Thesis for the Degree of Doctor of Philosophy

Quality-of-Experience and Reputation Aware Incentive Mechanism for Workers in Mobile Device Cloud

Registration No: 67/2019 - 2020

Department of Computer Science and Engineering

University of Dhaka

Dhaka, Bangladesh

August, 2021

Quality-of-Experience and Reputation Aware Incentive Mechanism for Workers in Mobile Device Cloud

Sajeeb Saha

Department of Computer Science and Engineering

University of Dhaka

Dhaka, Bangladesh

August, 2021

As the candidate's supervisor, I have approved this dissertation for submission.

Name: Dr. Md. Abdur Razzaque

Signed: ______________________________

Name: Dr. Md. Mustafizur Rahman

Signed: ______________________________

Declaration of Authorship

We declare that this thesis titled "Quality-of-Experience and Reputation Aware Incentive Mechanism for Workers in Mobile Device Cloud" and the works presented in it are our own. We confirm that:

- The full part of the work is done during PhD research study in University of Dhaka, Bangladesh.
- Any part of this thesis has not previously been submitted for a degree or any other qualification in this University or any other institution.
- We have consulted the published works of others with appropriate references.
- This thesis work is done entirely by us and our contributions and enhancements from other works are clearly stated.

Signed:

Candidate

Countersigned:

Supervisor: Dr. Md. Abdur Razzaque

Co-supervisor: Dr. Md. Mustafizur Rahman

Abstract

Mobile Device Cloud (MDC) is a collaborative mobile cloud computing platform in which neighboring smart devices form an alliance of shared resources to mitigate resource-scarcity of an individual user device. It unfolds an improved computing opportunity for hand-held mobile devices to run compute-intensive applications like visual text translation, face recognition, augmented reality, and real-time health monitoring etc. exploiting code offloading mechanism. However, the sustainability of such a distributed platform depends on spontaneous participation of the involved mobile devices, i.e., resource-requester (buyer) and resource-provider (seller or worker). A fundamental challenge in such a resource-trading system is the selection of reliable worker mobile devices that enhances the computation quality of user applications. Moreover, participation of the worker mobile devices greatly depends on their compensations provided for the used resources. In this thesis, we focus on incentivizing mobile worker devices based on their task execution qualities to materialize a sustainable MDC system.

Selection of worker mobile devices for task offloading imposes great research challenges including computation quality and worker reliability. Unfortunately, these two performance parameters often oppose each other. In this thesis, we first develop an optimization framework that trades-off in between application execution speedup and reliability while maintaining device energy within a predefined range. We also design an algorithm for developing a dependency tree among the modules of a software application so as to allow higher number of parallel executions, wherever and whenever it is possible. The emulation results of the proposed algorithm outperform the relevant state-of-the-art works in terms of application completion time, communication latency and rescheduling overhead.

The second contribution of this thesis is to maximize user Quality-of-Experience (QoE) at minimum cost while providing attractive incentives to mobile worker devices. In literature works, mobile devices are assumed either to take part in execution voluntarily or aim to optimize one objective parameter (quality or cost) only. In this thesis, the aforementioned challenging problem is formulated as a multi-objective linear programming (MOLP) optimization function that exploits reverse-auction bidding policy. Practical application scenarios have been considered to trade-off between the cost and quality of execution. Due to NP-hardness of the MOLP, we offer two greedy worker selection algorithms for maximizing user QoE and minimizing execution cost. In both the algorithms, the amount of incentive awarded to a worker is determined following the QoE offered to a user. Theoretical proofs on holding desirable properties of the proposed incentive mechanisms have been presented. Simulation results depict effectiveness of our incentive algorithms compared to the state-of-the-art approaches.

Table of Contents

List of Figures

List of Tables

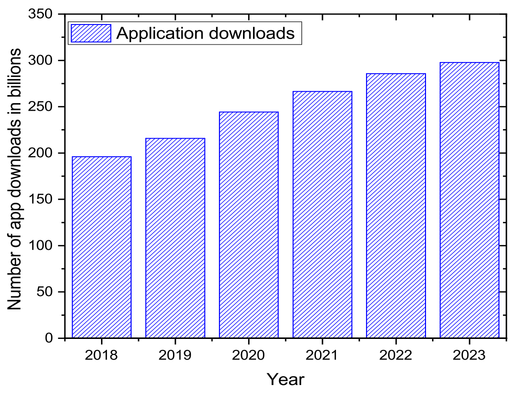
List of Algorithms

Chapter 1 Introduction

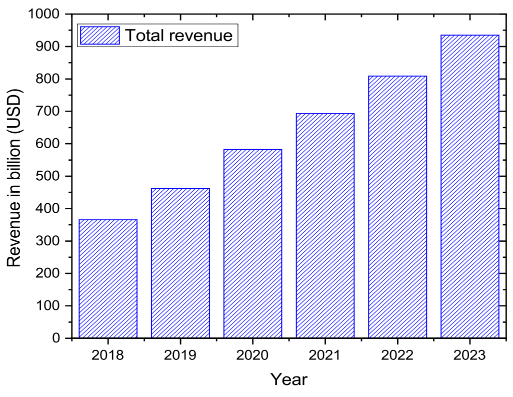
1.1 Introduction

The emergence of portable and wearable devices has increased the usage of mobile applications dramatically over the last few years. A recent study shows that around 195 billion applications were downloaded in 2018 which is projected to grow to 298 billion by 2023 giving a 52 % increase in just five years (Figure 1.1 (a) [1]). At the same time, the projected revenue from these applications will increase from 365 billion to 935 billion USD accelerating a 155 % increase as shown in Figure 1.1 (b) [1]. Now a days, the usage of mobile devices have gone beyond simple connectivity and the offered services require more complex processing. These include compute-intensive applications like speech recognition to aid in identifying snippets of audio or identifying objects from a set of images, reality augmentation to improve our daily living, collaborative sensing and monitoring to assist in distributed decision making and coordination, possibly in real-time [2, 3, 4, 5].

Over the last few years, there has been a significant advancement in the sophistication of mobile computing applications. Two key factors have been made this possible. Firstly, the processing and storage capability of portable mobile devices like smartphone, tablet, and wearable devices has been significantly improved in



(a) Application downloads



(b) Revenue

Figure 1.1: Mobile application downloads and revenue

every generation. Although energy constraint is still a burden for mobile devices, smart power management technologies have been introduced to enable longer-lived computation facilities in mobile devices. Secondly, the communication technology has been improved significantly offering diverse connectivity options with higher data rate for the mobile devices and enabled remote processing and execution of applications [6]. However, mobile devices still have stringent resource constraints like low CPU power, insufficient memory and storage size, limited bandwidth and limited battery power. As a result, execution of these applications on mobile devices, most of the time, is not feasible for various issues including delay, energy, reliability, etc [7, 8]. These issues have been addressed by many researchers with a general solution called Mobile Cloud Computing (MCC). The MCC utilizes code offloading mechanism to partition and offload computationally intensive and storage demanding tasks to a remote server with vast computational resources (i.e., cloud, cloudlet). Code offloading is proven to reduce the response time and energy consumption of mobile applications to overcome resource scarcities of mobile devices [9, 10, 11, 12, 13].

The Mobile Device Cloud (MDC) is a form of MCC where compute-intensive application tasks are offloaded to nearby mobile devices and these devices collectively act as a cloud server to execute the application tasks on behalf of the mobile device [6, 14, 15, 16, 17, 18]. The key philosophy of this MDC technology is to exploit the underutilized resources of nearby mobile devices. Since executors in MDC are located closer to the initiators, the energy consumption, communication latency or response delay have been reduced significantly compared to cloud based solutions.

To execute an application in an MDC system, a major challenge is to select reliable devices while minimizing execution time due to mobility and heterogeneous computation capacity of the mobile devices [16, 19]. Moreover, amount of energy available at a mobile device and its received signal strength also impact greatly on the reliable execution of a task [20, 21]. Considering these observations, this research aims to develop a system that optimally allocates offloaded codes to the members of MDC. The main target is to minimize execution time with the most reliable devices considering the device specific parameters like mobility, available energy, signal strength, computing capacity and application specific parameter like module dependency.

Voluntary task execution might impose a major challenge on the sustainability of an MDC system due to lack of participation of worker mobile devices. Moreover, bandwidth charges and limited resources further demotivate the workers in participating task execution in MDC [15, 22]. For this reason, to design a sustainable MDC technology, incentivizing the workers is essential in addition to the compensation of the used resources. Considering these observations, we extend our research to develop a Quality-of-Experience (QoE) aware incentive mechanism for workers in an MDC system to maximize the user QoE and minimize the execution cost.

The rest of the chapter is organized as follows. An introduction to the state-of-the-art code offloading technologies are presented in section 1.2. An overview of Mobile Device Cloud including its applications and challenges are described in Section 1.3. The problem statement and the solution methodology have been discussed in Section 1.4. Section 1.5 reflects the contributions of our research. Finally, the thesis organization is presented in Section 1.6.

1.2 Code Offloading

A mobile device often meets a lot of entities capable of sharing its resources that creates an opportunity for remote computation in a mobile environment. Offloading to these nearby devices, a mobile device can provide better execution performance in addition to prolonging the battery lifetime. However, in addition to performance gain and energy savings, the key questions for offloading decision are: where to offload, which to offload and whom to offload? Addressing these questions to offload application tasks in a remote server, researchers have exploited three different mechanisms as illustrated in Fig. 1.2.

At the early stage of introducing concept of code offloading, the distant enterprise cloud is used for offloaded execution that are explored in [12, 13, 23, 24, 25, 26]. In this method, the mobile device or the remote cloud determines which portion of the code is to be offloaded for execution in the cloud. An enterprise cloud or cloud server contains a shared pool of configurable computing resources which are provisioned dynamically to provide a ubiquitous, convenient, on-demand service to

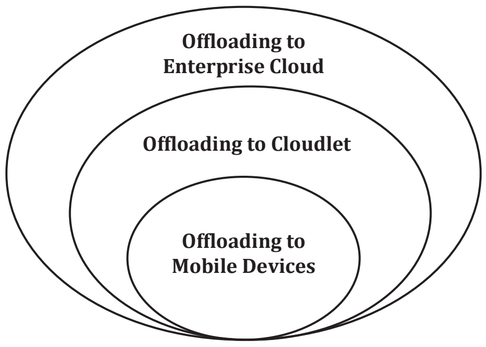


Figure 1.2: Mobile application offloading technologies

a user. These cloud servers are owned and maintained by third party providers and Internet access is a prerequisite to get services from these servers. Though the cloud offloading technique significantly reduces the execution time, some real-time and latency sensitive applications cannot tolerate it due to excessive communication delay between cloud provider and mobile devices [16, 27].

The second category of code offloading strategies introduced cloudlets which are smaller version of the enterprise cloud that contain limited amount of resource-rich computation entities placed at different locations for executing offloaded program-segments [28, 29, 30, 31, 32]. These cloudlet servers are located in public or commercial places like coffee shops, bus terminals, markets, or airport lounge where people gather together and spend some time. Since the cloudlets are in the vicinity of mobile devices, the data transmission latency gets reduced significantly compared to distant enterprise cloud environment. However, the performance of offloading to cloudlet degrades significantly due to the increasing penetration of mobile devices and contention between mobile devices under a single cloudlet, especially when with limited resources [33].

To enhance the computation performance further, researchers introduced Mo-

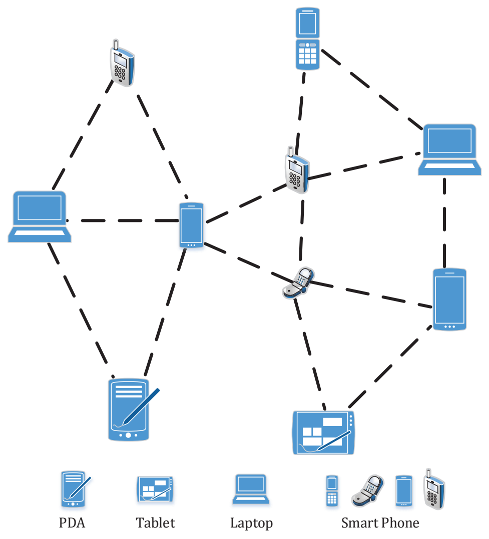


Figure 1.3: A mobile device cloud environment

bile Device Cloud (MDC) where compute-intensive application tasks are offloaded to nearby mobile devices and these devices collectively act as a cloud server to execute the application tasks on behalf of the mobile device (as shown in Fig. 1.3) [16, 27, 34, 35, 36]. In the literature, this opportunistic computation of application codes on nearby mobile devices is also known as Ad-hoc Mobile Cloud [37, 38, 39] Dynamic Mobile Cloud [14, 40, 41] or Virtual Mobile Cloud [13, 42]. The key philosophy of this MDC technology is to exploit the underutilized resources of nearby mobile devices. Since executors in MDC are located closer to the initiators, the energy consumption, communication latency or response delay have been reduced significantly compared to cloud based solutions.

1.3 An Overview of Mobile Device Cloud

Almost at everywhere, we are surrounded with a plethora of mobile devices either in large scale stationary place (i.e. stadium, shopping center, restaurants or Movie Theater) or during travelling by bus, launch, train and on air (Fig. 1.4). Though the devices are mobile, collectively they become stationary in the context of onboard vehicle or at the situated place. In such cases, collaboration among these stationary mobile devices may unfold improve computing opportunities for running resource hungry applications.

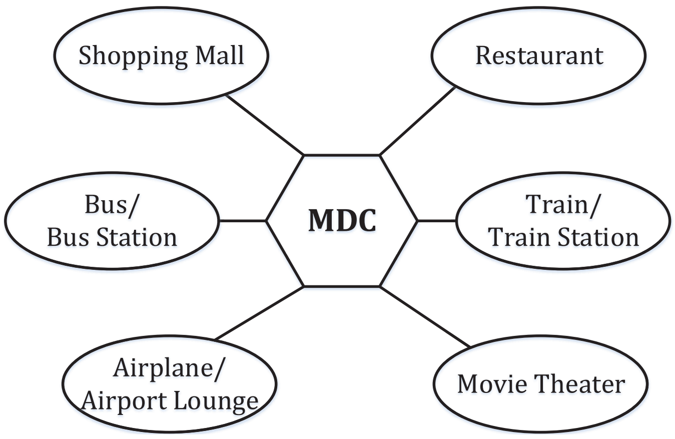


Figure 1.4: Opportunistic environment for mobile device cloud

The main essence of MDC technology is to exploit the unutilized resources of nearby hand-held smart devices to fulfill the demand of a resource-constraint device [43, 44, 45]. The inclusion of diverse smart applications demanding high computation with low latency, the mandate of such collaborative computation technologies is increasing day by day [16, 21]. Some of the key benefits of MDC technology are highlighted as follows:

- Increased battery lifetime: Available energy of a device is one of the key concerns while executing an application in a mobile device. Due to mobility, such kind of battery powered device always seeks for offloading facility with minimum energy consumption. Distant cloud based offloading mechanism consumes a significant amount of energy power for the communication. Due to proximity of the mobile devices, MDC offloading significantly reduces the communication latency resulting a lower energy consumption compared to other offloading strategies.

- Enhanced execution quality: In context of device resource capacity, a mobile device tries to find an offloading solution for two reasons: 1) if the device doesn't contain the necessary resource to execute the application or 2) the device wants a higher execution quality compared to its local execution. In both cases, MDC system might be capable of executing the application with desired quality, resulting in a better quality-of-experience (QoE) for the user device.

- Increased resource utilization: In MDC, the unused resources of mobile devices are shared with resource-constrained devices to execute their applications. Such kind of resource sharing facilitates efficient utilization of idle resources while enabling execution of applications beyond the resource capability of a mobile device.

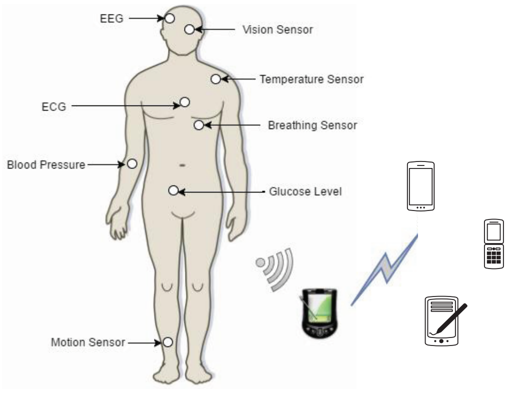
1.3.1 Applications of MDC

The development of MDC was motivated from parallel and distributed computation of applications in wireless Adhoc networks. The emergence of MDC has expedited the growth of a broad variety of distributed applications on a large scale in recent

(a) Real-time text translation

(b) Natural language processing

(c) Emergency disaster response



(d) Real-time healthcare data processing

Figure 1.5: Applications of mobile device cloud

years. The following areas are major categories of MDC applications.

- *Real-time text translation:* Real-time text translation is a typical area amongst the many possible applications of MDC system. For example, consider a tourist traveling in a foreign country by train as shown in Figure 1.5 (a). To spend his leisure period he starts to watch a movie. Unfortunately, the subtitle of movie is not native to the tourist. Though his movie player has a text translation software to translate the text in his native language, he becomes unable to run it with his available battery power and computing resources. To get rid of this problem, he initiates an application that creates an MDC environment where the subtitle text is transferred to the nearby devices and each device translates part of the text [41].

- *Natural language processing:* Recent research has increasingly focused on natural language processing applications that has made the communication much easier compared to any other times. Though these applications require a large amount of computation resources, real-time processing of such applications can be supported by the shareable resources of an MDC system. Consider a student attending a conference where speakers from different countries have come to deliver talks on their research results. Unfortunately, the speakers are delivering speeches in their native languages which the student cannot understand. Though the student has a speech recognition and synthesis application in his cellphone, he cannot use it due to lack of computational resource and/or battery power. To solve this problem, he may initiate an MDC system and use the computational resources of other attendees in the seminar room as shown in Figure 1.5 (b) [46].

- *Emergency disaster responses:* Search and rescue of the inhabitants after a natural disaster like earthquake, flood, tsunami, and landslide become almost impossible due to lack of communication infrastructure. In such emergency situations, MDC system can be deployed to coordinate the search and rescue of different relief groups. For example, consider a search and rescue operation where several people are missing after a natural disaster as shown in Figure 1.5 (c). Relatives have gathered in the rescue center with photographs of the missing persons. The matching of such huge amount of images cannot be done in a local device as the device is not equipped with enough resource to process such tasks. On the other hand, due to lack of infrastructure the images cannot be transferred to a centralized server. To solve this problem, the relief workers may create an MDC infrastructure using their mobile devices and collaboratively process individual images to identify a person [3, 47, 48].

- *Real-time healthcare data processing:* The health-care facilities provided to the citizens act as one of the key metrics to measure the prosperity of a nation. The health-care applications are always seeking for more powerful and efficient mechanisms to deliver on-time support to the emergency patients with the best care and pleasant services. The MDC system can be utilized to collect healthcare data of patients and diagnose in real-time to monitor and track their activities which can significantly reduce the casualty rate of emergency patients (Figure 1.5 (d)) [49, 50].
- *Reality augmentation:* With the help of computer vision and object recognition, smartphone applications can now imitate natural environments or situations which is known as augmented reality. Execution of such computation intensive applications can also take support from the nearby devices to process these virtual objects whenever necessary in the form of an MDC environment [51, 52].
- *Tactical networks:* Tactical networks mainly depend on collaborative processing of battlefield data where infrastructure based services cannot be installed. The MDC system can be a solution to provide such a collaborative environment to facilitate execution of Military applications like identification of objects, monitoring and sensing of targets, etc [53].

1.3.2 Challenges of Code Offloading in MDC

In a study, the authors show that, most of the time mobile devices stay in idle state whereas other devices can't afford to execute large applications by their limited processing powers [11]. It becomes impossible due to the limited computation capacity of mobile devices to execute real-time multimedia applications. To do so, we can aggregate the idle computation power of the nearby mobile devices under an MDC to execute a compute intensive application. However, the MDC itself issues several challenges including determination of portion of code to be offloaded and where to offload, maintenance of connectivity among mobile devices, assurance of device availability in terms of minimum battery energy, etc. This challenge is extended further if the application expects execution with reliable devices having heterogeneous computation capacity to minimize execution time. Moreover, the amount of energy available at user device and its signal strength greatly impact on the reliable execution of a task. The following subsections explain the key challenges of code offloading in MDC environment.

1.3.2.1 Resource heterogeneity

Offloading an application primarily depends on the shareable amount of resources availability at nearby mobile devices. These resources include mostly CPU for processing and RAM for temporary storage of application codes. Now a days, mobile devices are equipped with rich computational resources, though these resources greatly vary from one device to another. For this reason, selection of an appropriate mobile device among these heterogeneous computation entities to execute application tasks imposes a major challenge in making offloading decisions [54].

1.3.2.2 Task interdependency

In code offloading, an application is apportioned into a set of modules which are known as tasks. The key philosophy of this apportion is to enable parallel execution to minimize the execution time of an application [55]. Though the tasks are independently executable, their execution may be dependent on producing the execution result of other tasks. In such cases, scheduling of a child task before the parent task may adversely affect the overall execution result of an application. For this reason, estimation of interdependency and maintaining the dependencies among the tasks imposes another challenge in scheduling of application tasks.

1.3.2.3 Application deadline

Deadline defines the maximum allowable time to execute an application including the communication latency. While offloading an application to nearby devices, a primary objective is to execute the application with a lower execution time compared to the local device. Moreover, time-sensitive applications expect the execution result within a maximum tolerable delay. In such situations, application deadline plays a key role in selection of the mobile devices so that the execution results can be provided to the user device within the allowable deadline [41].

1.3.2.4 Energy-critical user device

One of the key reasons of code offloading is the lack of having sufficient energy in a mobile device. The key philosophy of MDC is to exploit idle resource of nearby devices without exhausting their battery powers. If a device drains out of its battery power for executing offloaded codes then the mobile device will not be able to provide basic services to its user. For this reason, available energy of a mobile device put restriction on the selection of a mobile device to execute an offloaded task.

1.3.2.5 Lack of compensation for mobile users

In the emerging industry 4.0, MDC is becoming one of the key contributors that will be used for the collection and processing of data from different disruptive technologies like IoT, AI, Robotics, Cloud Computing, and Virtual reality applications [56, 57]. Execution performance of an offloaded application greatly varies with the number of participated devices as well as their resource capacity. Participation of more number of devices increases the chance of having execution with higher quality. Lack of a compensation of the used resources (Bandwidth charges, CPU, RAM) might demotivate the owner of mobile devices to perform task execution in MDC [15, 58]. However, to make such a collaborative technology sustainable, the collaborative participation of nearby mobile worker devices is a prerequisite.

