributed Syst.*, vol. 26, no. 8, pp. 2363–2372, 2015.

[86] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[87] X. Fan, J. Cao, and H. Mao, "A survey of mobile cloud computing," *zTE Communications*, vol. 9, no. 1, pp. 4–8, 2011.

[88] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving portable computer battery power through remote process execution," *ACM SIGMOBILE Mobile Comp. and Comm. Rev.*, vol. 2, no. 1, pp. 19–26, 1998.

[89] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," in *Proc. 17th ACM Symp. on Operating Syst. Principles*, 1999, pp. 48–63.

[90] M. Patel, B Naughton, C Chan, N Sprecher, S Abeta, A Neal, *et al.*, "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[91] *Open edge computing*, `http://openedgecomputing.org`, (accessed October 5, 2021).

[92] *Openfog consortium*, `https://www.openfogconsortium.org`, (accessed October 5, 2021).

[93] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, *CRAWDAD dataset epfl/mobility (v. 2009-02-24)*, Downloaded from `https://crawdad.org/epfl/mobility/20090224`, Feb. 2009.

[94] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Comm. & Mobile Comp.*, vol. 2, no. 5, pp. 483–502, 2002.

[95] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Res. Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[96] (2017). GPS visualizer.

[97] D. S. Johnson, "A theoretican's guide to the experimental analysis of algorithms," *Data structures, near neighbor searches, and methodology: fifth ann sixth DIMACS implementation challenges*, vol. 59, pp. 215–250, 2002.

[98] (2009). IBM ILOG CPLEX V12.1 user's manual.

[99] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[100] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, and X. S. Shen, "Toward efficient content delivery for automated driving services: An edge computing solution," *IEEE Network*, vol. 32, no. 1, pp. 80–86, 2018.

[101] A. H. Sodhro, Z. Luo, A. K. Sangaiah, and S. W. Baik, "Mobile edge computing based qos optimization in medical healthcare applications," *Int. J. of Inf. Manag.*, vol. 45, pp. 308–318, 2019.

[102] H. Wang, J. Gong, Y. Zhuang, H. Shen, and J. Lach, "Healthedge: Task scheduling for edge computing with health emergency and human behavior consideration in smart homes," in *IEEE Int. Conf. on Big Data*, 2017, pp. 1213–1222.

[103] H. R. Varian, "Position auctions," *International Journal of Industrial Organization*, vol. 25, no. 6, pp. 1163 –1178, 2007.

[104] P. Cramton, Y. Shoham, and R. Steinberg, *Combinatorial Auctions*. MIT Press, 2006.

[105] T. Roughgarden, "Algorithmic game theory," *Communications of the ACM*, vol. 53, no. 7, pp. 78–86, 2010.

[106] M. R. Garey and D. S. Johnson, *Computers and intractability*. freeman San Francisco, 1979, vol. 174.

[107] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 713–728, 2005.

[108] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger, "Approximation algorithms for knapsack problems with cardinality constraints," *European Journal of Operational Research*, vol. 123, no. 2, pp. 333–345, 2000.

[109] L. Khachiyan, "A polynomial algorithm for linear programming," *Doklady Akademii Nauk SSSR*, vol. 224, no. 5, pp. 1093–1096, 1979.

[110] *DSRC Technology*, `https://www.auto-talks.com/technology/dsrc-technology`.

[111] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *ACM SIGPLAN Notices*, ACM, vol. 53, 2018, pp. 751–766.

[112] Tesla, *Autopilot*, `https://www.tesla.com/autopilot`, 2018.

[113] Geotab, *Gridlocked Cities - U.S. traffic congestion maps*, `https://www.geotab.com/gridlocked-cities/`.

[114] Y. Choi, S. Park, and H. Cha, "Graphics-aware power governing for mobile devices," in *Proc. 17th Annual Int. Conf. on Mobile Systems, Applications, and Services*, ser. MobiSys '19, ACM, 2019, pp. 469–481.

[115] S. Uppoor, O. Trullols-Cruces, M. Fiore, and J. M. Barcelo-Ordinas, "Generation and analysis of a large-scale urban vehicular mobility dataset," *IEEE Trans. Mobile Comp.*, vol. 13, no. 5, pp. 1061–1075, 2013.

[116] S. Wang, Z. Qian, J. Yuan, and I. You, "A dvfs based energy-efficient tasks scheduling in a data center," *IEEE Access*, vol. 5, pp. 13 090–13 102, 2017.

[117] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. on Comm.*, vol. 65, no. 8, pp. 3571–3584, 2017.

[118] E. Björnson and E. G. Larsson, "How energy-efficient can a wireless communication system become?" In *Asilomar Conference on Signals, Systems and Computers*, 2018.

[119] Z. Xu, X. Li, X. Zhao, M. H. Zhang, and Z. Wang, "Dsrc versus 4g-lte for connected vehicle applications: A study on field experiments of vehicular communication performance," *Journal of Advanced Transportation*, vol. 2017, 2017.