1.4 Problem Description and Solution Methods

In this section, we briefly explain the problem addressed in this dissertation and key philosophy of solution methodologies.

1.4.1 Dissertation Problem

To execute an application in MDC system, the most fundamental challenge is to select suitable worker devices. Finding a set of qualified and reliable workers is the prerequisite to execute an offloaded application and thus to meet the user requirements. Large applications are partitioned into a set of tasks to make them executable on MDC. A task is executed either on MDC or local device to minimize energy consumption, response time, data transmission time exploiting idle computing resources. In the literature, code offloading mechanism can be classified in two dimensions based on the payment of the worker devices:

- *Voluntary task execution in MDC:* In voluntary task execution approach, the key philosophy is to execute the tasks of an application with the voluntary participation of nearby mobile devices. In this approach, the main focus is to minimize the application execution time through parallel execution of individual tasks [16, 21, 27, 34, 41]. This strategy neither focuses on the reliability of worker devices nor provides any compensation to them for the used resources.

- *Paid task execution in MDC:* In this approach, a user offloads an application to nearby worker devices with a fixed budget. The worker devices contends for their interested tasks with an approximate cost of their shared resources [22, 58, 59, 60]. In this approach, the key philosophy is to minimize the task execution time while making the minimum expenditure.

1.4.1.1 Scope of the work

In MDC system, worker device selection plays a pivotal role in the successful execution of an application. Nearby devices with high computing resources may often not provide services with good reliability, and vice-versa. As MDC is an emerging technology, numerous researchers are working on this fundamental challenge of task allocation on worker devices to minimize execution time. However, a number of related key factors including selection of offloadable tasks, dependency among tasks were not considered in scheduling tasks [16, 27, 34]. Similarly, only a few of the worker related parameters were considered including associativity time, reputation, available energy, and resource capacity while assigning tasks to worker devices [16, 21, 41]. However, for successful execution of an application with better performance all the above mentioned parameters need to be addressed jointly.

In addition to this, such a voluntary-based task execution faces sustainability problem due to lack of participation of reliable workers with rich computation capability. Bandwidth charges and limited resources also demotivate the workers further in participating task execution in MDC. To execute an application with guaranteed service, a payment should be provided based on the execution service [22, 58, 59, 60].

In this thesis, we aim to develop an MDC framework to facilitate collaborative computation of an application utilizing the idle resources of nearby mobile devices [61, 62]. Considering the sustainability of the technology, we focus on designing a payment system based on the execution result of the tasks to compensate the used resources and increasing the participation. To promote high quality execution from workers, we would like to introduce incentive as a reward in addition to their regular payment [63]. The detail discussion on the development of code offloading framework for task assignment and incentive algorithms for worker payment are given in Chapter 3 and Chapter 4.

1.4.1.2 Research objectives and motivation

The research objectives and motivation towards the proposed MDC system are described in the following sections.

- Enhanced user-quality-of-experience (QoE): The expectation toward offloading an application is to enhance the user-QoE with minimum cost. The user-QoE is defined as the improvement of execution performance observed by a user for running an application in MDC. However, it is observed that the quality and the cost hold an inverse relationship, i.e., increasing the execution quality demands selection of workers with rich computation resources; on the other hand, the selection of high computation resources demands higher payoffs, resulting in higher execution cost. Therefore, an efficient tradeoff is needed to address both the parameters so that we can optimize the cost while preserving the required quality or vice-versa.

- Reliable task execution: To execute the tasks of an offloaded application, the selection of worker devices considers resource capacity and the reliability of a worker device. The trustworthiness of a worker device plays an important role in the successful execution of an offloaded application. The selection of a felonious worker may result in a failed execution of a task due to falsified advertisement of resources that induces resubmission of the task. Such kind of resubmission decreases the quality of an execution that inherently hampers the satisfaction of a user. Our proposed MDC system takes account of such felonious activities during task allocation to reduce the amount of resubmission resulting in better user satisfaction.

- Utilizing idle resources: MDC based offloading facilitates better usage of idle resources of the nearby worker devices. However, a worker device may be interested to share a partial amount of resources for offloading service. Moreover, a worker device could be involved in executing tasks of multiple applications from different users. For this reason, a task can be offloaded to a worker device based on its available resource capacity. To address the aforementioned specification, in our proposed system, we have taken account of the maximum resource capacity of a worker device in assigning a task.

- Cost minimization: A primary concern of the MDC system is to motivate mobile devices to participate as workers through resource sharing. To increase participation, such a method of motivation is to implement a payoff for the shared resources. Conventionally, a user always attempts to execute an application with possible minimum cost while a worker will try to maximize its profit. Hence, the system should be able to produce a competitive environment to minimize the overall cost while providing attractive incentives in executing an application.
- System sustainability: Sustainability is defined as the constant existence of an application or a system despite all the adverse environment. The MDC technology produces an environment that enables mobile devices to share their idle resources in executing application tasks of other users. As the technology is fully dependent on the availability of shared resources offered by the nearby mobile devices, a prime concern is the sustainability of the technology. Increasing participation of adequate number of devices can highly improve the system performance. For this reason, the development of a mechanism is essential to invite more workers in the system facilitating better utilization of resources that would inherently increase the sustainability of the technology.

1.4.1.3 Coordination in offloading

In MDC, an offloaded execution consists of a series of activities including apportion of tasks, selection of worker mobile devices, scheduling of tasks according to dependency, accumulation and sending the result back to the user device, etc. Such kind of activities contain complex processing and require both high computation and energy. Running this decision making algorithm on the user device might avert the gain of the overall process. For this reason, a resource-rich computation entity is introduced to facilitate smooth trading of the resources. To serve the purpose, in this work, we assume a cloudlet, which is a resource-rich computation unit equipped with line power supply and stays close to users, making it suitable to execute the decision algorithm within a short period.

1.4.2 Solution Methodology

In this section, we describe our proposed architecture and computational framework to execute an application in MDC environment.

1.4.2.1 Code offloading architecture in MDC

Usually, computation-intensive applications are hard to execute on mobile devices and if executed on the mobile devices the batteries drain out quickly as it requires a large number of CPU cycles. If the application is apportioned into different tasks and offloaded to nearby cloudlets or other mobile devices, then the whole application can be executed in much less time while saving a considerable amount of energy. To facilitate the collaborative execution of different tasks we provide an architecture which is shown in Figure 1.6, where several mobile devices are connected through a wireless interface (Wi-Fi). It is a two-tier architecture with user and worker mobile devices at tier one and cloudlet at tier two. The user device has a compute-intensive application to execute but is unable to execute it due to constrained resources. So, it communicates with the cloudlet at tier two to make an offloaded execution of the application. The cloudlet then communicates with the connected worker devices and selects the appropriate devices to distribute the apportioned tasks of the application for execution. The cloudlet coordinates the distribution of tasks to worker devices, accumulation and sending the result back to the user device.

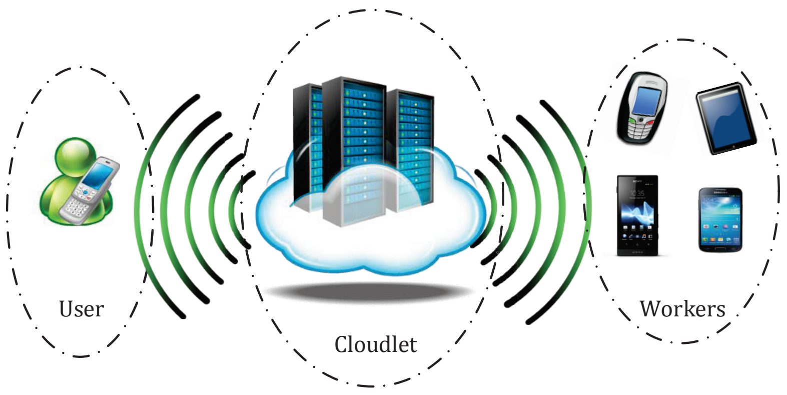


Figure 1.6: Code offloading architecture for MDC

1.4.2.2 Framework of proposed MDC system

In this dissertation, we develop a computation framework for the smooth operation of different entities to execute an offloaded application. The developed framework employs three major components to collect and distribute apportioned tasks to individual workers. Figure 1.7 shows the basic framework and the interaction among different entities of our proposed system. At first, we develop a voluntary execution service where the tasks are executed without any payment or benefit. The proposed task-worker allocation algorithm optimally assigns tasks that speed up the computation through reliable worker devices. Later, a payment mechanism has been devised that motivates high-quality workers to participate in sharing resources for executing offloaded tasks. To increase the task execution quality an incentive mechanism has been adopted that provides additional payment as incentives to qualified execution. Such kind of incentive mechanism enhances the user quality-of-experience and creates a win-win situation both for users and workers which is essential for the sustainability of the technology.

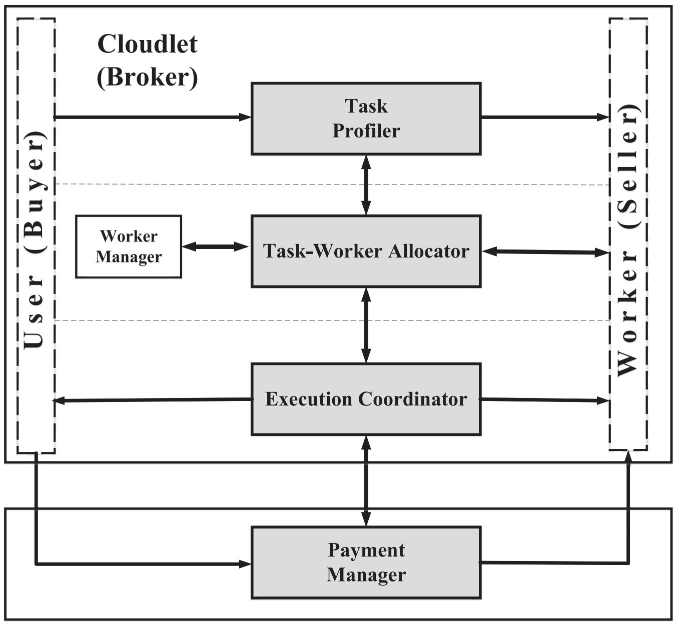


Figure 1.7: Framework of the proposed mechanism

1.4.2.3 Optimization of the Framework

Code Offloading in MDC poses challenges that address novel modeling and optimization concepts in the selection of worker devices (i) to enhance the task execution quality, (ii) to increase the reliability in task execution, (iii) to minimize the execution cost, and (iv) to incentivize the worker devices for high-quality execution. The key philosophy of this work is to enhance the task execution quality with reliable worker devices while minimizing the task execution cost. In this thesis, the problem has been addressed as a multi-objective optimization since the goal of the problem is to find an optimal solution from a set of conflicting objectives and hence we need a tradeoff between them.

1.5 Contributions of This Thesis

The MDC is a virtual cloud technology that is created opportunistically with the help of stationary mobile devices in a particular area. The sole purpose of this technology is to utilize the idle resources in a collaborative execution environment offered by resource-constrained devices. The system contains no fixed devices except the cloudlet which monitors and administrates the activities of these devices. Hence, in addition to computation resources, reliability, association period of these devices plays an important role in disseminating application tasks to these devices. In this thesis, we address quality and reliability aware worker selection problem to execute a compute-intensive application in MDC. We explore the MDC platform to design and develop a collaborative and distributed computing architecture to execute a compute-intensive application with required quality and reliability. The main philosophy of the work is to create a virtual cloud platform with the idle computing resources of the nearby worker devices where the application tasks can be scheduled in a non-overlapping manner to attain the desired quality of execution. This thesis also answers how can the worker devices be benefited for sharing their resources in application execution. To do so, we aim to incentivize the worker devices following their execution qualities in addition to their regular payment for successful execution of application tasks.

The first contribution of this dissertation is to develop a code offloading framework named TESAR for the MDC system where worker devices participate voluntarily in executing application tasks of a user [62]. We formulate an optimal function that assigns each task to a worker device intending to speed up the computation and maximizing the reliability. The goal of the formulated mechanism is to make a tradeoff between execution speedup and reliable execution of tasks based on the application requirement. The objective function has been designed in such a way that it can be used to maximize the reliability or computation speedup or to make a tradeoff between these two. To conduct the experimentation, we implement an emulation testbed using android platform that demonstrates that our proposed TESAR system can significantly improve the application execution performance compared to other state-of-the-art-works in terms of execution time, communication latency, and rescheduling overhead.

To motivate the worker devices in sharing resources, in our second work, we emphasize to extend our proposed system to incorporate a payment mechanism for the used resources of the workers. Focusing on this, we develop a QoE-aware incentive mechanism for workers in an MDC system to maximize the user QoE and minimize the execution cost. To facilitate resource trading, we employed a reverse auction mechanism that allocates tasks of an application among the workers in the system. Based on the bids of the workers, we formulate an optimization function with an objective to minimize user cost and maximize the quality of the executed task. As this formulation turns to be an NP-hard problem with the growing number of tasks and worker devices, we develop two alternative greedy algorithms for maximizing the user QoE and minimizing the execution cost respectively. In contrast to regular payment mechanisms, we incentivize the worker devices based on their execution quality in addition to their bid amount. The desirable properties of our proposed incentive mechanisms have been proven theoretically. For numerical analysis, we conducted simulation in MATLAB that depicts the effectiveness of our proposed algorithms with state-of-the-art-works in terms of user QoE, cost and satisfaction.

1.6 Organization of This Thesis

The outline of the thesis is as follows. In Chapter 2, we overview state-of-the-art task execution mechanisms in MDC and discuss the motivation of this work. In Chapter 3, we develop a voluntary task execution mechanism to speed up the computation in MDC. Further improvement on user QoE is explored through exploiting incentive mechanisms for offering qualified execution in Chapter 4. Finally, we conclude the thesis in Chapter 5 by summarizing the findings in the thesis and describing avenues of possible extensions to this work.

Chapter 2
Literature Review

In this chapter, we overview necessary background studies for task execution in MDC system. We also discuss the state-of-the-art incentive mechanisms available for worker devices in MDC system.

2.1 Introduction

A Mobile Device Cloud (MDC) comprises of a group of mobile devices that collectively act as a cloud computing service provider. The computing resources of these mobile devices are exploited to execute offloaded tasks of an application. The shared idle resources create an opportunistic computing community that allows collaborative execution of compute-intensive application tasks [14, 16, 21]. Such type of virtual cloud computing infrastructure plays a crucial role in situations with no or weak connections to the Internet.

In recent years, MDC has become a major research area in mobile cloud computing and has received immense interests from the research communities. The appropriate selection of worker devices is the heart and foundation for the execution of any offloaded task. Reliable execution of a task also closely impacts on the selection of worker devices. Besides, user QoE highly depends on the execution quality of a task. Moreover, offloading to MDC can provide monetary savings by avoiding data charging costs while offering enhanced performance with reduced communication latency. Execution of tasks in MDC can be categorized into two strategies: 1) voluntary task execution and 2) paid task execution. In a voluntary task execution approach, resources of nearby mobile devices are shared free of cost for the collaborative execution of a compute-intensive application [16, 21, 27, 34, 41, 64, 65, 66, 67]. In return for the service, no benefit is offered to the worker mobile devices. On the other hand, in the paid task execution approach, worker mobile devices are offered with a handsome amount as compensation of the shared resources used for executing application tasks of a user [15, 22, 58, 59, 60, 68, 69, 70, 71]. The main motivation towards offering this payment is to increase participation of worker devices for sharing idle resources.

Since the inception of the MDC mechanism, a good number of researches has been performed to find appropriate worker devices for the execution of offloaded tasks. In the literature, the primary focus of these approaches concentrated on minimizing the execution time of application tasks to speed up computation performance. To do so, user applications are apportioned into tasks and distributed into different worker devices. Worker mobility, signal strength, and resource capacity act as the key parameters in the selection of worker devices. The outcome of these researches depicts that offloading tasks to nearby mobile devices can significantly improve the application execution performance while prolonging the battery lifetime of a user mobile device. However, the successful execution of tasks mostly relies on the selection of reliable workers because a poor worker may significantly hamper offloading performance. The selection of reliable workers for maximizing task execution quality is yet to be well explored in the literature.

Next, considering bandwidth charges and resource usage researchers emphasized designing a payment system as compensation of used resources and to motivate workers in sharing resources. In state-of-the-art-works, the key philosophy of such a payment mechanism concentrated on either minimizing task execution cost [22, 59] or maximizing profit of the worker devices [58, 60]. The selection of the worker devices is governed by an auction mechanism where worker payment is determined by the amount of resources used in the execution of tasks. However, these approaches still lack in identifying reliable workers to increase user QoE for executing offloaded tasks. Moreover, in addition to a regular payment, worker devices can be incentivized according to the quality of the executed task.

In this chapter, state-of-the-art works on the selection of worker devices in MDC have been categorized based on different types of task execution strategies such as voluntary or payment-based and are listed in Table 2.1 and Table 2.2, respectively.

The rest of the chapter is organized as follows. In section 2.2, we focus on state-of-the-art-works that provide a solution for the execution of tasks with the voluntary participation of worker devices. Section 2.3 studies state-of-the-art-works on payment of the worker devices for the used resources of worker mobile devices to maximize the user QoE. Finally, we summarize this chapter in Section 2.4.

2.2 Code Offloading in Cloud Computing

The advancement of computing, storage, and network technologies has enabled the execution of diverse resource-hungry applications in a mobile device. Though computational resources and application demand are thriving at a rapid pace, still there exists a consistent resource constraint that inhibits to achieve a tangible performance from a mobile device. Researchers exploited code offloading mechanism in which a resource-hungry application or a portion of that is migrated to a remote server, having rich computation resources, to alleviate the resource constraints of a mobile device. The key philosophy of code offloading technique is to speed up computation and to reduce energy consumption. The following sections discretely describes the state-of-the-art cloud-based code offloading strategies.

2.2.1 Code Offloading to Remote Cloud

In the preliminary state-of-the-art works [12, 13, 23, 25, 26, 72, 73, 74], researchers used the remote cloud for offloading the computation. In [72], Chun et al. proposed a flexible task partitioning system to migrate and execute part of a task in the remote server. It used static analysis and dynamic profiling for partitioning tasks to reduce execution time and energy usage.

To extend the battery lifetime of a mobile device, Huang et al. formulated a Lyapunov optimization based dynamic offloading algorithm [12]. The algorithm determines offloadable tasks of an application according to change in the wireless environment while satisfying application delay deadline constraint.

Ra et al. proposed a runtime system named Odessa that supports mobile applications offloading tasks to the cloud [73]. They implemented a greedy offloading algorithm based on task execution time which incrementally makes a decision to offload each task in stages. Though Odessa is fast but it is far from optimal and performs poorly when the network bandwidth is limited. Moreover, the offloading decision for each task is made without the global view of the other tasks involved in the application.

In [13], Yang et al. designed a framework to provide runtime support for the dynamic computation partitioning and distributed execution of compute-intensive application tasks in remote cloud. A genetic algorithm was formulated to apportion the application into atomic tasks. Later, Yang et al. addressed the partitioning of application tasks between the cloud and mobile device and scheduling offloadable tasks on a constrained amount of cloud resources to minimize the average execution delay of latency sensitive applications [74]. The system also modeled a performance resource-load (PRL) which is used to make an optimal tradeoff between the application performance and cost of the cloud resources. Though their proposed mechanism significantly improved the throughput of task execution, their implementation ignored the cloud service time which is not practical.

Considering the cloud service time, Jia et al. presented an online task offloading algorithm to minimize completion time of tasks in a mobile application [23]. For sequential tasks, an optimal offloading algorithm was proposed whereas a load-balancing heuristic was introduced for concurrent tasks that maximizes the parallelism between the mobile and the cloud. However, their work failed to address the dependency between the tasks in an application.

To execute compute-intensive applications in a wearable device, Cheng et al. proposed a three-layer code offloading architecture consisting of wearable devices, mobile devices, and a remote cloud where application tasks are transferred to mobile device or remote cloud for offloaded execution [25]. They proposed a genetic algorithm-based fast heuristic method to determine the maximum number of tasks that can be executed on a wearable device within the tolerable delay deadline.

To offload tasks from a mobile device to a cloud server using 5G network, an intelligent computation offloading system was proposed by Khoda et al. in [26] that carried out tradeoff between energy savings and latency requirement of an application. They used a Lagrange Multiplier to regulate the code offloading decision and a statistical regression based model to estimate execution time of a task more accurately.

In the above offloading mechanisms, the main objective is to offload an application or application tasks to a remote cloud that minimizes task execution time and prolongs the battery lifetime of user mobile devices. However, such kind of offloading suffers a large communication latency due to cellular connectivity between the mobile device and the remote cloud that averts the remote cloud execution performance.

2.2.2 Code Offloading to Cloudlet

To minimize communication latency of remote cloud, Satyanarayanan et al. envisioned cloudlet, a trusted, resource-rich computation unit to execute offloaded application tasks of mobile users [28]. The concept and ideas stated that cloudlets can be exploited as an alternative to the remote cloud for serving tasks of multiple applications from different users using the virtual machine (VM) technology. The research also addressed promising prospects and key challenges towards the deployment of this technology.

In [29], Jararweh et al. introduced a cloudlet-based code offloading mechanism to reduce energy consumption and communication latency in mobile cloud computing that takes into account the mobility and movement nature of a mobile device. The experimental analysis showed that their proposed approach can significantly reduce energy consumption and communication latency while maintaining the desired QoS.

Considering user mobility, real-time network performance and server loads, Li et al. proposed a three-tier architecture including smartphones, cloudlets, and clouds to optimize the offloading decision [30]. They proposed a greedy search algorithm to predict the mobility of users and an efficient Wi-Fi access point (AP) selection method to determine appropriate AP for offloading application tasks.

Mahmud et al. developed a multi-objective nonlinear programming solution to execute context-aware application software in a local cloudlet [32]. The main objective of the proposed QoE and context-aware scheduling method was to maximize user QoE and application execution success rate.