ABSTRACT

RESOURCE MANAGEMENT IN EDGE COMPUTING SYSTEMS

by

TAYEBEH BAHREINI

December 2021

Advisor: Dr. Daniel Grosu

Major: Computer Science

Degree: Doctor of Philosophy

Efficient utilization of computing resources has always been an important challenge for service providers, leading to significant efforts on developing solutions, either in the form of new technology or new ways to enhance the efficiency of existing technologies. Mobile Edge Computing (MEC) is the latest technology developed to improve the high latency in mobile cloud computing systems which stems from the long distance between cloud servers and the end user. MEC systems are expected to improve the Quality of Service (QoS) by bringing servers closer to the end user, but when it comes to the cost of services, these systems face important challenges. The operating cost of MEC systems is higher than that of the remote clouds, due to the small servers which are distributed across the network. On the other hand, compared to the cloud data centers, edge nodes have more restricted capacity. Another challenge in MEC systems is the mobility of users, that might make the current allocation of resources inefficient or even infeasible in few minutes. These issues become more challenging in the Vehicular Edge Computing (VEC) systems where each vehicle can be considered as an edge node.

In this dissertation, we address the mentioned challenges of resource allocation in MEC systems and VEC systems by designing efficient algorithms for resource management with the aim of improving the performance of these systems (i.e., energy consumption, operating cost, latency, and reliability).

We address the Multi-Component Application Placement Problem (MCAPP) in MEC systems. We formulate this problem as a Mixed Integer Non-Linear Program (MINLP) with the objective of minimizing the total cost of running the applications. In our formulation, we take into account two important and challenging characteristics of MEC systems, the mobility of users and the network capabilities. We analyze the complexity of MCAPP and prove that it is NP-hard, that is, finding the optimal solution in reasonable amount of time is infeasible. We design two algorithms, one based on matching and local search and one based on a greedy approach, and evaluate their performance by conducting an extensive experimental analysis driven by two types of user mobility models, real-life mobility traces and random-walk. The results show that the proposed algorithms obtain near-optimal solutions and require small execution times for reasonably large problem instances.

We also address the resource allocation and monetization challenges in MEC systems, where users have heterogeneous demands and compete for high quality services. We formulate the Edge Resource Allocation Problem (ERAP) as a Mixed-Integer Linear Program (MILP) and prove that ERAP is NP-hard. To solve the problem efficiently, we propose two resource allocation mechanisms. First, we develop an auction-based mechanism and prove that the proposed mechanism is *individually-rational* and produces *envy-free allocations*. We also propose an LP-based approximation mechanism that does not guarantee envy-freeness, but it provides solutions that are guaranteed to be within a given distance from the optimal solution. We evaluate the performance of the proposed mechanisms by conducting an extensive experimental analysis on ERAP instances of various sizes. We use the optimal solutions obtained by solving the MILP model using a commercial solver as benchmarks to evaluate the quality of solutions. Our analysis shows that the proposed mechanisms obtain near optimal solutions for fairly large size instances of the problem in a reasonable amount of time.

Another contribution is VECMAN, a framework for energy-aware resource management in VEC systems. The main motivation behind VECMAN is to is improve the energy

efficiency through sharing computing resources among connected EVs. However, the uncertainties in the future location of vehicles make it hard to decide which vehicles participate in resource sharing and how long they share their resources so that all participants benefit from resource sharing. VECMAN is composed of two algorithms: (i) a resource selector algorithm that determines the participating vehicles and the duration of resource sharing period; and (ii) an energy manager algorithm that manages computing resources of the participating vehicles with the aim of minimizing the computational energy consumption. We evaluate the proposed algorithms and show that they considerably reduce the vehicles' computational energy consumption compared to the state-of-the-art baselines.

AUTOBIOGRAPHICAL STATEMENT

Tayebeh Bahreini is currently a Ph.D. candidate in Computer Science at Wayne State University. She received her M.Sc. degree in Computer Engineering, from Shahed University, Iran in 2014, and her B.Sc. degree in Computer Science from University of Isfahan, Iran, in 2010. Her main research interests are edge and cloud computing, distributed systems, parallel computing, and combinatorial optimization. She is the recipient of the *2019 National Center for Women & Information Technology (NCWIT) Collegiate National Award*. She was selected as one of the *2019 Top Ten Women in Edge* and a *Finalist for the Edge Woman of the Year 2019 Award* by the Edge Computing World organization for her contribution to research in edge computing. She was selected to participate in the 8th Heidelberg Laureate Forum and the 2020 Rising Stars in Electrical Engineering and Computer Science Workshop at the University of California, Berkeley. She received, the 2020 Ralph H. Kummler Distinguished Achievement Award in Graduate Student Research from Wayne State University College of Engineering.