In [75, 76], Jin et al. studied resource sharing for cloudlet in mobile cloud computing, and designed efficient auction mechanisms to guarantee the truthfulness of the bidders. The proposed schemes conduct well for homogeneous systems, where the amount of resources required by each buyer is the same as the amount of resources available at each seller. Moreover, their works restrict the one-to-one resource trading mode, which omits the fact that the resource-rich seller (i.e., mobile device) can support multiple buyers of resources in a practical mobile cloud system.

From the above discussion, it is evident that cloudlet execution can significantly reduce the communication latency for being in close proximity to the user device. Though cloudlets can notably reduce communication latency, their performance becomes unreliable with the increasing number of offloading requests. This is due to the fact that the amount of resources in a cloudlet is limited in comparison to the master cloud and hence it cannot satisfy the requests of user applications at pick hours.

2.3 MDC as a Code Offloading Framework

The immense increase of number of mobile devices and applications running on those, cloudlet based solutions can no longer satisfy the resource requirement let alone the remote master cloud based solutions. Offloading modules of an application to nearby mobile devices with idle computing resources is considered to be the future of mobile computing [42, 77, 78]. Such a mechanism was first introduced by Canepa et al. in [42] that provided a guideline to create a virtual cloud computing framework using the resources of mobile devices in the vicinity of a user. The authors implemented a prototype application that showed the effectiveness of the proposed framework in terms of execution time.

Fernando et al. envisioned a dynamic mobile cloud computing framework to opportunistically exploit the runtime resources of the mobile devices for executing offloaded tasks cost-effectively considering mobility and cost [14]. They also discussed the possible challenges in such an opportunistic network like mobility, job partitioning, job distribution, cost estimation, connectivity options, and fault tolerance, etc.

To exploit the idle resources of nearby mobile devices, Shi et al. also proposed a code offloading framework named Serendipity in [6]. The paper proposes a powerful job structure that enables a mobile computation initiator to use remote computational resources available in other mobile systems to speedup computing and conserve energy. The paper also provides an algorithm to disseminate and monitor the tasks among mobile devices. Based on the Serendipity architecture, Shi et al. developed an application to use the computational resources of nearby mobile devices in Cirrus clouds [79]. The application used RollerNet and Haggle trace to emulate the Serendipity functionality, where fixed size of independent tasks are disseminated among the mobile devices. Although the authors considered the intermittent connectivity, computing capacity of the devices and quantification of network signal strengths were untouched.

Li et al. introduced a device-to-device (D2D) communication framework for exploiting computing resources of mobile devices in the vicinity to execute application tasks that overcome the limitation of cellular connectivity [20]. To discover the computing resources, subtask distribution and retrieval in these intermittent devices, two access schemes had been proposed that provide improved performance in terms of computation speedup and monetary savings. To solve the mobility problem of mobile devices, the authors proposed an offloading heuristic based on location information of the mobile devices [19]. To schedule a task, the scheduling algorithms take into account communication latency in addition to associativity time which is calculated from historical location trace of a mobile device.

To optimize the applications execution performance, Le et al. provided a collaborative infrastructure among nearby mobile devices by leveraging Wi-Fi Direct technology that not only minimizes energy consumption but also expands the hardware capability of mobile devices [37]. The simulation results also reveal that task execution with nearby mobile devices is more beneficial compared to remote cloud server-based execution due to higher communication latency of cellular networks. In [38], Balasubramanian et al. exploited the resources of nearby mobile devices to provide Infrastructure as a Service in the Mobile Ad-hoc Cloud Computing environment. They also outlined necessary architecture and algorithms to create a pool of devices with dedicated resources and efficient utilization of those resources.

2.4 Task Execution Approaches in MDC

Different task execution approaches that have been exploited to utilize idle resources of nearby mobile devices are described in the following sections.

2.4.1 Voluntary Task Execution

The increasing number of smartphones has brought an enormous opportunity of code offloading to nearby mobile devices. A good number of recent works have demonstrated the benefits of exploiting idle resources of nearby mobile devices compared to executing applications on remote clouds. For making the offloading decision and coordinating the activities few researchers involved a trusted third party while few works gave the responsibility to the user mobile device. Different approaches for voluntary task execution are described in the following sections.

2.4.1.1 Execution with local coordination

In [16], Mtibaa et al. implemented an emulation testbed which showed the effectiveness of MDC compared to remote cloud or cloudlet based code offloading solutions. The authors showed that offloading tasks to nearby mobile devices can save up to 50% execution time and 26% energy compared to master cloud-based execution. However, the authors did not consider heterogeneous computational capacities of the nearby devices. Furthermore, the system assumed all devices having the same energy level and the number of offloadable tasks was fixed at 50%, which is not often practical.

Later, the authors provided a generic computation offloading framework to heterogeneous devices including master cloud and cloudlets [27]. The framework maximized the computational gain with respect to response time, energy consumption and network lifetime. However, there were no consideration of execution dependency among the application modules, expected data rate and available energy level of the devices.

Fernando et al. exploited a work-stealing method [64] on a set of mobile devices for better load sharing among the worker nodes in [80]. It was implemented using Bluetooth technology. Later, the authors exploited Wi-Fi Direct technology to utilize the computing resources of nearby mobile devices and introduced a mobile crowd that performed as a complement to the remote cloud [41]. For scheduling, a task, task heterogeneity, available resource at a worker, and dynamism were identified as the key challenges for which they employed a preemptive work-stealing mechanism on a set of worker nodes to balance the amount of workload and minimize the execution time. The model worked only for independent tasks and it did not consider the computing capacity of the devices, resulting in stealing jobs from weak workers frequently which in turn causes poor system throughput.

To prolong its battery lifetime, Truong-Huu et al. proposed a dynamic opportunistic offloading algorithm to determine the feasibility of offloading to a certain mobile device [65]. The optimization model was formulated as a Markov decision process with an objective to minimize the execution cost while considering different types of cost parameters (i. e., computation cost, communication cost, and penalty cost), resulting in a higher utilization of resources of nearby worker devices. However, their scheduling may lead to several levels of failed execution due to offloading only based on availability time and not considering the resource capacity of worker devices. Moreover, the execution of all tasks in parallel also leads to an impractical assumption of the task graph of an application.

Gao et al. introduced a multi-objective optimization model considering task execution time and resource consumption to select a set of target workers in an MDC environment with multiple devices containing multiple tasks [66]. Later, a set of heuristics has been applied to the model to achieve the minimum collective execution time while consuming the lowest amount of resources. In [39], Balasubramanian et al. formulated a model to ensure the parallel execution of tasks with a minimum number of worker devices. A composition score was calculated based on the shared resources of the device (CPU, Memory, and Storage) to guarantee the selection of the best available devices for task execution.

Lin et al. proposed a code offloading framework named Circa that exhibits the feasibility of code offloading in the proximity of nearby mobile devices exploiting iBeacons, a wireless location-based transmitter system [35]. The system used an efficient and fair task allocation algorithm to disseminate tasks of an application among reliable worker devices. In [36], Guiguis et al. delineated transient clouds, a collaborative computing framework for the offloaded execution of tasks with the help of nearby mobile devices. The authors explored centralized and distributed approaches to allocate tasks among mobile devices according to their capabilities. A modified Hungarian algorithm had been introduced in the centralized approach to balance the workload and Distributed Hash Tables were used to minimize the communication cost in the distributed mechanism.

2.4.1.2 Execution with external coordination

To further minimize the task completion time Habak et al. proposed Femto-Cloud [21], a dynamic and self-configuring cluster head based MDC architecture coordinated by a Cloudlet. The authors tried to maximize the overall workload of the participating devices in the cluster. The improvement was achieved because of applying priority based task assignment and earliest deadline heuristic on task assignment and result collection, respectively. However, they didn't consider signal strength (or data rate) offered by the devices and their residual energy levels. Moreover, these works incurred a huge burden on the potential workers due to a biased assignment of tasks.

Pandey et al. proposed a robust and distributed computing framework named Maestro to support concurrent execution of mobile application tasks in an MDC where each mobile device plays the role of a service provider, service requester, and broker [67]. Tasks were scheduled according to dependency whereas a replication and deduplication-based task scheduling have been applied for critical and similar tasks, respectively to achieve robustness in the system [81].

2.4.1.3 Comparative characteristics

The key characteristics of some of the voluntary task execution approaches are summarized in Table 2.1. Note that all of the above works emphasized the benefits of MDC technology that distributes computing loads to other mobile devices. However, most of them ignored inter-dependencies among the tasks while utilizing the parallel execution to speed-up the execution of the application [16, 21, 27, 41, 65, 66]. Furthermore, whether a target mobile device is reliable for code offloading (or not) has not been quantified for its selection. In this work, we develop a code offloading framework for the MDC system that makes a tradeoff between execution speedup and reliable execution of codes [62]. The performance is optimized through selecting devices that are reliable, offer higher computing capacities, signal strengths, and energy levels.

Table 2.1: Summary of voluntary task execution approaches

State-of-the-art-works	**External Coordination**	**Worker Capacity**	**Worker Reputation**	**Task Heterogeneity**	**Task Dependency**
Mtibaa et al. [16]	X	X	X	X	X
Habak et al. [21]	✓	✓	X	X	X
Mtibaa et al. [27]	X	✓	X	✓	X
Lin et al. [35]	X	✓	✓	✓	X
Guiguis et al. [36]	X	✓	X	✓	X
Balasubramanian et al. [39]	X	✓	X	✓	X
Fernando et al. [41]	X	✓	X	✓	X
Truong-Huu et al. [65]	X	X	X	✓	X
Gao et al. [66]	X	✓	X	✓	X
Pandey et al. [81]	✓	✓	X	X	✓
Saha et al. [62]	✓	✓	✓	✓	✓

2.4.2 Necessity of a Payment Mechanism for Workers

The actual benefit of the emerging MDC technology can only be harvested through the effective participation of mobile worker devices in the computation process. For the deployment of a large scale MDC system, increasing participation of users is a prerequisite. Voluntary approaches cannot motivate workers to participate in executing application tasks as each execution requires a certain amount of computational resources and bandwidth charges, resulting in a monetary payoff for a worker. For this reason, a worker device needs a payoff as a compensation of the used resources, stimulating to share their idle resources for executing more application tasks of different users. Moreover, high-quality workers can be supported with additional payment as an incentive-based on their quality of execution. In such cases, incentive models can be introduced to encourage worker devices for sharing their idle resources [82]. The introduction of a reward giving mechanism can incentivize the used resources of mobile devices, increasing motivation for the worker devices.

2.4.3 Paid Task Execution in MDC

MDC-based computing systems exploited a resource trading mechanism to provide payment of the worker devices where a resource-constrained mobile user acts as a resource-requester or buyer and a nearby worker mobile device with shareable idle resources acts as a resource-provider or seller. In addition to general payment policy [15, 34, 59, 68, 83], auction mechanisms are widely used to provide payment of the workers in an MDC system [22, 58, 60, 69, 70, 71, 84]. An auction mechanism addresses the conflicts between the buyer and the seller arranges a competition to fairly allocate the resources, determines appropriate payment for the allocated resources and ensures the truthfulness of the corresponding entities [85, 86].

In a few state-of-art-works, the conventional forward auction mechanism has been employed where a particular resource or service is sold with the highest amount of bid. On the contrary, a number of literature works motivated in designing a reverse auction-based payment mechanism as the roles of the buyer (or user) and the seller (or worker) are changed in an MDC system. In such cases, a buyer placed a request for a resource and many sellers bid declaring a minimum amount of value to be paid for the resource and, finally, the bidder with the least demand value won the auction.

The following sections describe the state-of-the-art payment strategies used for the payment of workers in an MDC system.

2.4.3.1 General payment strategy

In [15], Miluzzo et al. provided an outline of a payment mechanism for resourceful nearby mobile devices that run compute-intensive offloaded tasks of an application in an MDC system. Though the payment mechanism considered execution cost parameters such as waiting time and bandwidth usage, few other influential parameters like worker reputation and workload, task interdependency, etc. were untouched in the system.

Asghari et al. proposed a self-organizing and distributed leader selection framework where the role of the leader is to discover resources and balance the consumption of energy in an MDC [68]. The leader selection process introduced a multi-player first-price sealed-bid auction mechanism to select a leader with minimum resource discovery cost and provide payment for the leadership role. A credit-based incentive model also has been introduced for the payment of the worker devices for sharing resources for the execution of tasks. However, the consideration of remaining energy as the only parameter may lead to the selection of an untrustworthy worker as a leader. Moreover, the leader selection process can consume a significant amount of time if the network contains a large number of worker devices hampering the quality of the task execution.

Noor et al. in [59], created a mobile cloud architecture that proposed a virtual cloud with mobile phones connected through different network operators. The system utilized the unused resources of the mobile phones to execute the submitted task of a particular user and applied a reputation based task allocation strategy to assign tasks to different mobile phones. Though the model achieves trustworthiness through rating points, it poses the latency problem of cloud architecture due to continuous communication between mobile devices through a cellular network. Moreover, they didn't provide any incentive to reputed workers in the system let alone QoE.

The user mobility patterns and its opportunistic contact rates with nearby devices were taken into consideration for determining the appropriate devices for offloading by Wang et al. in [34]. The authors developed a convex optimization technique to determine the amount of computation to be offloaded to other devices and results showed the efficiency of the algorithm in terms of higher computation success rate. A worker device was paid according to the amount of computation where the unit price for the computation of a task was determined by the user device.

Balasubramanian et al. designed a reinforcement learning-based autonomous energy management framework that makes use of energy threshold to select reliable worker devices for the offloading of tasks [83]. They also modeled a reward policy where the worker devices were rewarded based on the rate of energy consumption.

2.4.3.2 Reverse auction mechanism

In [22], Wang et al. considered a game-theoretic approach to find an equilibrium point on user cost and worker profit for an optimal allocation of tasks in worker devices. Though the system gains benefit in the context of both buyer and seller devices, it tried to minimize user cost in the context of remote cloud price, which is often not practical. They improved execution performance further by developing auction mechanisms considering task heterogeneity [58]. Bid winners were paid with the amount of immediate next bidding price for a task, leading to overpayment. Though the algorithms achieved desirable properties of an auction mechanism none of them considered workload and reliability of mobile devices to execute the task.

In [60] Tang et al. proposed a broker based mobile cloud system that uses a double-sided bidding mechanism to allocate idle resources of mobile users to resource-demanding mobile users. The authors provided two algorithms with a game-theoretic approach to find an equilibrium point that maximizes the benefits both for resource buyer and seller devices. However, like the other works, this work also concentrated only on task allocation and price determination strategy whereas parameters like worker device capacity, reliability was out of their consideration.

Li et al. proposed an optimization model to minimize the transmission energy for offloading tasks in ad hoc mobile cloud [69]. To encourage the participation of mobile devices, the system introduced an auction-based task offloading mechanism that provides payment to worker mobile devices for sharing their vacant resources in collaborative computation. Though the system significantly reduces the energy consumption of a user device, it cannot guarantee a feasible solution due to not considering the budget of a user. Moreover, the lack of consideration in worker reliability may lead to the selection of poor quality workers that hampers the user

QoE.

2.4.3.3 Regular auction mechanism

To increase the participation of mobile devices, an forward auction model has been proposed by Wang et al. in [70] that trades resources between task owners and worker devices. Resource allocation and price estimation were determined through a distributed algorithm whereas a payment evaluation procedure detected dishonest sellers in the system. Since bids were submitted privately to the selected participants, the optimal result couldn't be guaranteed from the system.

He et al. formulated the collaborative task execution with the help of nearby mobile devices and payment for used resources as a social welfare maximization problem [71]. Due to the NP-hardness of the formulated optimization function, an alternative primal-dual based online auction algorithm was devised to make the task allocation and payment decision in polynomial time. Later, He et al. proposed a Vickrey-Clarke-Groves (VCG)-based online auction algorithm to provide payment of the workers for their used resources in a mobile edge cloud environment [84]. The system adopted a dynamic pricing policy for the used resources due to the dynamic participation of mobile workers. However, both the systems were unable to recognize qualified workers let alone providing incentives for their quality of execution.

2.4.3.4 Comparative characteristics

The key working principles of different paid task execution strategies are summarized in Table 2.2. All the above works focused on a general objective to minimize execution time and cost of a user where the workers were compensated with the

Table 2.2: Summary of paid task execution approaches

State-of-the-art-works	**External Coordination**	**Worker Capacity**	**Worker Reputation**	**User Budget**	**User QoE**	**Additional Incentive**	**Payment Mechanism**
Miluzzo et al. [15]	X	X	X	X	X	X	General
Wang et al. [34]	X	✓	X	✓	X	X	General
Noor et al. [59]	✓	✓	✓	X	X	X	General
Asghari et al. [68]	X	✓	X	X	X	X	General
Venkatraman et al. [83]	X	✓	X	X	X	X	General
Wang et al. [22]	X	✓	X	✓	X	X	Game Theory
Wang et al. [58]	✓	✓	X	X	X	X	Reverse Auction
Tang et al. [60]	✓	✓	X	✓	X	X	Game Theory
Li et al. [69]	X	✓	X	X	X	X	Reverse Auction
Wang et al. [70]	X	✓	X	X	X	X	Forward Auction
He et al. [71], [84]	✓	✓	X	✓	X	X	Forward Auction
Saha et al. [63]	✓	✓	✓	✓	✓	✓	Reverse Auction

bid amount. There were no consideration of additional payment as incentive following quality execution of a task. Moreover, most of the works not only ignored the reliability of devices while selecting the workers but also neglected the interdependency among the tasks. The key philosophy of this work [63] is to select a set of reliable worker devices to execute an application task considering its dependency with others so as to minimize the user execution cost and to maximize the quality of execution. This novel strategy of rewarding worker devices with an additional incentive for offering high quality execution would help our system to attract more workers to participate and makes such MDC system sustainable.

2.5 Summary

In this chapter, we provide a detailed discussion on existing methodologies for the execution of tasks in MDC to enhance the execution performance of compute-intensive applications. In Chapter 1, we analyzed various design parameters to be addressed by an effective code offloading solution for exploiting idle resources of nearby worker devices in an MDC environment. Based on that, we studied the state-of-the-art voluntary and paid task execution approaches in this Chapter. Voluntary task execution approaches tried to minimize task execution time while skipped the reliability of the workers and only addressed the parallel execution of independent tasks. Paid task execution approaches mostly employed auction mechanism for the payment of the workers while unable to provide any quantification of user QoE. In the next chapters, we aim to provide solutions that can diminish the challenges of state-of-the-art-works.

Chapter 3

Voluntary Task Execution in MDC

We have discussed and compared the state-of-the-art works on different task execution approaches in MDC system proposed by the researchers in the previous chapter. By motivating from the limitations of the related works and to overcome the challenges of task execution in MDC, we develop and evaluate a reliable worker selection strategy in this chapter.

3.1 Introduction

With the advent of different mobile computing technologies, the expectation of mobile device users also has been increased dramatically on the functionality and quality of experience of user applications. To meet these expectations, mobile devices have opened up a plethora of computational infrastructure and engaged high accuracy sensors, large volumes of multimedia data, and complex artificial intelligence algorithms to provide improved performance for compute-intensive applications to the end-users. However, the execution of these applications on the user mobile device still cannot produce a feasible solution due to constrained resources, resulting in an offloaded execution with the help of remote cloud or cloudlets. Furthermore, remote cloud or cloudlets also cannot produce a satisfactory performance for having longer communication latency and resource scarcity, respectively. In a consequence, Mobile Device Cloud (MDC) has been introduced to provide a collaborative cloud computing environment exploiting the idle resources on nearby mobile devices. The opportunistic resources of the encountered nearby mobile devices have made MDC technology to alleviate the resource constraints of a user to execute compute-intensive applications like object, face, pattern and speech recognition, disaster response, natural language processing, m-health, and reality augmentation, etc.

Usually, in the MDC system, the mobile worker devices act as voluntary resource providers and participate voluntarily in the execution process of an offloaded application. Execution of tasks in an MDC system is more challenging than cloud or cloudlets due to the opportunistic nature of the nearby resources. The execution performance of an offloaded application in the MDC system highly depends on the smart utilization of the resources of nearby worker devices. Efficient strategies need to be employed so that the maximum resources of qualified workers can be utilized to execute the application within the required deadline. In voluntary participation, the primary challenge lies in the selection of a set of reliable worker devices to speedup the execution of an application.

In the literature, a good number of researches had been conducted to utilize the resources of nearby mobile devices for the execution of an offloaded application. In a testbed implementation, Mtibaa et al. showed that offloading tasks to nearby mobile devices can save upto 50 % execution time and 26 % energy consumption compared to remote cloud-based offloading mechanism [16]. Truong-Huu et al. also proposed a dynamic opportunistic offloading algorithm that utilizes the computation resources of nearby mobile devices to minimize the task execution time and prolong the battery lifetime of a mobile user [65]. However, none of them considered the resource capacity of the worker devices and interdependency among the tasks, resulting in poor execution performance.

In [36], Guiguis et al. introduced transient clouds for executing offloaded application tasks and developed centralized and distributed algorithms to exploit the idle resources of nearby mobile devices. Balasubramanian et al. modeled a composition score to select the worker devices that maximize parallelism in the execution of tasks with a minimum number of workers [39]. In [41], Fernando et al. initiated a preemptive work-stealing mechanism to utilize the computing resources of nearby mobile devices that minimized the task execution time and balanced the workload of worker devices. To select the opportunistic worker devices, Gao et al. formulated a multi-objective optimization model considering task execution time and resource consumption that applied a set of heuristics to minimize the task execution time and resource consumption [66]. However, these approaches failed to grab the potential benefits of the opportunistic resources due to the selection of unreliable workers that resulted in frequent failed execution of tasks. Moreover, running the decision algorithm on the user mobile device also imposed an extra burden on the available resources and energy of the user device.

Habak et al. applied priority-based task assignment and earliest deadline heuristic to minimize the task completion time in a cluster head based MDC architecture which was coordinated by a Cloudlet [21]. In [81], Pandey et al. developed a robust and distributed computing framework that applied replication and deduplication mechanism to facilitate concurrent execution of application tasks in an

MDC system.

All these works presented the benefits of mobile device cloud technology that exploits the resources of nearby mobile devices. However, most of the works ignored the reliability of the worker devices and inter-dependency among the tasks that resulted in frequent resubmission and longer execution time, respectively in the execution of application tasks.

Considering these observations, in this thesis, we first develop an opportunistic offloading mechanism named TESAR that optimally allocates offloaded tasks to the members of MDC. The key idea is to select the most reliable worker devices with adequate resources that are capable to execute a submitted task. The main goal of our TESAR system is to minimize the task execution time considering the device-specific parameters like mobility, available energy, signal strength, computing capacity, and application-specific parameter like module dependency. In TESAR, apportion and dissemination of application tasks, accumulation of collected results and selection of the worker devices is coordinated by a cloudlet. According to the demand of an application, the system can be tuned to select the worker devices that provide minimum execution time or maximum reliability or makes a tradeoff between these two. We present performance analysis and results for task completion time, communication latency, and communication overhead with varying number of tasks, workers and parallel tasks. The results show that the proposed TESAR mechanism significantly outperforms state-of-the-art works - OMDC [27] and Honeybee [80].

What follows next are the summarization of the key contributions of this chapter:

- We develop a code offloading framework, namely TESAR, that speedup the computation and maximizes the reliability of an MDC system.

- The task allocation algorithm is formulated as a MILP problem to maximize computation speedup and reliability
- The system makes a tradeoff between execution speedup and reliable execution of tasks based on the application requirement. The objective function has been designed in such a way that it can be used to maximize the reliability or computation speedup or to make a tradeoff between these two.
- An extensive simulation has been carried out to evaluate the performance of the proposed TESAR system.

The rest of this chapter is organized as follows. In Section 3.2, we describe the code offloading architecture and framework for MDC. The mathematical formulation of our proposed TESAR system is discussed in detail in Section 3.3. The experimental testbed setup and results of performance evaluations are presented in Section 3.4. Finally, we conclude the paper in Section 3.5.

3.2 Code Offloading Framework for Mobile Device Cloud

An MDC system comprised of heterogeneous set of mobile devices including laptops, smart-phones, palmtops, tabs/pads, wearable devices like watch, glass, etc. The amount of computation, communication and storage resources of the devices greatly vary from each other. Typically, they have small scale computation and storage capacities that remain idle most of the time on a large number of devices. These idle computing resources can be accumulated together to run heavy weight applications that are not executable on their devices within delay deadline. In MDC, these idle resources act as small virtual machines (VMs) for executing codes of nearby users. The user can configure the VM of his/her device by limiting the amount of resources (CPU, storage, bandwidth usage, etc.) to serve applications of other users and to gain benefits. The amount of resources shared by a device is reconfigurable at any point of time and users have complete control of managing resources of their devices. What follows next, we present a framework with functional components and the system model for code offloading in an MDC system.

3.2.1 Compute-Intensive Code Offloading Framework

Limited resources of a mobile device restrict it to deliver the results of an application in time. Application codes requiring higher computing power, need a great number of CPU cycles, storage and thus its battery drains out very fast. However, if the application is partitioned into a number of tasks that are offloadable to nearby mobile devices, then the execution load gets distributed and completion time of the application gets reduced. To make use of the resources of nearby mobile devices and to enable parallelism in the execution of tasks, we propose a compute-intensive code offloading framework named TESAR as shown in Figure 3.1. The detailed descriptions for each of the functional modules are given below.

- **Task Profiler (TP):** The *TP* module receives an application with an execution deadline through the task receiver component from a mobile user. It then uses the workload analyzer component to split the application into a set of atomic tasks where each atomic task contains a number of instructions for offloaded execution. Using static analysis, it determines the tasks which are offloadable. The dependency estimation component is responsible for deter-

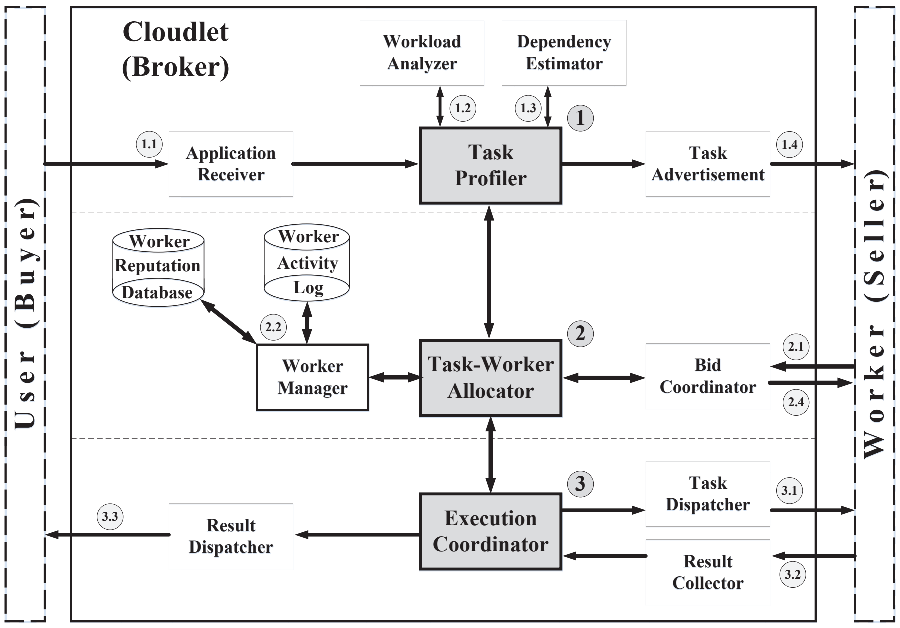


Figure 3.1: Compute-intensive code offloading framework

mining the interdependency among tasks which is used to identify execution order, facilitating parallel execution. The relative dependencies determine tasks that can be executed in parallel so as to minimize the overall execution time of the application. A rooted tree construction algorithm for determining the task dependencies is presented in section 3.3.2. For each task, it creates a tuple containing task number, instruction size, and task deadline. The *TP* then hands the tasks over to the task advertisement component for advertising the tasks to the worker devices. It also shares advertised task information with the task-worker allocator (*TWA*) for cross verification purposes.

- **Task-Worker Allocator (TWA):** The *TWA* module is the core of the proposed system, and it controls and coordinates functionalities of other modules. It triggers the *TP* to collect user applications and submits advertisements after profiling the application tasks. It collects worker bids through the bids coordinator component. The worker devices send their device-specific information which contains device ID, associativity time, and a set of bids. All the collected bids and device information are then transferred to the TWA module to determine the winners among the submitted bids. Notifications of winning bids are also dispatched through the same interface to selected worker devices.

The *TWA* module interacts with the Worker Manager component to determine the feasibility of workers to execute tasks successfully. The Worker Manager component monitors and stores the device-specific information that can be used during task allocation. Specifically, it stores device ID, signal strength, clock cycle, available energy, reputation value, and updates it periodically. It evaluates submitted bids to determine the quality and reliability of worker devices through historical trace using worker reputation database and activity log. It is also responsible for updating the reputation of a mobile device after successful completion or failure of execution of a task respectively. The reputation is given as a means of recognition that is increased/decreased based on the successful/failed execution of a task. Whenever a new device gets registered, the *TWA* module communicates with the Worker Manager component to collect and store the reputation information. Detail procedure of calculating the reputation values is presented in the section 3.3.5.

After accumulating all this information, the *TWA* runs a worker selection algorithm to determines the optimal mapping of individual worker devices to different application tasks. The task-worker mapping is transferred to the execution coordinator (*EC*) for scheduling of tasks along with a notification to the winners from the set of candidate worker devices. Moreover, the *TWA* ensures that no device will be assigned two tasks that can run simultaneously, i.e., having zero relative execution dependency. Detail operation of the offloading decision-making mechanism is presented in the section 3.3.

- **Execution Coordinator (EC):** On reception of the task-worker mapping list from the *TWA*, the *EC* calls the task dispatcher component to schedule the tasks in order. After successful execution, results are collected and transmitted back to the user device. Failed executions are reallocated by the *TWA* to the next available bidder.

The critical part of the proposed *TESAR* framework is the identification of the parent-child relationship between tasks and mapping *worker* devices to offloadable tasks. Although we provide a complete architecture for an MDC system, this work, particularly, focuses on *TWA* that optimally assigns computation tasks to *worker* devices. However, for parallel execution of offloadable tasks, identification of parent-child relationships among the tasks is a prerequisite. For this reason, the *TP* executes Algorithm 1, described in section 3.3.2, to construct a dependency tree between the tasks, and then the *TWA* determines the optimal assignment of computation tasks to *worker* devices.

3.2.2 System Model for Collaborative Computation in MDC

Our proposed collaborative code offloading system has two tiers. A device that requires to offload codes to others for faster execution, known as *user* device and the (*worker*) devices where the compute-intensive codes are offloaded for execution

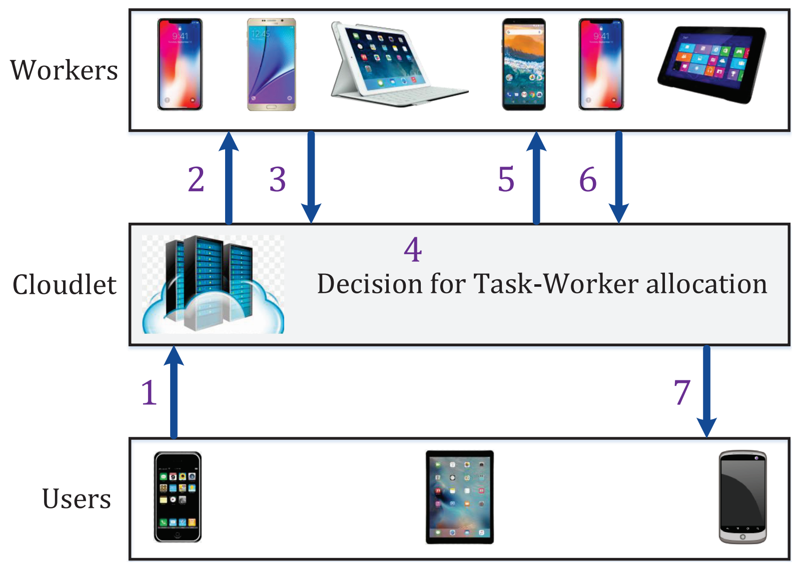


Figure 3.2: Collaborative computation in a mobile device cloud

resides in tier one. The cloudlet and wireless communication infrastructure that facilitate collaboration are at tier two. The mobile devices are connected to the cloudlet through Wi-Fi Access Points (APs). A (*worker*) device communicates with the cloudlet and show its interest to run any compute-intensive applications. The cloudlet collects information of the (*worker*) devices, makes scheduling plan and offloads tasks of the application in favor of the *user* devices. It also performs the mapping of application tasks to the VMs of (*worker*) devices and offloads accordingly. It acts as a broker by partitioning the application into tasks, coordinating the distribution of tasks to remote devices and accumulating results obtained from *worker* devices.

Let us assume, there are sufficiently available devices in the MDC system to allocate each task in a compute-intensive application, and there is no task unbidden, while the cloudlet broadcasts. We also consider that a candidate worker device (as known as, candidate device) agrees to execute whatever amount of task is allocated in the MDC. Figure 3.2 illustrates the details of the proposed system model and decision process of MDC and thereby are explained in following steps.

S1. When a user requires to offload an application for faster execution, it submits the application to the cloudlet.

S2. As soon as the cloudlet receives the application, it splits the application into atomic tasks and immediately advertises the tasks to the workers.

S3. In response to the advertisement, interested workers submit their bids for corresponding tasks to the cloudlet.

S4. Being an auctioneer, the cloudlet receives bids from the workers and it determines winner set from the candidate bids.

S5. Based on the task-worker mapping list, the cloudlet allocates each task to the corresponding worker device, as determined in step 4.

S6. After completion of task execution, the worker immediately returns the result to the cloudlet.

S7. Finally, the cloudlet aggregates the results of each task, collecting from individual workers, and returns to the user device.

In this chapter, the initial reputation of mobile devices has been chosen randomly. Moreover, a number of works have already been carried out to determine the associativity time of mobile devices [30],[87] and we have adopted [30] since it exploits historical trace of mobile devices and fairly captures the MDC environment.

3.3 Optimization Problem Formulation

The main contribution of this work is to optimally map the execution of application tasks on different worker devices so that the total execution time can be minimized while preserving higher reliability. This is a challenging issue as highly reliable devices may not be equipped with greater computing capacity. To distribute the tasks among the worker mobile devices, we have devised a MILP objective function that considers a number of placement and capacity constraints. Particularly, the objective function makes the selection of remote mobile devices considering execution time and reliability parameters jointly to maximize the gain.

In this subsection, we first define an application model for compute-intensive MDC system. Then, we determine the sets of parent and child tasks of the application and time required for executing those locally. After that, we derive an objective function that will select remote mobile hosts in such a way that total execution time gets minimized and reliability of execution in respect of device reputation get maximized while satisfying a number of constraints. Table 3.1 shows the list of notations used in this work.

Binary variable $y_{m,k} = \{1, 0\}$ is used to indicate whether a task $m \in \mathcal{M}$ is executed on device k or not. Since the application contains both local and remote executable tasks, the total number of tasks therefore, can be obtained as,

$$\mathcal{M} = |\mathbb{R}| + |\mathbb{L}|. \tag{3.1}$$

3.3.1 Compute-intensive Application Model

Each application in an MDC system can be considered as a directed graph *G($\mathcal{M}$, e)* with $\mathcal{M}$ processing tasks where each task performs a specific operation of the

Table 3.1: List of notations for voluntary task execution

$\mathcal{M}$	Set of tasks in the application
$\mathcal{K}$	Set of available devices
Π_m	Set of parents of task m
L	Set of Leaf nodes
$\mathbb{R}$ / $\mathbb{L}$	Set of tasks executed Remotely / Locally
$\mathcal{S}_m^o$ /$\mathcal{S}_m^{o'}$	Instruction size of task $m \in \mathbb{R}$ / $m \in \mathbb{L}$
$\mathbb{V}_m$	Output instruction size of task $m \in \mathcal{M}$
μ_k	Execution speed of device $k \in \mathcal{K}$
$\mathcal{T}_m^l(o)$ / $\mathcal{T}_m^l(o')$	Local device execution time of task $m \in \mathbb{R}$ / $m \in \mathbb{L}$
$\mathcal{T}_{m,k}^x$	Remote execution time of task $m \in \mathbb{R}$ in device $k \in \mathcal{K}$
$\mathcal{T}_{m,k}^t$ / $\mathcal{T}_{m,k}^r$	Input transmission /Output reception time of task $m \in \mathbb{R}$
$\mathbb{B}_k^d$	Uplink bandwidth between cloudlet and device $k \in \mathcal{K}$
$\mathbb{B}_k^{d'}$	Downlink bandwidth between cloudlet and device $k \in \mathcal{K}$
$\mathcal{E}_{m,k}^x$ / $\mathcal{E}_{m,k}^t$	Execution /Input transmission energy of task $m \in \mathcal{R}$ in $k \in \mathcal{K}$
$\mathcal{E}_{m,k}^r$	Output transmission energy of task $m \in \mathcal{R}$ in $k \in \mathcal{K}$
$\phi_{n,m}$	Percentage of dependency of task $m \in \mathcal{M}$ on its parent $n \in \mathcal{M}$
γ_k	Signal strength of device $k \in \mathcal{K}$
η_k	Associativity time of device $k \in \mathcal{K}$
E_k	Available energy of device $k \in \mathcal{K}$
Ω_k	Reputation of device $k \in \mathcal{K}$

application. The directed edges between tasks form a rooted tree to define dependencies between tasks. This rooted tree determines execution flow of the application as shown in Fig 3.3. Each edge e consists of two weights - $\mathcal{S}_m^o$ and $\mathbb{V}_m$, the input and output instruction sizes, respectively, of the task $m \in \mathcal{M}$. A dependent child task can be executed only when output of the parent task is available. However, all parallel tasks can be executed simultaneously based on availability of the *worker* devices.

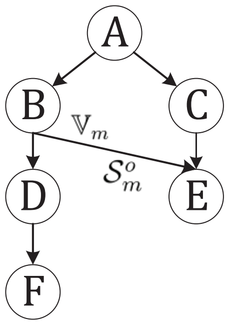


Figure 3.3: Dependency tree of the application tasks

The execution time of an application is inversely related with the number of dependency levels since the later decreases execution parallelism. The total execution time of the application is the maximum execution time of a subtree including communication delays. In this work, local and remote executions have been used interchangeably with *user* and *worker* device executions, respectively.

3.3.2 Construction of Rooted Tree of Tasks

A task n is said to be the parent of another task m if, there exists a dependency between the execution of tasks m and n that is, task m requires the output from n at any particular instance of its execution. A task may require outputs form more than one parent tasks and the set of all parent tasks of task m is given by Π_m. Again, a task has been considered as leaf, if it has no child dependent on it. The set of leaf tasks is given by L. Algorithm 1 summarizes the steps of determining Π_m

Algorithm 1 Algorithm for constructing parent and leaf sets of tasks

INPUT: $G(\mathcal{M}, e)$: *A graph of tasks* ($\mathcal{M}$) *of an application with edges* (e)

P : *Parent task from which the execution begins*

OUTPUT: Π_m : *Set of parent tasks for each task* $m \in \mathcal{M}$

L : *Set of leaf tasks*

```
1.  $Set\ \Pi_m \leftarrow \Phi,\ Color_m \leftarrow White,\ \forall m \in \mathcal{M}$
2.  $Set\ L \leftarrow \Phi,\ Queue\ Q \leftarrow \Phi,\ \Pi_P \leftarrow \Phi$
3.  $Q.enqueue(P)$
4.  while $Q$ is not empty do
5.    $n \leftarrow Q.dequeue()$
6.    for all $m \in \mathcal{M} | (n, m) \in e$ do
7.      $\Pi_m \leftarrow \Pi_m \cup \{n\}$
8.      if $Color_m = White$ then
9.        $Q.enqueue(m)$
10.     end if
11.   end for
12.   for all $m \in \mathcal{M}$ do
13.     if $(n, m) \cap e = \Phi$ then
14.       $L \leftarrow L \cup \{n\}$
15.     end if
16.   end for
17.   $Color_n = Black$
18. end while
```

and L from the dependency graph $G(\mathcal{M}, e)$. Here, each element of e is a directional edge from parent task to child task.

3.3.3 Time Estimation for Local Execution

Time of execution for the whole application in the local device is the total time required for the completion of both local and remote executable tasks. Here, time required for executing task $m \in \mathbb{R}$ is given by, $\mathcal{T}^l_m(o) = \frac{\mathcal{S}^o_m}{\mu_{k|k=0}}$ where, $\mathcal{S}^o_m$ represents the size of task $m \in \mathbb{R}$ and μ_k represents the execution speed of device $k \in \mathcal{K}$; $k = 0$ represents the local device and all the remote devices under the cloudlet takes a nonzero value of k. Similarly, time required for executing task $m \in \mathbb{L}$ is represented as, $\mathcal{T}^l_m(o') = \frac{\mathcal{S}^{o'}_m}{\mu_{k|k=0}}$. Therefore, the total local execution time is obtained from the execution delay of all local ($\mathbb{L}$) and remote ($\mathbb{R}$) executable tasks,

$$\mathcal{T}(l) = \sum_{m=1}^{|\mathbb{R}|} \mathcal{T}^l_m(o) + \sum_{m=1}^{|\mathbb{L}|} \mathcal{T}^l_m(o'). \tag{3.2}$$

3.3.4 Time Estimation for Remote Execution

To calculate the execution time of offloadable tasks at remote hosts and unoffloadable tasks at local device, we need to consider the time for transmitting the tasks, execute the tasks and collect the results back to the cloudlet. The time for transmitting task $m \in \mathcal{M}$ from the cloudlet to device $k \in \mathcal{K}$ can be expressed as,

$$\mathcal{T}^t_{m,k} = (\frac{\mathcal{S}^{o'}_m}{\mathbb{B}^d_{k|k=0}} + \frac{\mathcal{S}^o_m}{\mathbb{B}^d_{k|k>0}}) \times y_{m,k}, \tag{3.3}$$

where, $\mathbb{B}^d_k$ represents the available bandwidth for data transmission from cloudlet to device $k \in \mathcal{K}$; $k = 0$ again represents the *user* device which executes the unoffloadable tasks. Now, the time for execution of task $m \in \mathcal{M}$ in device $k \in \mathcal{K}$

of MDC can be represented as,

$$\mathcal{T}^{x}_{m,k} = (\frac{\mathcal{S}^{o'}_{m}}{\mu_{k|k=0}} + \frac{\mathcal{S}^{o'}_{m}}{\mu_{k|k>0}}) \times y_{m,k}. \tag{3.4}$$

After completion of execution of a task, all devices including the *user* will transmit the result back to the cloudlet. The time required to send the execution result of task $m \in \mathcal{M}$ from device $k \in \mathcal{K}$ to cloudlet can be calculated as,

$$\mathcal{T}^{r}_{m,k} = \frac{\mathbb{V}_m}{\mathbb{B}^{d'}_{k}} \times y_{m,k}, \tag{3.5}$$

where, $\mathbb{V}_m$ represents the size of results produced after the execution of task m. Therefore, the total execution time of task $m \in \mathcal{M}$ in device $k \in \mathcal{K}$ can be obtained as,

$$\mathcal{T}_{m,k} = \mathcal{T}^{t}_{m,k} + \mathcal{T}^{x}_{m,k} + \mathcal{T}^{r}_{m,k}. \tag{3.6}$$

Now, for calculating the completion time of a task, we have summed up the total time required to execute the task from the beginning of execution of the application. This is performed by adding the execution time of the task ($\mathcal{T}_{m,k}$) with the total completion time of it's parent ($\mathcal{T}_n(r) | n \in \Pi_m$). However, a child may not always depends completely on the result of it's parents. It may start independently and after completion of a certain percentage of execution it requires the results from it's parents. This dependency relation is represented by $\phi_{n,m}$. Therefore, the total completion time of task $m \in \mathcal{M}$ is given by,

$$\mathcal{T}_m(r) = max\ (\mathcal{T}_{n,\bar{k}}) + \mathcal{T}_{m,k}(1 - \phi_{n,m}); \quad \forall n \in \Pi_m. \tag{3.7}$$

The total time required for execution completion of the application is therefore, the largest time required for completion among all the leaf tasks. The total time required for execution of the complete application is represented as,

$$\mathcal{T}(r) = max\ (\mathcal{T}_m(r)); \quad \forall m \in L. \tag{3.8}$$

In order to evaluate the completion time of an application in MDC and compare among different scheduling arrangements, scheduling speedup factor needs to be calculated. The speedup factor of a scheduling can be represented as,

$$\mathcal{T}(f) = 1 - \frac{\mathcal{T}(r)}{\mathcal{T}(l)}. \tag{3.9}$$

3.3.5 Calculation of Reputation Value

For successful execution of offloaded tasks reliable *worker* selection is a prerequisite. If a low performing or unreliable *worker* is selected for execution of a task, it turns into an unsuccessful execution, increasing response time and hence deteriorates the overall performance of offloading mechanism. Any device may advertise itself as a first-rate *worker* with high computing capability whereas its successful execution rate might be very poor. The reputation parameter can be used to guard against such unqualified *workers*. The reputation of a device is calculated as,

$$\Omega_k = \delta \times \Omega_k + (1-\delta) \times \frac{\sum\limits_{m \in \mathcal{R}} d_{m,k}}{\sum\limits_{m \in \mathbb{R}} y_{m,k}}, \tag{3.10}$$

where, δ is a relative weight parameter and takes value from the range [0,1]; $d_{m,k}$ is a binary variable having value 1 when, task $m \in \mathbb{R}$ is completed successfully in device $k \in \mathcal{K}$ and, 0 otherwise. Similarly, binary variable $y_{m,k}$ is set to 1, if $m \in \mathbb{R}$ is submitted to device $k \in \mathcal{K}$ for execution, and 0 otherwise.

While scheduling the offloadable tasks of an application, reputation of the mobile devices for task execution needs to be considered. Involving devices with higher reputation for execution of the tasks of an application increases the execution reliability. Calculation of average reputation of workers for scheduling all the tasks of an application can be expressed as,

$$\Omega_t = \frac{1}{|\mathcal{M}|}\sum_{k\in\mathcal{K}} \Omega_k \times y_{m,k}. \tag{3.11}$$

As soon as a registered worker device bids for a task, the *TWA* communicates with the worker manager component and loads reputation value (Ω_k) of that device. Whenever execution of a particular application ends, the *TWA* calculates reputation value of all the involved mobile devices and updates the reputation database of individual workers.

3.3.6 Expected Energy for Remote Execution

Since mobile devices suffer from the scarcity of energy, we need to calculate the total energy that will be required to offload a task to a *user* device. To calculate the total energy consumed to offload a task, we need to consider the energy required for the transmission and execution the task and collection of the result back. Energy required to transmit task $m \in \mathcal{M}$ from cloudlet to device $k \in \mathcal{K}$ is given by,

$$\mathcal{E}^t_{m,k} = \mathcal{T}^t_{m,k} \times \epsilon^t_k \times y_{m,k}, \tag{3.12}$$

where, ϵ^t_k represents the energy consumption rate for transmission by device $k \in \mathcal{K}$. Now, the energy required to execute task $m \in \mathcal{M}$ in device $k \in \mathcal{K}$ is expressed as,

$$\mathcal{E}^x_{m,k} = \mathcal{T}^x_{m,k} \times \epsilon^x_k \times y_{m,k}. \tag{3.13}$$

Similarly, energy required to transmit output of task $m \in \mathcal{M}$ from device $k \in \mathcal{K}$ to cloudlet is,

$$\mathcal{E}^r_{m,k} = \mathcal{T}^r_{m,k} \times \epsilon^r_k \times y_{m,k}, \tag{3.14}$$

where, ϵ^x_k and ϵ^r_k represents the energy consumption rate of device $k \in \mathcal{K}$ for execution and result transmission, respectively.

3.3.7 Optimal Selection of Mobile Workers

Now to select the optimal set of remote mobile devices for the execution of offloadable tasks, we need to choose those devices for which the execution delay of offloadable and unoffloadable tasks gets minimized while total reputation of all the scheduling devices gets maximized. The objective function for the selection of remote mobile devices is formulated as,
$Maximize:$

$$\mathcal{Z} = \alpha \times \mathcal{T}(f) + (1-\alpha) \times \Omega_t. \tag{3.15}$$

Here, weight factor α has been used to represent relative priority between application completion time and device reputation. The value of α can be determined by the requirement of the application. Time sensitive applications can set a higher of α while applications requiring high reliability can choose a lower percentage of α.

Constraints:

Each task should be executed in a single device at a time.

$$\sum_{k \in \mathcal{K}} y_{m,k} = 1; \quad \forall m \in \mathcal{M} \tag{3.16}$$

The offloadable computation time of the tasks through MDC should be less than local computation time of the whole application, *i.e.*,

$$\mathcal{T}(r) < \mathcal{T}(l). \tag{3.17}$$

Participating node signal strength should be greater than a certain minimum threshold, *i.e.*,

$$\gamma_k > \gamma; \quad \forall k \in \mathcal{K}, \tag{3.18}$$

where, γ_k represents the signal strength of device $k \in \mathcal{K}$ which is obtained through simulation process; and, γ represents the threshold value of signal strength what must be satisfied by a potential *worker*.

Participating device energy, after execution, should be greater than a certain minimum threshold, that is,

$$E_k > \mathcal{E}^t_{m,k} + \mathcal{E}^x_{m,k} + \mathcal{E}^r_{m,k} + \Psi; \quad \forall k \in \mathcal{K}, \forall m \in \mathcal{M}, \tag{3.19}$$

where, Ψ represents the energy threshold that a *worker* device must hold after the completion of the execution.

During the execution and transmission period, the participating devices will be available within the range of the cloudlet, *i.e.*,

$$\mathcal{T}_m(r) < \eta_k; \quad \forall m \in \mathcal{M}, \forall k \in \mathcal{K}, \tag{3.20}$$

where, η_k represents the associativity time of device $k \in \mathcal{K}$ with the cloudlet.

All the unoffloadable tasks ($m \in \mathbb{L}$) must have to be executed on the local device

$$y_{m,k} = \begin{cases} 1, & \text{if } k = 0 \quad \forall m \in \mathcal{M}, \forall k \in \mathcal{K} \\ 0, & \text{otherwise.} \end{cases} \tag{3.21}$$

Note that, the objective function of the proposed *TESAR* algorithm provided in equation (3.15) selects those mobiles for which total execution time in remote mobile device cloud is minimum and have the highest previous reputation of execution. It is a multi-objective mixed integer linear programming (MILP) problem that has both combinatorial and continuous constraints.

To solve the MILP problem, the NEOS optimization tool [88] has been used to find the impact of optimization function parameters and the optimal mapping between tasks and *worker* devices for task allocation and scheduling in *TESAR*. Two Intel Xeon E5-2698@2.3 GHz CPU with 192GB RAM has been used to find the optimal scheduling for an application containing 12∼15 tasks and 60∼80 mobile devices. Note that, with the increase of number of tasks and available mobile devices, real-time solution of *TESAR* becomes intractable in a typical cloudlet and thus the problem can be grouped as NP-complete one [89]. However, the constraints (3.16 - 3.21) facilitate us to significantly reduce the input sets in *TESAR* environment and thus the optimal solution was found in polynomial time.

3.4 Performance Evaluation

In this section, we discuss the emulation testbed that is used to implement the proposed task offloading algorithm *TESAR* and compare the obtained results with state-of-the-art works. We compare the performance of *TESAR* with the following algorithms:

- **OMDC:** In OMDC [27], the application tasks are assigned to different available *workers* in a round robin scheduling order.
- **Honeybee:** In Honebee [80], application tasks are scheduled on different *worker* devices based on availability in a purely random fashion. If a poor *worker* was chosen for a task, the work stealing mechanism is applied to take out the task (from the poor *worker*) and is executed later on a computation rich *worker*.
- **Random:** In this mechanism, the tasks are assigned to different *worker* devices randomly without considering the device status.

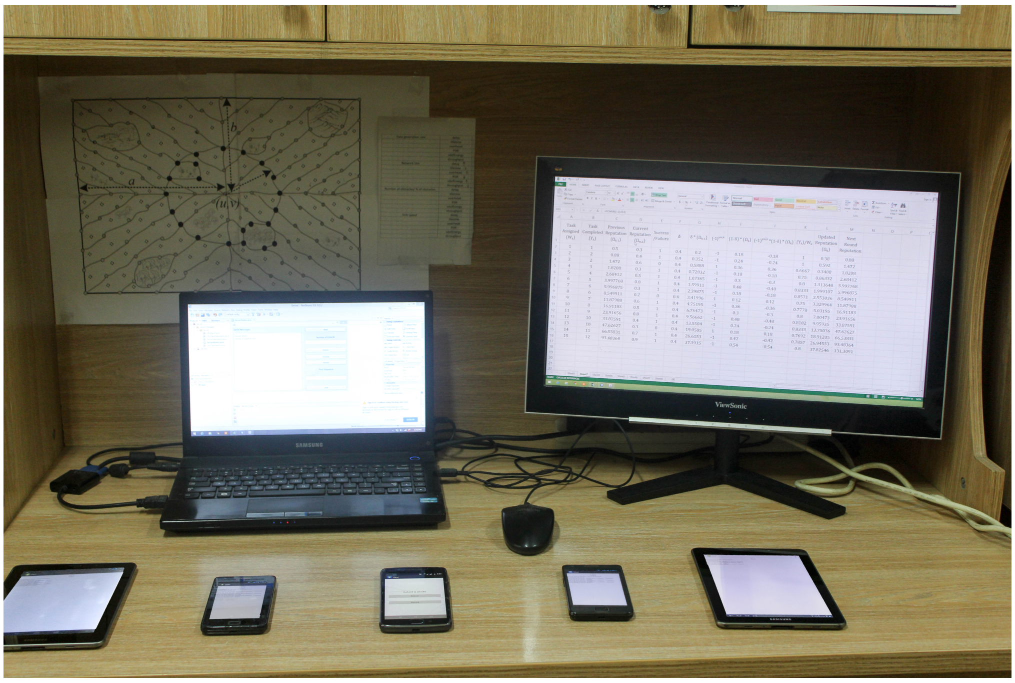


Figure 3.4: Emulation testbed

3.4.1 Experimental Testbed

To evaluate the performance of our proposed *TESAR*, an emulation testbed has been set up by implementing an Android application on a number of heterogeneous mobile devices. The cloudlet functionalities are implemented on a laptop through which all the mobile devices are connected. A sample snapshot of the emulation environment is shown in Figure 3.4. The mobile devices and the laptop communicate to each other via a Wi-Fi access point. Different parameters and their values used to carry out the emulation are summarized in Table 3.2.

We consider *prime number calculation* problem as an experimental prototype to represent a compute-intensive application. Generation of prime numbers with a large range requires a lot of computation. This particular problem can easily be

Table 3.2: Configuration and settings of devices

Device	**Model**	**OS Version**	**RAM**	**CPU**
Laptop (*cloudlet*)	ASUS ZenBook UX303LN	Windows 10	8GB	Core i5-5200U 2.20GHz
Cell Phone (*user*)	Sony LT18i	Android 4.0.4	512 MB	1.4 GHz Scorpion
Tablet PC (*worker*)	Symphony T8Q	Android 4.2.1	1 GB	Quad-core 1.2 GHz Cortex A7
Cell Phone (*worker*)	Walton Primo X2mini	Android 4.2.1	1 GB	Quad-core 1.5 GHz Cortex-A7

subdivided into several tasks that are passed through Algorithm 1 to construct the parent-child dependency tree. Then, we run the objective function on this set of tasks for distributing the execution of tasks on nearby *worker* devices. Figure 3.5 shows the android application interface which is used by a user/worker device to interact with the cloudlet.

In this experiment, the *prime number* problem produces primes between 1 and 300000, where the complete range is divided into tasks of different size. The number of tasks varies from 4 to 12 according to the need of the experiment. Total number of available devices were 12 which is also varied for measuring different performance metrics. First task of the application is always executed on the *user* device. The *user* device can execute one or more tasks while the rest of the tasks are offloaded to be executed on the *worker* devices. A device must contain a certain percentage of remaining battery power (ψ) for self-sustainability and a minimum of -80dB

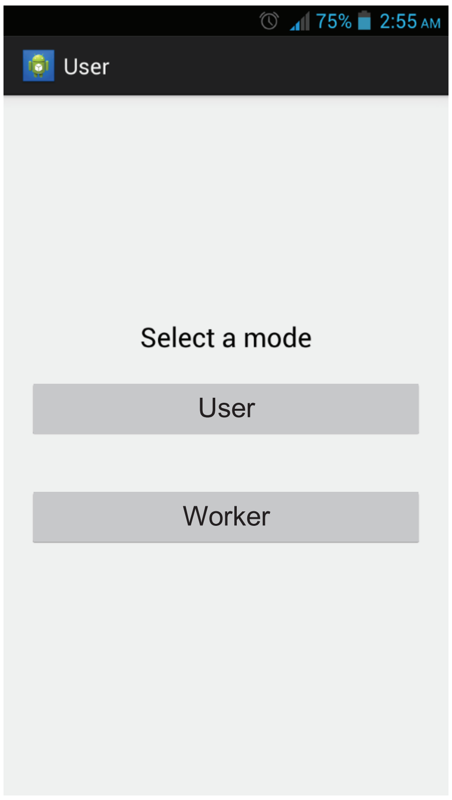


Figure 3.5: User/Worker interface of android application

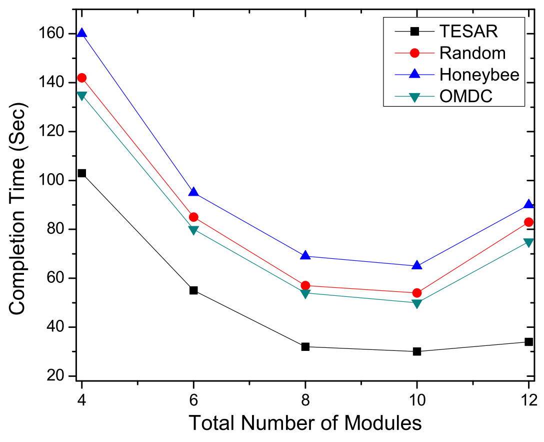
signal strength to be a candidate *worker*. The ψ is a system defined parameter and its value can be tuned following the needs of the computing environment and without loss of generality, we have kept $\psi = 20\%$ in our experiments. The access point that has been used to connect the mobile devices with the cloudlet supports IEEE 802.11b/g/n and can achieve maximum 150Mbps data rate through different channels [90]. The value of α has been chosen to be 0.6 to give emphasis on the execution time. All the experiments have been conducted for 20 times and the obtained results are averaged. The local device takes 235 seconds on an average to execute the application without offloading.

3.4.2 Results and Discussion

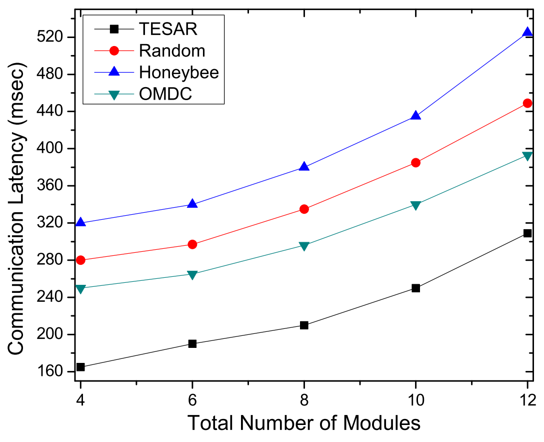
This subsection provides the experimental result and analysis of our proposed *TESAR* system with other benchmark solutions. In most of the cases, random allocation method fails to execute the allocated task as it doesn't consider the capability of the *worker* device and handles failed tasks. The results depict that, the magnitude of transmission time is negligible compared to completion time. The results for random allocation are obtained from the successful completion of application executions only.

3.4.2.1 Impact of number of tasks in an application

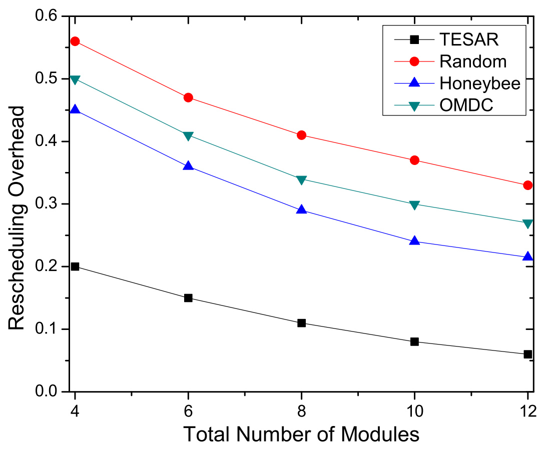
Fig. 3.6 shows the impact of varying the total number of tasks in an application on the performances of the studied systems. Fig. 3.6(a) shows that, initially, the total completion time is decreased significantly with the increasing number of application tasks in all the studied systems. Such behavior is theoretically expected as well since the scope of parallel execution is enhanced with the number of tasks. However, after reaching at a certain level of partitioning (10 tasks in the figure), the completion time starts increasing gradually with the number of tasks. This is due to the fact that, as the number of tasks increases, the assignment of tasks to relatively poor *worker* devices also increases and communication latency among the interdependent tasks is increased with the same rate. In case of communication latency (Fig. 3.6(b)), with the rise of the number of tasks, transmission time and output reception time increases for all the approaches. Fig. 3.6(c) shows the result of rescheduling overhead with the growth of number of tasks. As the number of tasks increases, the task size decreases and hence the rescheduling overhead is also decreased. However, our *TESAR* system outperforms all others with respect to



(a) Completion Time



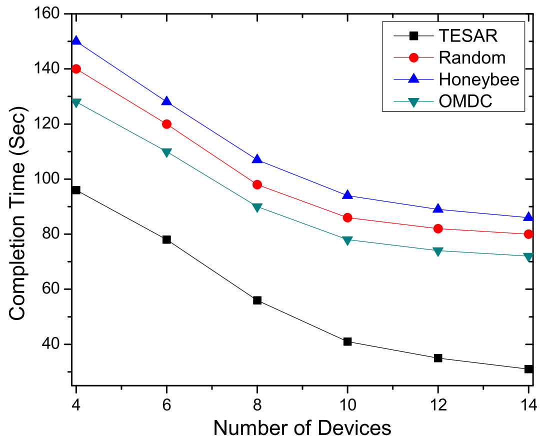
(b) Communication Latency



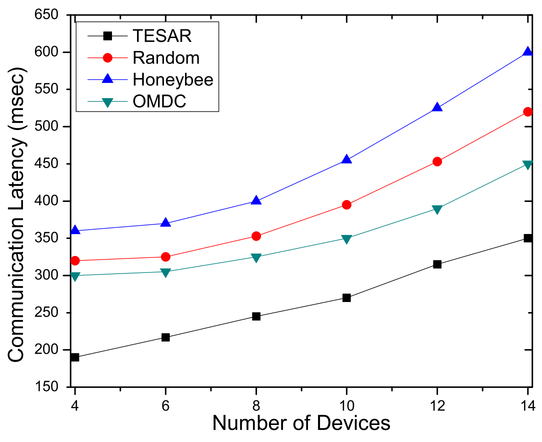
(c) Rescheduling Overhead

Figure 3.6: Impacts of increasing number of tasks in an application

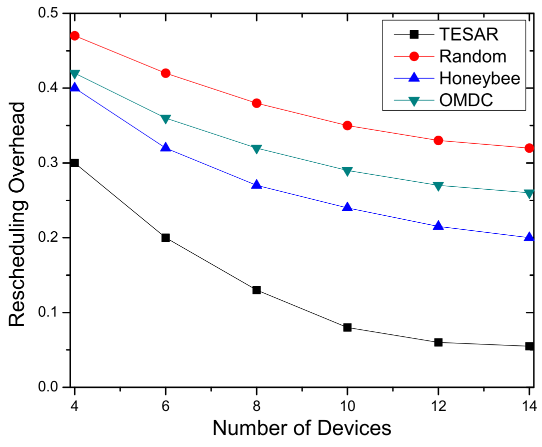
completion time, communication latency and rescheduling overhead since it selects devices with higher signal strength, reliability and associativity period.



(a) Completion Time



(b) Communication Latency

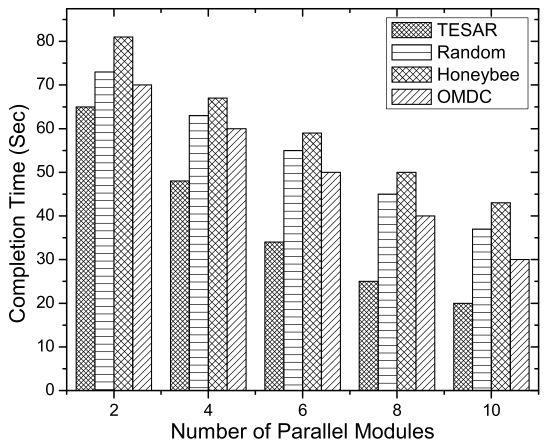


(c) Rescheduling Overhead

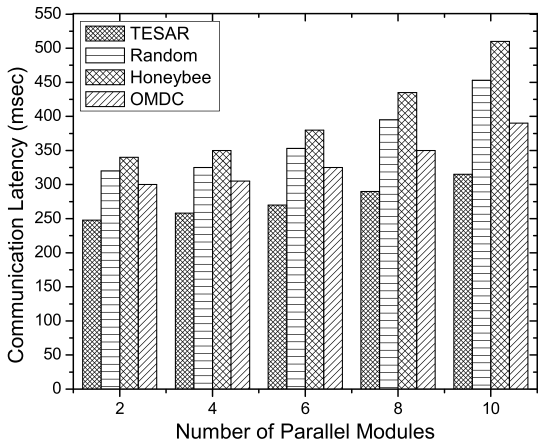
Figure 3.7: Impacts of increasing number of *worker* devices

3.4.2.2 Impact of number of worker devices

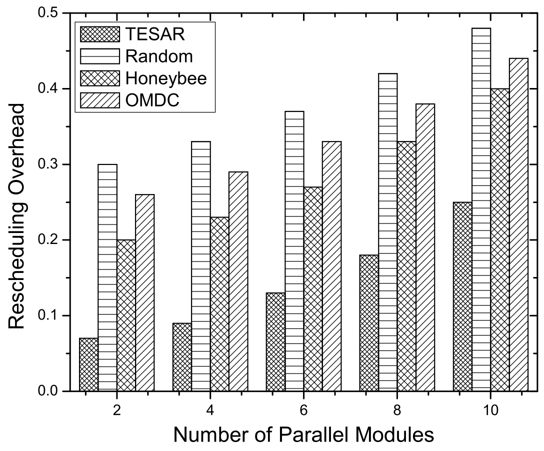
With increasing number of devices, the opportunity of selecting more suitable candidates for code offloading is enhanced, resulting in better performances in completion time as well as rescheduling overhead of tasks, as shown in Fig. 3.7(a) and (c). However, with the increase in the number of devices, communication latency for code offloading is increased gradually (Fig. 3.7(b)). Nevertheless, since *TESAR* method selects devices with high reputation, it can avoid rescheduling of application tasks to a large extent and hence it experiences better performance compared to state-of-the-art offloading algorithms.



(a) Completion Time



(b) Communication Latency



(c) Rescheduling Overhead

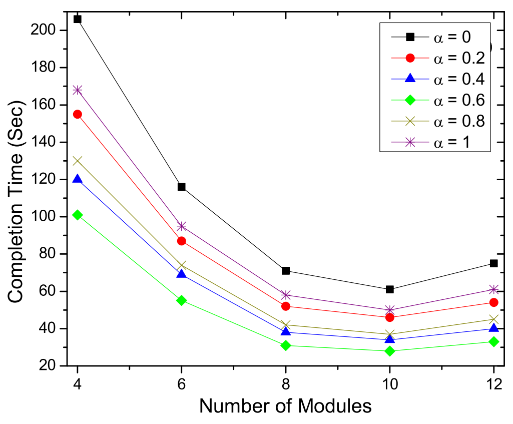
Figure 3.8: Impacts of increasing number of parallel tasks

3.4.2.3 Impact of number of parallel tasks

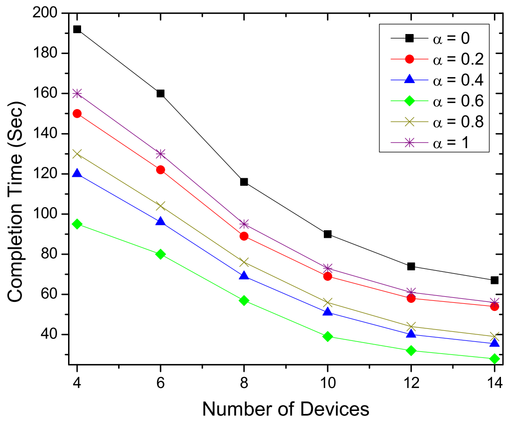
Comparative study among the systems on varying number of parallel tasks is illustrated in Fig. 3.8. Here, the number of total tasks is fixed at 12, amongst which the number of parallel tasks has been increased from 2 to 10. Fig. 3.8 shows that, the completion time of the application decreases gradually with the number of parallel tasks. However, computation latency and task rescheduling overhead rise for higher number of parallel tasks. This is caused by increased communication latency among the higher number of tasks. Again, the likelihood to task rescheduling increases with growing number of parallel executable tasks. However, the proposed *TESAR* system considers partial dependency and it selects optimal devices for code offloading and hence outperforms compared to other offloading algorithms under study.

3.4.2.4 Impact of α value on the performance of ***TESAR***

Fig 3.9 shows performances of the proposed *TESAR* system in terms of completion time of applications with respect to increasing number of devices and tasks for different values of α. The graph reveals the fact that, the proposed *TESAR* system provides the worst completion time for $\alpha = 0$. The completion time is decreased with the gradual increase in α value. This is because, with the increase of α, the algorithm chooses devices having high computational speed and reasonable reliability. It exhibits the optimal behavior when α takes the value of 0.6. However, further increase of α value starts increasing the completion time again. This is due to the fact that, much higher value of α forces the system to choose devices offering reduced reliability, causing a number of tasks experience rescheduling and therefore, completion time of the application is increased.



(a) Increasing number of tasks



(b) Increasing number of devices

Figure 3.9: Impact of α value on the performance of *TESAR*

3.5 Summary

In this chapter, we have focused on strategies for code offloading to surrounding mobile devices instead of distant remote cloud. Offloading decision has been implemented to make a tradeoff between execution speedup and reliability for compute-intensive applications. The proposed *TESAR* system employs cloudlet infrastructure to coordinate apportion of application into tasks and to distribute on different *worker* devices for faster execution. The system outperforms as the best *worker* devices are extracted from the set of candidate *worker*s by considering offered computation speed, reliability, signal strength and available energy. Simultaneous execution of parallel tasks with most suitable *worker* device achieves better result in terms of execution time, communication latency and rescheduling overhead compared to the state-of-the-art works for varying number of tasks and worker devices.

Chapter 4
Incentivizing Workers in MDC

In the previous chapter, we have discussed the formulation of a voluntary approach to execute the tasks of an offloaded application and found that, to exploit the resources of nearby mobile devices, the selection of reliable worker devices is the key to speedup the computation performance. In this chapter, we will discuss on algorithms to incentivize the worker devices for their used resources in executing a task successfully. Here, we also present detailed performance evaluation results.

4.1 Introduction

With huge advances in recent years, mobile devices (e.g., smartphones, smartwatches, and tablets) have become ubiquitous and are rapidly growing as a dominant computing platform for users. These devices, which are equipped with a plethora of embedded sensors (e.g., GPS, camera, audio, proximity, and temperature) to execute various kinds of interactive and real-time applications requiring a large amount of computation. Although the divergence of mobile applications is increasing every day, the execution performance is not yet sufficient due to resource constraints, mainly in terms of limited CPU, memory, and battery capacity [91, 92].

To enhance the computation performance of mobile applications, mobile device cloud (MDC) [6, 15, 16, 37] has been introduced which is an opportunistic computation offloading technology that exploits idle resources of stationary mobile devices. Mobile devices in a large scale stationary location (e.g., theater, shopping mall, stadium, or restaurant) or onboard a vehicle (e.g., bus, train, and airplane) may collaboratively form a cloud service infrastructure. MDC technology has facilitated the running of many compute-intensive mobile applications such as intelligent transportation, natural language translation, augmented reality, and real-time health monitoring [4, 93]. A study shows that mobile devices are kept in idle state approximately 89 % of the time, and during this period they consume not more than 11 % of the available system resources [94]. The idle resources from such a plethora of mobile devices present untapped computing opportunities [61]. The MDC technology not only mitigates scarcity of computation resources in cloudlet-based offloading mechanisms [91, 95, 96] but also resolves communication latency of remote clouds [4, 10, 12].

Typically, an application user (i.e., a buyer device) offloads an application code to the MDC manager, and then it is executed by different worker devices (i.e., sellers) having a sufficient amount of idle resources. The actual benefit of the emerging MDC technology can only be utilized through the effective participation of mobile worker devices in the computation process. For this reason, it is important to ensure the participation of nearby worker devices in such a computation system so as to exploit unused resources efficiently. In state-of-the-art works, authors have focused on designing an MDC framework, where the worker devices participate voluntarily in the task execution process [16, 21, 27, 41, 62, 65, 66, 81]. However, these methods lack to attract a good number of reliable workers in the resource-trading mechanism due to a lack of compensation for the used resources. Moreover, qualified and reputed worker devices became demotivated in sharing resources as their contribution were unrecognized in the allocation of future tasks. Thus, a payment mechanism should be planned to ensure the participation and sharing of resources of eligible worker devices. Moreover, the introduction of a reward mechanism can incentivize the use of mobile device resources, increasing motivation for worker devices.

In the literature, a number of strategies have introduced to provide payments for the used resources of the worker devices. A few researchers addressed a general payment strategy that discretely exploited parameters like execution cost, remaining energy, energy consumption, etc. parameters for the selection and later providing payment of the worker devices [15, 34, 59, 68, 83]. However, these mechanisms unable to fulfill the demand of different user applications due to the selection of untrustworthy workers with poor resource capacity and large communication overhead, resulting in poor execution performance. Few researchers exploited a reverse auction mechanism to provide payment for the used resources of the worker devices in executing the application tasks [22, 58, 60, 69]. These works applied a game-theoretic approach to finding an equilibrium point between the user cost and worker profit. However, these approaches failed to provide a feasible solution in most of the cases as the user budget exceeds the worker device payment due to immediate next bidding price strategy while few approaches completely neglected the budget of a user. A few works also considered a forward auction mechanism to allocate application tasks to different worker devices and offer payment for their used resources [70, 71, 84]. However, these approaches also unable to recognize qualified worker devices due to bidding to selected workers, resulting in a suboptimal result.

In general, all state-of-the-art-works are focused on designing a payment mechanism either to minimize the cost of the user or to maximize the profit of the worker devices. However, the impact of user Quality-of-Experience (QoE) for executing tasks in an MDC environment is yet to be investigated extensively.

The QoE metric quantifies the improvement of the execution time of an offloaded task observed by the user. Focusing only on minimizing execution cost leads to the selection of unreliable and/or poor workers, hampering the user QoE. A buyer device always wants to increase QoE at low-cost and a seller device expects higher payment. This payment acts as compensation for its used resources (e.g., CPU, memory, and energy) and bandwidth charges. Furthermore, an application user (i.e., buyer) might be motivated to pay more only if s/he receives high-quality execution supports from worker devices. Hence, it is a pre-eminent concern to design an incentive mechanism to enhance worker participation, while considering the resource capacity and reputation of worker devices to execute the tasks with the aim of increasing user quality-of-experience (QoE).

In this chapter, we focus on the selection of reputed and resourceful worker devices to execute tasks of an application and incentivize them based on the quality of execution. We consider an MDC system consisting of a cloudlet acting as a cloud broker and a set of participating mobile devices (i.e., user and workers), as shown in Figure 4.1. The user device has an application (with a set of individual tasks) that requires offloading to the cloudlet for execution. After getting an announcement from the cloudlet, a worker device expresses its willingness to execute a certain task/tasks. Following the reverse-auction bidding policy, the cloudlet then determines an optimal mapping of the tasks to be executed on the worker devices so as to maximize user QoE and minimize execution cost. After the successful execution of the assigned tasks, claimed cost with additional incentives according to execu-

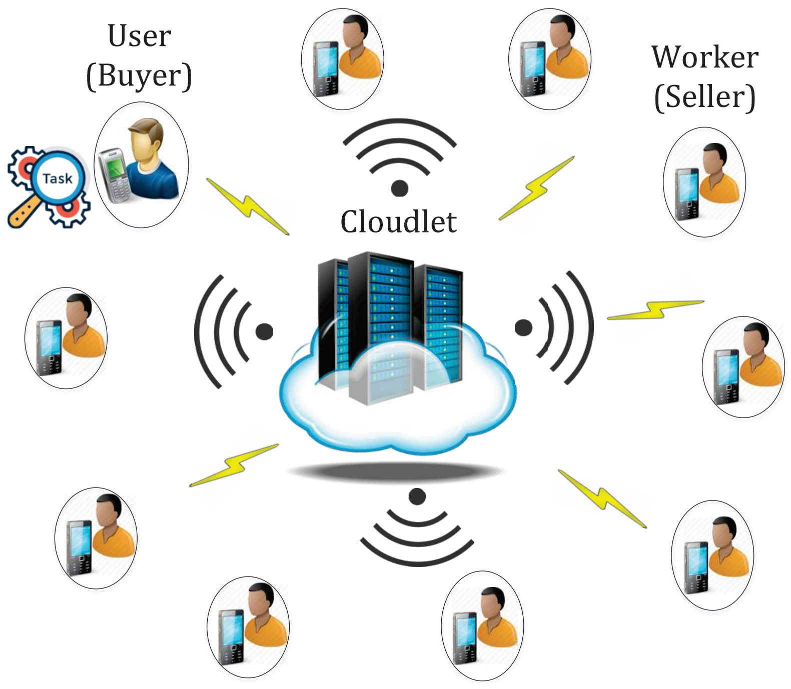


Figure 4.1: Architecture for collaborative computation

tion quality is paid to the worker devices. Contributions listed in the works of this chapter are:

- We design a framework for a QoE-aware incentive mechanism to execute applications by workers in MDC. To the best of our knowledge, this is the first work to improve the user-QoE through incentivizing worker devices in addition to their regular bid payment, according to task execution quality.
- We formulate a multi-objective linear programming (MOLP) function that determines the optimal provisioning of application tasks on high performing worker devices with the aim of increasing user QoE with reduced cost.
- Due to the NP-hardness of MOLP, we develop two greedy task-worker as-

signment algorithms and incentive mechanisms to facilitate resource sharing using reverse-auction theory.

- The novelty of this work lies in paying additional incentives to the workers following their offered qualities of user task execution.
- The reliability and trustworthiness of the workers and the correctness of the incentive mechanisms have been proved theoretically.
- The performances of the proposed incentive mechanisms were evaluated in MATLAB [97], and significant improvements in user QoE and cost reduction were demonstrated.

The remainder of this chapter is organized as follows. Section 4.2 presents an incentive aware computation framework for MDC along with our assumptions. Section 4.3 formulates the worker device selection and incentive disbursement methods and presents a theoretical analysis. Section 4.4 presents the simulation environment and experimental results of the proposed incentive mechanisms with comparative analysis. Finally, the paper is concluded in Section 4.5.

4.2 System Model and Assumptions

This section introduces a novel computation framework for an MDC system, interactions among its functional modules and assumptions made for modeling the task execution. In the remainder of the chapter, without loss of generality, the terms *buyer* and *seller* devices should be considered synonymous to the terms user and worker devices, respectively.

4.2.1 Computation Framework

We consider an MDC system with three different entities: users *(buyers)*, workers *(sellers)* and a cloudlet *(broker)*, which are working in two tiers. Mobile devices that are running applications that require additional resources (CPU, memory, etc.) for code execution, act as *buyers* and devices providing the required computation resources act as *sellers*; both reside at tier one. Interested seller devices bid for different application tasks based on their shareable resources. As presented in Figure 4.2, the allocation and distribution of these tasks and resources are done by a cloudlet, which acts as the central controller for task execution and resides in tier two. The cloudlet also acts as a broker to select the winners from a pool of candidate workers and their payment disbursement. Detailed descriptions of four functional modules of our proposed computation framework are appended below. To enable parallel execution, the dependency estimation component determines the interdependency among the tasks and [62].

Task Profiler (TP): The **TP** module receives an application with an allotted budget ($\mathcal{P}$) and an execution deadline through the *task receiver* component. It then uses the *workload analyzer* component to split the application into a set of atomic tasks $\mathcal{M}$, where each atomic task $m \in \mathcal{M}$ contains $\mathcal{S}_m$ number of instructions for offloaded execution. The *dependency estimation* component is responsible for determining the interdependency among tasks [62] which is used to identify execution order, facilitating parallel execution. The **TP** then hands the tasks over to the *task advertisement* component for advertising the tasks to the worker devices. Each advertised task is a three-parameters tuple denoted by $< m, \mathcal{S}_m, \mathcal{T}_m >$, where, $m \in \mathcal{M}$ is the task ID, $\mathcal{S}_m$ is its number of instructions and $\mathcal{T}_m$ defines the task deadline. Based on the task size and historical transaction prices, it also calculates

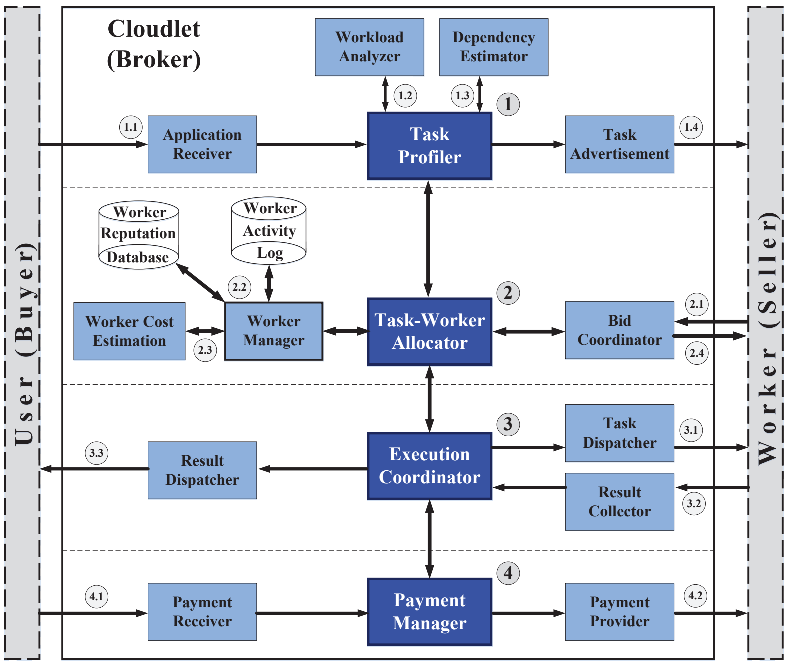


Figure 4.2: Computation framework of the proposed incentive-driven MDC system

maximum and minimum payable amounts (B_m^{max} and B_m^{min}, respectively) for each task to guard against bids from felonious workers [98]. The **TP** shares the lists of advertised tasks information with the **TWA**.

Task-Worker Allocator (TWA): The **TWA** module is the core of the proposed system, and it controls and coordinates functionalities of other modules. It triggers the **TP** to collect user applications and submits advertisements after profiling the application tasks. It collects worker bids ($\mathcal{B}$) through the *bids coordinator* component. The worker devices send their device-specific information, identified by the tuple $< k, \mathcal{B}^k, \mathcal{R}^k, \eta^k, \mathcal{H}^k >$, which contains device identifier (k), workload capacity ($\mathcal{H}^k$), associativity time (η^k), and a set of bids ($\mathcal{B}^k$). Parameter $\mathcal{R}^k$ is used to determine the aggregated resources required to execute a single instruction, whereas the workload capacity $\mathcal{H}^k$ indicates the maximum number of instructions that can be handled by the worker [58, 70]. The associativity time η^k indicates the expected amount of time a worker device will stay in the vicinity. Associativity time of a worker device can be calculated based on its contextual information and current GPS location which has been exploited in [99]. A single worker $k \in \mathcal{K}$ is allowed to bid for multiple tasks $m \in \mathcal{M}$, and each bid $\mathcal{B}_m^k \in \mathcal{B}$ is represented by tuple $< k, m, b_m^k, \mathcal{Q}_m^k >$, which contains execution cost (b_m^k) and execution quality ($\mathcal{Q}_m^k$) promised by the worker $k \in \mathcal{K}$. However, to win a bid a worker device must have to satisfy the reputation and computation resource constraint in addition to bid cost. Moreover, a worker device will be entitled to execute multiple tasks only if it has sufficient computational resources and the tasks are sequential in order of execution.

All the collected bids and device information are then transferred to the **TWA** module to determine the winners among the submitted bids. After collecting all the bids from different workers, the **TWA** matches the advertised list of tasks with the corresponding worker bids to ensure that no task remains unbidden. Notifications of winning bids ($\mathcal{W} \subseteq \mathcal{B}$) are also dispatched through the same interface to selected worker devices ($\mathcal{V} \subseteq \mathcal{K}$). The **TWA** module interacts with the *worker manager* component to determine the dependability of workers to execute tasks successfully and to estimate the required cost. The *worker manager* component evaluates submitted bids to determine the quality and reliability of worker device through historical traces containing execution history and reputation information [100, 101]. After accumulating all information, it runs a worker selection algorithm to determine the winners from the set of candidate worker devices, and then it forwards the task-worker mapping list to the *execution coordinator (EC)* to schedule the tasks in order.

Execution Coordinator (EC): On reception of the task-worker mapping list from the **TWA**, the **EC** calls the *task dispatcher* component to schedule the tasks in order. This is to be noted that due to different computation and communication resources, the worker devices exhibits heterogeneous execution and communication latencies. However, ordered scheduling of the tasks helps to diminish any synchronization latencies due to such issues. After successful execution, results are collected and transmitted back to the user device with a payment ($\mathcal{P}' \leq \mathcal{P}$) disbursement request. Failed executions are reallocated by the **TWA** to the next available bidder. After the execution of all the submitted tasks, reputation of the allocated worker devices are updated according to their execution results and sent to the **TWA** module to store in the *worker manager* database.

Payment Manager (PM): The **PM** receives the agreed amount of payment from the user device through the *payment receiver* component. Upon getting the successful execution notification from the **EC**, the **PM** disburses the individual amount of payment ($\mathcal{P}^v$) to the corresponding winning worker devices $v \in \mathcal{V}$ according to their bids along with incentives, if any, with the help of the *payment provider* component. It also collects a certain percentage of the worker bid cost as the utility ($\mathcal{U}^0$) of the cloudlet, which is acting as a broker, coordinating all these transactions and communication activities on behalf of the user and worker devices.

Figure 4.3 shows the sequence of activities among the user tasks, worker devices, and cloudlet platform for application execution.

The proposed incentive mechanisms trade among the *seller* devices to execute application tasks of a *buyer* with minimum cost and maximum quality. The incentive mechanisms should satisfy the following desirable properties:

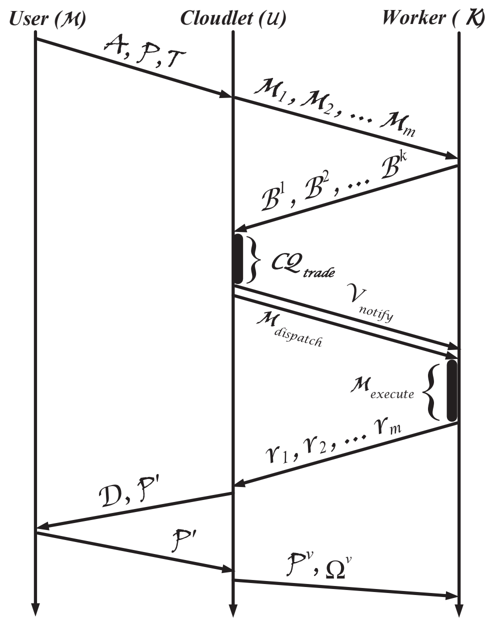

User ($\mathcal{M}$)
Cloudlet ($\mathcal{U}$)
Worker ($\mathcal{K}$)
$\mathcal{A}, \mathcal{P}, \mathcal{T}$
$\mathcal{M}_1, \mathcal{M}_2, \ldots \mathcal{M}_m$
$\mathcal{B}^1, \mathcal{B}^2, \ldots \mathcal{B}^k$
CQ_{trade}
$\mathcal{V}_{notify}$
$\mathcal{M}_{dispatch}$
$\mathcal{M}_{execute}$
$\Upsilon_1, \Upsilon_2, \ldots \Upsilon_m$
$\mathcal{D}, \mathcal{P}'$
$\mathcal{P}'$
$\mathcal{P}^v, \Omega^v$

Figure 4.3: Sequence diagram of proposed incentive mechanism

- **Computational efficiency**: An incentive mechanism is said to be computationally efficient if it can produce an auction decision in polynomial time.

- **Individual rationality**: The incentive mechanism must ensure a positive utility to bid-winning worker devices and the cloudlet to facilitate execution of application tasks, i.e., $\mathcal{U}_m^k > 0, \ \ \forall k \in \mathcal{K}, \ \forall m \in \mathcal{M}; \ \mathcal{U}_m^0 > 0, \ \ \forall m \in \mathcal{M}$, where $\mathcal{U}_m^k$ denotes the utility of worker $k \in \mathcal{K}$ for task $m \in \mathcal{M}$. This basic requirement is mandatory to encourage the participation of worker devices in the system.

- **Truthfulness**: An incentive mechanism is truthful if it can guarantee that only the bidders declaring true costs are eligible to win the auction. No bidder can increase its utility by submitting a bid other than its actual cost.

- **Budget balance**: The incentive mechanism must guarantee that the total amount of payment $\mathcal{P}'$ charged by different worker devices is within the budget allocation $\mathcal{P}$ of a user for a certain application, i.e., $\mathcal{P}' \leq \mathcal{P}$.

4.2.2 Assumptions

Based on the state-of-the-art works, we have made the following assumptions in this work.

We assume the application as a directed rooted tree where the tasks represent vertices, and the linkages correspond to dependencies among them. Execution of the tasks begins from the root, where parallel tasks start their execution simultaneously and dependent tasks start execution after completion of the parent task [92, 102, 62]. Hence, the execution performance of an application mostly relies on the number of dependent and parallel tasks. The overall execution delay is calculated considering both execution times and communication latencies involved.

We assume there will be a sufficient number of worker devices in the system to allocate all the application tasks, and each task will be bid on by at least one worker. The worker devices agree to execute tasks assigned to them, and each device will execute one task at a time [22, 58]. However, it may execute multiple tasks of one application. These worker devices are symmetric, independent, and risk neutral, having no security or privacy violations. A worker device may bid for multiple advertised tasks, where each bid has been generated randomly considering given task size and deadline. When a worker device submits quality information in a bid, it includes communication latency with actual execution time [103].

We assume the system will be running on a trusted platform where all executions will be in a secure environment and all the device-cloudlet interactions will be governed by proper authorization and authentication techniques [104]. We consider a quasi-static behavior for mobility of the user and worker devices to determine the associativity time (η), where the movement of the devices will remain relatively unchanged for a given period of time that is sufficient to execute the allocated task and return the result to the cloudlet [36, 37, 99]. In this thesis, current GPS location and contextual information have been used to determine the associativity time of a worker mobile device [99].

The major notations used in this paper are listed in Table 4.1.

4.3 Proposed Incentive Mechanism

Successful execution of application tasks greatly depends on the selection of high performing and reputed worker devices. The execution time of an application task also significantly varies from one worker to another due to their resource heterogeneities, which offers varied QoE for users. This section first details the design of an optimal selection process of worker devices considering user QoE and task execution cost. Due to the NP-hardness of the optimal solution, we then develop greedy algorithms for task assignments on suitable workers so as either to maximize QoE or minimize execution cost. Finally, this section ends by presenting a QoE-aware incentive payment mechanism and theoretical proofs of its properties.

4.3.1 Optimal Selection of Worker Devices

The competency of our proposed QoE-aware incentive mechanism mostly relies on efficient selection of worker devices so that the overall execution quality is increased and cost is decreased. The efficiency of application task allocations by the

Table 4.1: List of notations for paid task execution

Symbol	**Description**
$\mathcal{M}$	Set of advertised tasks
$\mathcal{K}$	Set of candidate worker devices
$\mathcal{V}$	Set of winning worker devices
$\mathcal{B}$	Set of bids submitted by worker devices
$\mathcal{W}$	Set of winning bids
$\mathcal{S}_m$	Number of instructions in task $m \in \mathcal{M}$
$\mathcal{T}_m$	Execution deadline of task $m \in \mathcal{M}$
$\mathcal{R}_m$	Resource requirement for executing task $m \in \mathcal{M}$
$\mathcal{C}_m^k$	Claimed cost of worker $k \in \mathcal{K}$ for task $m \in \mathcal{M}$
$\mathcal{Q}_m^k$	Offered quality of worker $k \in \mathcal{K}$, $m \in \mathcal{M}$
Ω_m^k	Earned reputation of worker $k \in \mathcal{K}$, $m \in \mathcal{M}$
$\mathcal{P}_m^k$	Payment of worker $k \in \mathcal{K}$ for task $m \in \mathcal{M}$
$\mathcal{U}_m^0$	Utility of cloudlet for serving task $m \in \mathcal{M}$
$\mathcal{U}_m^k$	Utility of worker $k \in \mathcal{K}$ for task $m \in \mathcal{M}$

TWA module on different worker devices also depends on their reputations and resource availabilities. In addition to that, interdependency among the tasks and costs demanded by the worker devices for task execution are also important. Thus, provisioning of application tasks on worker devices is a multi-objective, multi-constrained problem. The next subsections describe methods for measuring user QoE and execution cost metrics, followed by formulation of an optimization framework for the problem.

4.3.1.1 User quality-of-experience (QoE)

As discussed earlier, each task has an associated deadline $\mathcal{T}_m$ determined by the cloudlet [105], within which the output of a task must be available to the cloudlet. Execution quality of an application, as termed as service level agreement (SLA) quality, is defined as the ratio of task execution time on the worker device to that on user device. Therefore, quality for a single task $m \in \mathcal{M}$ being offloaded to device $k \in \mathcal{K}$ can be calculated as

$$\mathcal{Q}_m^k = 1 - (\frac{\mathcal{T}_m^k + \mathcal{L}_m^k}{\mathcal{T}_m}), \quad \mathcal{Q}_m^k \in (0, 1], \tag{4.1}$$

where $\mathcal{T}_m^k$ denotes task execution time, and $\mathcal{L}_m^k$ indicates communication latency between worker device $k \in \mathcal{K}$ and the cloudlet during input and output transmission for task $m \in \mathcal{M}$. The task execution time $\mathcal{T}_m^k$ is calculated by $\frac{\mathcal{S}_m}{\mu^k}$, where μ^k represents the CPU speed of a worker device $k \in \mathcal{K}$. The ratio $\frac{\mathcal{T}_m^k + \mathcal{L}_m^k}{\mathcal{T}_m} < 1$ because execution delay cannot exceed the deadline for any task. Now, combining the quality of all tasks, the QoE observed by the user for an application can be expressed as

$$\mathcal{Q} = \frac{1}{|\mathcal{M}|} \sum_{m=1}^{|\mathcal{M}|} \mathcal{Q}_m^k, \quad \mathcal{Q} \in (0, 1]. \tag{4.2}$$

The target of our QoE-aware incentive mechanism is to increase the value of $\mathcal{Q}$ for a user application.

4.3.1.2 Cost of execution

A worker device participating in task execution incurs a cost due to usage of a certain amount of resources (CPU, memory, bandwidth, etc.). Therefore, a payment for the used resources is necessary to incentivize the worker device, promoting this service model [58]. The cost to execute each task $m \in \mathcal{M}$ is determined by the amount of resources utilized by the task during task execution time. The amount of computation resources (CPU clock speed) required to execute task $m \in \mathcal{M}$ with task size $\mathcal{S}_m$ can be given as

$$\mathcal{R}_m = \mathcal{S}_m \times \mathcal{R}^k, \tag{4.3}$$

where $\mathcal{R}^k$ is the resources required by candidate device $k \in \mathcal{K}$ to execute a single instruction. Then, the total cost is calculated by considering cloudlet utility along with execution cost. Now, considering $\mathcal{C}^k$ to be the cost of utilizing a unit resource in device $k \in \mathcal{K}$, the cost of executing task $m \in \mathcal{M}$ can be determined by

$$\mathcal{C}_m^k = \mathcal{R}_m \times \mathcal{C}^k. \tag{4.4}$$

After successful completion of a task, the corresponding worker devices earn payments with incentives in line their execution qualities. From application task profiling and worker selection to task dissemination, the cloudlet controls and coordinates the whole process of execution in the MDC system. Thus, our worker devices pay a certain proportion of their bid amounts to the cloudlet as a coordinating utility. The utility of the cloudlet for executing task $m \in \mathcal{M}$ by candidate device $k \in \mathcal{K}$ is

$$\mathcal{U}_m^0 = b_m^k \times \lambda, \tag{4.5}$$

where b_m^k is the price of bidding by worker device $k \in \mathcal{K}$ to execute task $m \in \mathcal{M}$ that contains a marginal profit with accumulated costs for used resources and cloudlet payment, and λ is the utility percentage that will be given to the cloudlet for coordinating this task assignment and execution. The value of this utility percentage is a system design parameter, and it may vary from one cloudlet to another over a given period. The total utility of the cloudlet $\mathcal{U}^0$ for providing necessary support to execute all tasks of an application is scaled by

$$\mathcal{U}^0 = \sum_{m=1}^{|\mathcal{M}|} \mathcal{U}_m^0. \ \forall k \in \mathcal{K} \tag{4.6}$$

Therefore, the utility of candidate $k \in \mathcal{K}$ for executing a single task $m \in \mathcal{M}$ is

$$\mathcal{U}_m^k = b_m^k - \mathcal{C}_m^k - \mathcal{U}_m^0, \tag{4.7}$$

where $\mathcal{C}_m^k$ is the actual cost of executing task $m \in \mathcal{M}$ on device $k \in \mathcal{K}$. Accumulating the cost of each task $m \in \mathcal{M}$ in the application, we can get total bidding cost of application execution on the candidate devices, quantified as

$$\mathcal{C}_\mathcal{M} = \sum_{m=1}^{|\mathcal{M}|} \sum_{k=1}^{|\mathcal{K}|} b_m^k \times y_m^k, \tag{4.8}$$

where binary variable $y_m^k \in \{0, 1\}$ takes value 1 if task $m \in \mathcal{M}$ is executed on worker device $k \in \mathcal{K}$ and 0 otherwise. Hence, the normalized bidding cost $\mathcal{C}$ of the user is gained by

$$\mathcal{C} = \frac{\mathcal{C}_\mathcal{M}}{\mathcal{P}}, \quad \mathcal{C} \in [0, 1], \tag{4.9}$$

where $\mathcal{P}$ is user-sanctioned budget for execution of the complete application and is the summation of the maximum allowable bid costs for each task $m \in \mathcal{M}$, i.e., $\mathcal{P} = \sum_{m=1}^{|\mathcal{M}|} B_m^{max}$. While selecting worker devices, the proposed incentive mechanism aims to minimize this normalized execution cost for an application.

4.3.1.3 Optimal objective function

The selection of an optimal set of candidate devices for offloading tasks of an application can now be formulated as

$Maximize:$

$$\mathcal{Z} = \operatorname*{argmax}_{\mathcal{W} \in P(\mathcal{B})} \sum_{\forall w \in \mathcal{W}} \{\beta \times \mathcal{Q} - (1-\beta) \times \mathcal{C}\}, \tag{4.10}$$

subject to:

$$\mathcal{C}_{\mathcal{M}} \leq \mathcal{P} \tag{4.11}$$

$$\eta^k \geq max(\mathcal{T}_n^k) + \mathcal{T}_m^k(1 - \Phi_{n,m}), \quad \forall n \in \Pi_m, \tag{4.12}$$

$$\Omega^k \geq \gamma, \quad \forall k \in \mathcal{K} \tag{4.13}$$

$$\sum_{m=1}^{|\mathcal{M}|} \mathcal{S}_m \times y_m^k \leq \mathcal{H}^k, \quad \forall k \in \mathcal{K}, \quad \forall m \in \mathcal{M} \tag{4.14}$$

$$\mathcal{U}_m^k > 0, \quad \forall k \in \mathcal{K}, \forall m \in \mathcal{M}; \quad \mathcal{U}^0 > 0, \quad \forall m \in \mathcal{M} \tag{4.15}$$

$$\sum_{k \in \mathcal{K}} y_m^k = 1, \quad \forall m \in \mathcal{M}. \tag{4.16}$$

In the above formulation, the aim of the objective function (4.10) is to excel in task QoE while reducing execution cost. It chooses a bid set $\mathcal{W}$ from the power set of bids $P(\mathcal{B})$, which optimizes the said parameters. Here, β is the relative weight parameter, which works as a control knob. Its value can be tuned to obtain different tradeoff levels between application QoE and execution cost required by various types of applications. Setting $\beta = 1$ translates it into a quality maximization problem, and $\beta = 0$ makes it a cost minimization problem, while other values correspond to various quantified levels of tradeoff between the two. The cloudlet determines an appropriate value of β following user demands.

The budget constraint defined in (4.11) means that the total payment of workers must not exceed the user-sanctioned payment. The availability constraint (4.12) specifies the minimum amount of time a selected candidate device must stay in the system. Here, Π_m is the set of parents of a task $m \in \mathcal{M}$ and $\Phi_{n,m}$ is the percentage of dependency of a child task $m \in \mathcal{M}$ on its parent $n \in \Pi_m$ [62]. The reputation constraint (4.13) ensures that to win an auction for any task $m \in \mathcal{M}$, the reputations of all selected candidates must be greater than a certain minimum threshold. The taskload constraint (4.14) refers to the fact that a candidate device's total assigned taskload must be less than or equal to its specified maximum capacity $\mathcal{H}^k$. The utility constraint (4.15) ensures that each device executing a task of an application will earn positive revenue. The cloudlet will also earn positive utility for supporting the execution service for each individual task. Similarly, the atomicity constraint (4.16) confirms that a single task will not be assigned to multiple candidate devices, and each task should be executed only once.

Theorem 1. The proposed worker device selection problem in (4.10) is NP-hard.

Proof: The optimization framework in (4.10) is a MOLP because since it contains two conflicting objectives (i.e., maximizing quality and minimizing cost) with combinatorial and continuous constraints. The worker selection problem can be reduced to a multiprocessor scheduling problem (an NP-complete scheduling problem) [106] by leveraging the constraints and considering that all workers offer equal quality. A multiprocessor scheduling problem has following components,

1. A set $S = \{J_1, J_2, ...J_n\}$ of jobs,
2. A partial order $\prec$ on S,
3. A weight function, W, giving the processing time of each job at different processors,
4. A set of processors, K

The problem aims to minimize the overall processing time of jobs which is given as,

$Minimize:$

$$\sum_{j=1}^{k}\sum_{i=1}^{n} W_i \times x_j \tag{4.17}$$

subject to:

$$t(j) < t(j') : j \prec (j') \tag{4.18}$$

The worker selection problem can be reduced to multiprocessor scheduling problem by leveraging the constraints and considering that all workers can solve time with equal quality. If execution of a task depends fully of its parent task, i.e., task can start its execution after the complete execution of its parent task, the ILP is then takes the form:

$Minimize:$

$$\mathcal{Z} = \quad \arg\min_{k \in \mathcal{K}} \ ((1-\beta) \times \mathcal{C}), \tag{4.19}$$

subject to:

$$\mathcal{T}^k \geq \max(\mathcal{T}_n^k), \quad \forall_n \in \Pi_m, \tag{4.20}$$

where, the constraint ensures that, a child task starts execution after completion of its parent task. As we can easily reduce the proposed ILP to multiprocessor scheduling problem, it can safely be declared that, the proposed ILP is at least as hard as multiprocessor scheduling problem, which is NP-hard. Hence, the proposed MOLP problem is NP-hard and cannot provide a polynomial time solution.

In a practical MDC platform, a typical application containing approximately $10-15$ individual tasks may generate thousands of bids. To find boundary values of

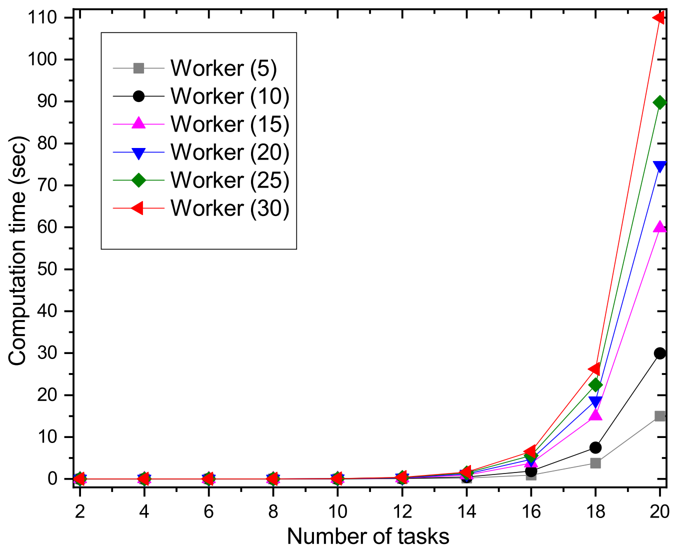


Figure 4.4: Computation time for optimal selection of workers

worker and tasks in a typical MDC environment, we simulate the objective function in NEOS optimization server (2× Intel Xeon e5-2698 @ 2.3GHZ CPU and 92GB RAM) with $\beta = 0.5$. The graphs in Fig. 4.4 show that the computation time for a higher number of tasks and workers exponentially increases due to exploring an enormous number of task-worker assignments. For 20 tasks and 30 workers, the run time exceeds 100 seconds, which might not be tolerable for a decision-making algorithm. To overcome this problem, we develop two greedy solutions for assigning tasks to workers.

4.3.2 Greedy Selection of Worker Devices

To support real-time processing of user applications, this section introduces light-weight greedy worker selection algorithms focusing on either maximizing execution quality or minimizing execution cost. Algorithm 2 selects workers that maximize task execution quality while maintaining total cost within the allocated budget. On the other hand, Algorithm 3 selects workers that demand minimum cost while maintaining the required quality of execution. Detailed descriptions of the algorithms are given in the following subsections.

4.3.2.1 Maximizing task execution quality

Algorithm 2 takes a set of bids $\mathcal{B}$ and a set of tasks $\mathcal{M}$ as input and produces a set of winning bids $\mathcal{W}$ as output. It uses a priority queue of bids $\mathcal{B}'$ in descending order of their execution qualities (line 6). First, the algorithm prepares a candidate bid set after pruning the bids that cannot meet the minimum reputation and fall outside of the bid boundary. These bids are sorted with higher quality values and stored in the priority queue (lines $4-8$). Then, the task of the topmost bid is assigned to the corresponding worker device that fulfills the required resource requirements (lines $10-11$). After successful task assignment, the corresponding task and the bid is removed from the queue and the procedure is repeated until all the tasks assignment are completed (lines $9-17$).

4.3.2.2 Minimizing task execution cost

The algorithm selects worker devices that can satisfy execution constraints, ascertain minimum execution quality, and reduce overall execution costs. Delay tolerant applications, such as text translation, audio video transmission, online forum, and blogging can compromise in terms of execution quality, providing us the opportunity to minimize execution cost.

Algorithm 3 follows similar steps of Algorithm 2, except the priority queue $\mathcal{B}'$ is based on minimum execution cost (line 6). The algorithm selects all bids that

Algorithm 2 Worker selection for maximizing quality

```
INPUT: Set of bids from all worker devices for all tasks, $\mathcal{B} \leftarrow \bigcup_{\forall k \in \mathcal{K}, \forall m \in \mathcal{M}} \mathcal{B}_m^k$
OUTPUT: Set of winning bids, $\mathcal{W}$
 1. Set $\mathcal{W} \leftarrow \phi$, $\mathcal{M}' \leftarrow \mathcal{M}$
 2. Set $\mathcal{R}_u^k \leftarrow 0 \ \ \forall k \in \mathcal{K}$
 3. Set Priority Queue, $\mathcal{B}' \leftarrow \phi$
    //Select the candidate workers
 4. for all Bids $\mathcal{B}_m^k \in \mathcal{B}$ do
 5.   if $((b_m^k > B_m^{min} \ \&\& \ b_m^k < B_m^{max}) \ \&\& \ (\Omega^k \geq \gamma) \ \&\& \ (\mathcal{Q}_m^k > 0))$ then
 6.     $\mathcal{B}'.push(\mathcal{B}_m^k)$ //Insert item following priority on higher values of $\mathcal{Q}_m^k$
 7.   end if
 8. end for
    //Pop the topmost bid and assign the task to the corresponding worker
 9. while $(\mathcal{B}' \neq \phi \ \&\& \ \mathcal{M}' \neq \phi)$ do
10.   $\mathcal{F} \leftarrow \mathcal{B}'.pop()$
11.   if $(\mathcal{R}_m + \mathcal{R}_u^k < \mathcal{H}^k \ \&\& \ \mathcal{F}.k \cap \mathcal{M}' \neq \phi)$ then
12.     $\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{F}$
13.     $\mathcal{R}_u^k \leftarrow \mathcal{R}_m + \mathcal{R}_u^k$
14.     $\mathcal{M}' \leftarrow \mathcal{M}' \setminus k \mid k \in \mathcal{F}$   //Remove the assigned task
15.   end if
16.   $\mathcal{B}' \leftarrow \mathcal{B}' \setminus \mathcal{F}$   //Remove the assigned bid
17. end while
18. return $\mathcal{W}$
```

provide minimum task execution cost and ensures at least minimum quality (lines $4-17$).

Algorithm 3 Worker selection for minimizing cost

```
INPUT: Set of bids from all worker devices for all tasks, $\mathcal{B} \leftarrow \bigcup_{\forall k \in \mathcal{K}, \forall m \in \mathcal{M}} \mathcal{B}_m^k$
OUTPUT: Set of winning bids, $\mathcal{W}$
1. Set $\mathcal{W} \leftarrow \phi, \mathcal{M}' \leftarrow \mathcal{M}$
2. Set $\mathcal{R}_u^k \leftarrow 0 \ \ \forall k \in \mathcal{K}$
3. Set Priority Queue, $\mathcal{B}' \leftarrow \phi$
   //Select the candidate workers
4. for all Bids $\mathcal{B}_m^k \in \mathcal{B}$ do
5.    if $((b_m^k > B_m^{min} \ \&\& \ b_m^k < B_m^{max}) \ \&\& \ (\Omega^k \geq \gamma) \ \&\& \ (\mathcal{Q}_m^k > 0))$ then
6.       $\mathcal{B}'.push(\mathcal{B}_m^k)$ //Insert item following priority on lower values of $b_m^k$
7.    end if
8. end for
   //Pop the topmost bid and assign the task to the corresponding worker
9. while $(\mathcal{B}' \neq \phi \ \&\& \ \mathcal{M}' \neq \phi)$ do
10.   $\mathcal{F} \leftarrow \mathcal{B}'.pop()$
11.   if $(\mathcal{R}_m + \mathcal{R}_u^k < \mathcal{H}^k \ \&\& \ \mathcal{F}.k \cap \mathcal{M}' \neq \phi)$ then
12.      $\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{F}$
13.      $\mathcal{R}_u^k \leftarrow \mathcal{R}_m + \mathcal{R}_u^k$
14.      $\mathcal{M}' \leftarrow \mathcal{M}' \setminus k \mid k \in \mathcal{F}$   //Remove the assigned task
15.   end if
16.   $\mathcal{B}' \leftarrow \mathcal{B}' \setminus \mathcal{F}$   //Remove the assigned bid
17. end while
18. return $\mathcal{W}$
```

What we unfold next is the process of providing additional incentives to the high performing workers following their offered qualities of task executions. Note here that the incentive mechanisms are applicable for workers assigned tasks either by MOLP system or greedy algorithms.

4.3.3 Quality-of-Experience-Aware Incentive Mechanism

Upon successful execution of all tasks of an application, payment is calculated for disbursement to the corresponding bid winners $v \in \mathcal{V}$. A worker device is paid the bid amount (b_m^v) and an additional incentive based on the executed task quality $(\mathcal{Q}'^v_m)$, which is measured after task execution. This incentive might play a vital role in motivating worker devices to submit rational bids and increasing participation of valued workers in the bidding process.

The entitlement of an incentive depends on two parameters. Firstly, the bid cost must be less than the maximum budget for the task (i.e., $b_m^v < B_m^{max}$). Secondly, the provided execution quality of a task must be higher than the committed quality (i.e., $\mathcal{Q}'^v_m > \mathcal{Q}_m^v$). If these two criteria are fulfilled then we calculate the amount of bonus quality provided by the worker device, i.e., $(\frac{\mathcal{Q}'^v_m - \mathcal{Q}_m^v}{\mathcal{Q}_m^v})$. A worker device is given a portion of the unused budget $(B_m^{max} - b_m^v)$ as an incentive for the provided bonus quality. Otherwise, the worker will not be entitled to any incentive amount. Thus, we calculate the incentive amount σ_m^v as

$$\sigma_m^v = \begin{cases} (B_m^{max} - b_m^v) \times \frac{\mathcal{Q}'^v_m - \mathcal{Q}_m^v}{\mathcal{Q}_m^v}, & \text{if } b_m^v < B_m^{max} \\ & \&\& \ \ \mathcal{Q}'^v_m > \mathcal{Q}_m^v. \\ 0, & \text{otherwise.} \end{cases} \tag{4.21}$$

Now, the cloudlet is responsible for computing the required payment to the worker devices. A detailed description of the payment strategy for bid-winning worker devices is presented in Algorithm 4.

Algorithm 4 Incentive payment for winner devices

INPUT: *Set of winning bids*, $\mathcal{W}$, from Algo 1 or Algo 2.

OUTPUT: *Payment* $\mathcal{P}^v$ *for winner device,* $v \in \mathcal{V}$*, Total amount of payment,* $\mathcal{P}'$ *for the user application.*

```
1.  Set $\mathcal{P}^v \leftarrow 0, \mathcal{P}' \leftarrow 0$
2.  $\mathcal{V} = \{v \mid v \text{ is a winning device listed in } \mathcal{W}\}$
3.  for all winning devices $v \in \mathcal{V}$ do
4.      for all tasks $m \in \mathcal{M}$ do
5.          Calculate incentive amount, $\sigma_m^v$ using Eqn. (4.21)
6.          $\mathcal{P}_m^v \leftarrow b_m^v + \sigma_m^v$   //Total payment for a task $m$
7.          $\mathcal{P}^v \leftarrow \mathcal{P}^v + \mathcal{P}_m^v$   //Total payment for a worker $v$
8.          $\mathcal{P}' \leftarrow \mathcal{P}' + \mathcal{P}_m^v$   //Total payment for the application
9.      end for
10. end for
11. return $\mathcal{P}^v, \mathcal{P}'$
```

After completion of all tasks, the cloudlet picks a worker device and calculates incentives of each task executed by it using (4.21), total payment for the task ($\mathcal{P}_m^v$), and payment for all the executed tasks ($\mathcal{P}^v$). These steps are repeated for all the worker devices (lines $3-7$). Finally, the cloudlet calculates the total actual payment ($\mathcal{P}'$) of the user and disburses payments to individual workers upon reception of actual payment from the user (line 8). We call the incentive payment mechanism *IMaxQ* when the workers are selected to maximize task execution quality (in Algorithm 2) and *IMinC* when the workers are selected to minimize task execution cost (in Algorithm 3).

4.3.4 Updating Reputations of Worker Devices

Reputations of worker devices are updated by the ***TWA*** module based on the execution results of the tasks; this is important for selecting winning devices in future task assignments. To encourage truthful bidding, successful execution with the offered quality provides a positive reputation, whereas failure in maintaining the offered quality or deadline results in a penalty to protect against dishonest activity of unqualified workers.

To calculate the reputation of a winning device $v \in \mathcal{V}$, the cloudlet first calculates the quality enhancement indicator $\mathbb{E}_m^v$ of the executed task based on qualities $\mathcal{Q}_m^v$ and $\mathcal{Q'}_m^v$ offered in SLA and provided by the worker, respectively.

$$\mathbb{E}_m^v = \begin{cases} 1, & \text{if } \mathcal{Q'}_m^v \geq \mathcal{Q}_m^v \\ (1 - \frac{\mathcal{Q}_m^v - \mathcal{Q'}_m^v}{\mathcal{Q}_m^v}), & \text{if } 0 < \mathcal{Q'}_m^v < \mathcal{Q}_m^v \\ -1, & \text{unsuccessful.} \end{cases} \tag{4.22}$$

The task quality enhancement indicator ($\mathbb{E}_m^v$) is used to calculate the reputation/penalty value gained for executing the current task $m \in \mathcal{M}$ as

$$\Omega_m^v = \alpha \times \mathbb{E}_m^v \times (-1)^{x_m^v} \times (-1)^{y_m^v}, \tag{4.23}$$

where α is a weight parameter used to put emphasis on the currently availed reputation/penalty. In this work, the value of α was set to 0.1, placing only a small significance on the reputation achieved for executing the current task. The binary variable $x_m^v \in \{1, 0\}$ is used to represent service level agreement (SLA) retention status. x_m^v is set to 1 if worker device $v \in \mathcal{V}$ successfully executes task $m \in \mathcal{M}$, maintaining $\mathcal{Q'}_m^v \geq \mathcal{Q}_m^v$; otherwise, it is set to 0. Similarly, binary variable $y_m^v \in \{1, 0\}$ takes value 1 if task $m \in \mathcal{M}$ successfully executed on the worker device $v \in \mathcal{V}$ and 0 otherwise. Finally, the reputation of a worker device is updated for future task assignments considering previous reputation as

$$\Omega^v = max(0, min(\Omega'^v + \Omega^v_m, 1)). \tag{4.24}$$

Equation (4.24) bounds the reputation for a worker device $v \in \mathcal{V}$ between 0 and 1. In this work, the initial reputation of a worker device was considered as 1.0 to encourage new workers to participate in the MDC system. Note that (4.23) and (4.24) jointly ensure that on-time successful execution increases the reputation of a worker and that delayed or failed execution causes a penalty for the device. In this way, the dynamic reputation update of a worker facilitates our proposed incentive mechanisms to select high performing workers for task execution, increasing user QoE as well as incentives for workers.

4.3.5 An Illustrative Example

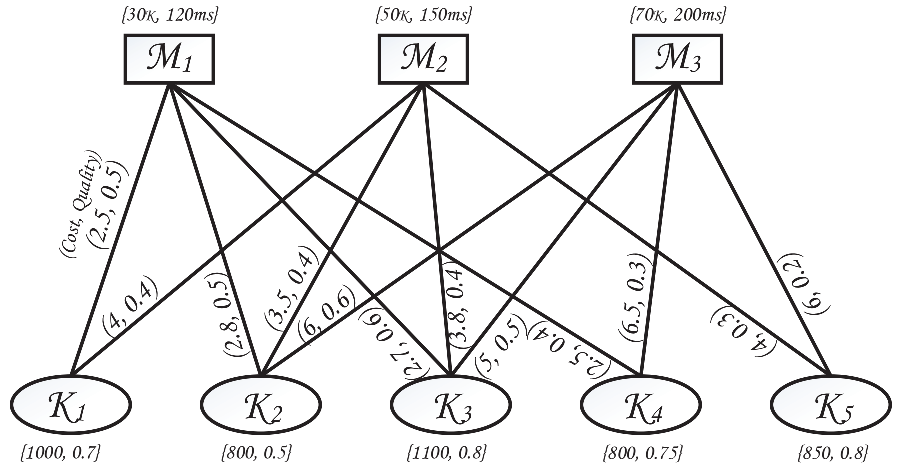


Figure 4.5: An example scenario for task assignment

This section presents an illustrative example on operation processes of the proposed *IMaxQ*, *IMinC* and *Optimal* solutions. Consider a scenario depicted in Fig. 4.5, where the cloudlet divides an application into three tasks: $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ with allowable maximum execution costs 3, 5, and 7 units, respectively. We also assumed that five worker devices $\mathcal{K}_1 - \mathcal{K}_5$ available in the system bid for the designated tasks shown by edges (in Fig. 4.5, labeled with bid cost and committed quality). The instruction sizes and deadlines of the tasks are labeled at the top and available resource capacities and reputations of the worker devices are labeled at the bottom.

The task assignment to different workers, their incentives, and user cost savings have been shown in following Table 4.2. It is to be noted here that the worker $\mathcal{K}_2$ could not win any bid due to its poor reputation. Furthermore, as expected theoretically, the proposed *IMinC* has brought out the maximum cost savings for the user and *IMaxQ* has offered the lowest while the *Optimal* solution (with $\beta = 0.5$) has worked out with competitive savings. We also notice that, in all algorithms, tasks were executed satisfying the expected QoE within the allocated budget. Finally, the amount of incentive (σ_m^v) awarded to a worker is directly proportional to the quality ($\mathcal{Q}'^v_m$) it offers in executing a user task.

4.3.6 Theoretical Proofs of Desirable Properties

In this section, we provide theoretical proofs of the desirable properties of the proposed incentive mechanisms, including computational efficiency, individual rationality, truthfulness, and budget balance.

Lemma 1. The proposed incentive mechanisms run in polynomial time, meaning they are computationally efficient.

Table 4.2: Bid winners, their incentives, and user cost savings

	$\mathcal{K}$	$\mathcal{M}$	b^v_m	$\mathcal{Q}^v_m$	$\mathcal{Q'}^v_m$	σ^v_m	$Savings(\%)$
IMaxQ	3	$\mathcal{M}_1$	2.7	0.6	0.7	0.05	
	1	$\mathcal{M}_2$	4.0	0.4	0.5	0.25	10.0%
	4	$\mathcal{M}_3$	6.5	0.3	0.3	0	
IMinC	1	$\mathcal{M}_1$	2.5	0.5	0.6	0.1	
	3	$\mathcal{M}_2$	3.8	0.4	0.5	0.3	12.0%
	5	$\mathcal{M}_3$	6.0	0.2	0.3	0.5	
Optimal	4	$\mathcal{M}_1$	2.5	0.4	0.4	0	
	1	$\mathcal{M}_2$	4.0	0.4	0.5	0.25	11.67%
	5	$\mathcal{M}_3$	6.0	0.2	0.3	0.5	

Proof: In Algorithm 1, lines $4-8$ run for all bids to check the eligibility of bids and to prune bids that do not maintain the required constraints, incurring a runtime complexity of $O(|\mathcal{B}|)$. Inside the loop, candidate bids are pushed in a priority queue (line 6) that has a runtime complexity of $O(|\mathcal{B}|log|\mathcal{B}|)$, totaling $O(|\mathcal{B}| \times |\mathcal{B}|log|\mathcal{B}|)$. Lines $9-17$ add complexity of $O(|\mathcal{B}|)$, and the remaining statements are executed in constant time; hence, the overall runtime complexity of Algorithm 1 is $O(|\mathcal{M}| \times |\mathcal{K}| \times |\mathcal{M}| \times |\mathcal{K}|log|\mathcal{M}| \times |\mathcal{K}| + |\mathcal{M}| \times |\mathcal{K}|) \approx O(|\mathcal{M}|^2 \times |\mathcal{K}|^2 log|\mathcal{M}| \times |\mathcal{K}|)$, where $\mathcal{M}$ is the set of application tasks and $\mathcal{K}$ is the set of available worker devices. The runtime complexity of Algorithm 2 is the same as that of Algorithm 1. Similarly, Algorithm 3 has a runtime complexity of $O(|\mathcal{M}| \times |\mathcal{K}|)$.

Thus, the complexities of the algorithms are bounded by $|\mathcal{M}|$ and $|\mathcal{K}|$, so they are computable within polynomial time.

Lemma 2. The proposed incentive mechanisms are individually rational to the cloudlet and worker devices.

Proof: Consider the two following scenarios:

1. **If a worker device $k \in \mathcal{K}$ wins no bid,** it will have no resulting cost, $\mathcal{C}'^{k} = 0$, payment $\mathcal{P}^k = 0$, and $\mathcal{U}^k = 0$. In this case, the utility of the cloudlet $\mathcal{U}^0 = 0$.

2. **If a worker device $k \in \mathcal{K}$ wins a bid**, its utility is calculated as $\mathcal{U}_m^k = \mathcal{P}_m^k - \mathcal{C}_m^k - \mathcal{U}_m^0$, where payment is determined as $\mathcal{P}_m^k = b_m^k + \sigma$. According to (4.5), cloudlet utility ($\mathcal{U}_m^0$) will be nonzero for a successful task execution by a worker device. Moreover, for a successful execution by a worker device, payment will be at least equal to the bid price. As worker devices bid for a task $m \in \mathcal{M}$ including cloudlet payment ($\mathcal{U}_m^0$) and its resource cost $\mathcal{C}_m^k$, bid price must be higher than $\mathcal{U}_m^0 + \mathcal{C}_m^k$, which has been ensured through constraint (4.15). Therefore, in conclusion, it can be clearly seen that the utility of a worker device and cloudlet will be nonzero.

Lemma 3. The proposed incentive mechanisms are truthful.

Proof: Let $\mathcal{U}_m^k$ and b_m^k denote the utility and bid price, respectively, of a worker device $k \in \mathcal{K}$ if it bids with actual cost $\mathcal{C}_m^k$, and let $\bar{\mathcal{U}}_m^k$ and $\bar{b}_m^k$ denote the illegitimate utility and bid price, respectively, caused by bad intension of a worker, where $b_m^k \neq \bar{b}_m^k$. To prove this lemma, we must show that only a truthful bid price can provide the maximum utility of a worker device, i.e., $\mathcal{U}_m^k \geq \bar{\mathcal{U}}_m^k$, $\forall b_m^k \neq \bar{b}_m^k$. In this case, we first consider Algorithm 3 (*IMinC*).

Assume that $\bar{b}_m^k > b_m^k$, a win of $\bar{b}_m^k$ means that the worker wins when it bids $\bar{b}_m^k$, and a loss of $\bar{b}_m^k$ means that the worker loses when it bids $\bar{b}_m^k$. Algorithm 3 confirms that a worker with minimum bid cost wins the bid. Hence, $\bar{b}_m^k$ loses the bid in the presence of another bid that equals b_m^k, and thus the utility $\bar{\mathcal{U}}_m^k = 0$ and $\mathcal{U}_m^k > \bar{\mathcal{U}}_m^k$.

If $\bar{b}_m^k = b_m^k$, then a win of $\bar{b}_m^k$ means that b_m^k also wins. In this case, $\mathcal{U}_m^k = \bar{\mathcal{U}}_m^k$ signifies that the actual bid is made by a legitimate worker.

Lastly, if $\bar{b}_m^k < b_m^k$, then $\bar{b}_m^k$ wins the bid according to the worker selection criteria. However, in this case, the expected utility constraint $\bar{\mathcal{U}}_m^k < \mathcal{U}_m^k$ is preserved.

Similarly, for Algorithm 2 (*IMaxQ*), workers offering higher quality may submit overpriced bids. In this case, maximum bid cost (B_m^{max}) for a task $m \in \mathcal{M}$ is used to guard against such dishonest activity and reject such bids during the candidate worker set generation phase, which results in utility $\mathcal{U}_m^k = 0$ for worker $k \in \mathcal{K}$. Moreover, the incentive (σ) for qualified execution induces extra profit to deserving workers. For this reason, workers are motivated in bidding with actual cost to increase the opportunities for a winning bid.

It is to be noted here that a worker might win a task by misreporting both the quality and bid but it would be penalized with a negative reputation due to failure in achieving the committed quality.

Lemma 4. The proposed incentive mechanisms are budget balanced.

Proof: The total amount of payment for a user to execute all tasks is calculated as $\mathcal{P}' = \sum_{m=1}^{|\mathcal{M}|} \mathcal{P}_m^k$, including their incentive (lines $4-7$ in Algorithm 4). If all payments of tasks are equal to their maximum allowable bid prices (including incentive σ, if any), then the total payment will be $\mathcal{P}' = \mathcal{C}_{\mathcal{M}} = \mathcal{P}$ (according to (4.8)); otherwise, the payment will be $\mathcal{C}_{\mathcal{M}} \leq \mathcal{P}' \leq \mathcal{P}$. This means that the total cost of an application execution will always be less than or equal to the budget allocated by the user, i.e., $\mathcal{P}' \leq \mathcal{P}$. Hence, the proposed incentive mechanisms maintain the budget balance property.

4.4 Performance Evaluation

The performances of the proposed ***IMaxQ*** and ***IMinC*** algorithms were compared with two state-of-the-art works (***Min-Cost*** [58] and ***First-Fit*** [107]) through numerical simulations in MATLAB [97]. The ***Min-Cost*** algorithm employed a least cost per unit resource mechanism to select the winning bids. The workers that offer a minimum bid are selected for the execution of tasks and the corresponding worker devices were paid with the immediate next bid amount of the task. On the other hand, ***First-Fit*** is a baseline method extracted from [107] that greedily allocates tasks on worker devices with sufficient resources without considering execution quality, cost, or device reputation.

4.4.1 Simulation Setup

We assumed that a number of user devices (buyers) issue application code execution requests to a nearby cloudlet on which there are $10 - 100$ connected worker devices *(seller)*, randomly distributed in a $50 \times 50m^2$ area. A user application ranging the size from $100K - 500K$ instructions is supposed to execute in an MDC environment. Since the application will be distributed to a number of worker devices, it is split into a random size of smaller tasks containing instructions from $5000 - 125000$. Now, to execute these instructions, worker devices are elected by the greedy algorithms in the system. We consider the capacity of a worker is randomly chosen from the range of $10,000 - 1,000,000$ units. For simplicity, we assumed 1 unit of resources is required to execute each instruction, where 1 unit of cost was estimated for every 10,000 units of resources. The maximum amount of bid (B_m^{max}) for a task has been calculated based on the task size and the cost per unit

Table 4.3: Values of simulation parameters

Parameter	**Description**
Number of tasks	$4 - 24$
Number of workers	$10 - 100$
Application size	$100 - 500K$ instructions
Task size	$5 - 125K$ instructions
Cost per unit resource	10^{-4} units
Total budget	$10 - 50$ units
Worker available resource	$10000 - 1000000$ units
Arrival rate of tasks	$1 - 3$ tasks/second (poisson)
Arrival rate of workers	$2 - 5$ workers/second (poisson)
Task deadline	$500 - 1500$ ms
Reputation threshold (γ)	0.6
Cloudlet utility (λ)	20 % of worker bid cost
Simulation area	50×50 m^2
Simulation time	500 sec

of resources which has been varied according to the task size. The value of (B_m^{min}) has been set to 0 in all experiments. To generate the simulation data, we have implemented an experimental testbed based on our preliminary works [61, 62]. In the experiments, the reputation threshold of a candidate worker was set to 0.6. The percentage of cloudlet utility λ, has been set to 20 %. All simulation experiments were conducted on a PC with an Intel Core i5 2.2 GHz processor and 8 GB memory running Windows 8.1. This is to note here that both the greedy algorithms run in polynomial time and the runtime of both algorithms is few milliseconds which has been ignored in the calculation of task execution quality in the simulation results.

To compare with the state-of-the-art works, at first, we estimate the execution time for the whole application in the user device. Then we compute improvement of QoE, payment savings, and user satisfaction based on execution time, bid amount and number of successful execution (without resubmission), respectively. Details of measuring each performance metric are explained in section 4.4.2. This is to note that each simulation experiment was run for $500s$, and the results collected from 50 runs with different random seed values were averaged to plot each data point with corresponding confidence interval in the graphs. A summary of the values and ranges of different simulation parameters is given in Table 4.3.

4.4.2 Performance Metrics

We focused on the following performance metrics to evaluate the proposed methods and to compare them with other state-of-the-art methods.

- *Improvement of user QoE:* The user Quality-of-Experience (QoE) metric measures the task quality improvement of an offloaded task observed by the user. This gives the average percentage improvement of execution time in MDC compared to that of the user device. To compute it, at first, we estimate the execution time of the whole application in the user device. Then, we distribute the application among the selected worker devices based on the proposed greedy allocation algorithms. After getting the execution results, we compute execution time (including communication latency) for corresponding worker devices and the improvement of user QoE using Eq. 4.2.

- *User payment savings:* The user payment savings parameter reflects the surplus amount of the budget after the completion of payment and incentive of the worker devices. This is defined as the proportion of unused payment compared to the sanctioned budget of a user for a certain application execution; it is expressed as a percentage, i.e., $(1 - \frac{\mathcal{P}'}{\mathcal{P}}) \times 100$ %.

To compute it, at first, we estimate the maximum budget (including cloudlet cost, bandwidth cost, resource compensation, etc.) for the whole user application to be executed in the MDC environment. Based on the bid amount in auction and execution time, the cloudlet disburses payments to the winner worker devices including incentives. It is to note that the eligibility of incentives is determined by Eq. 4.21. Finally, we compute the payment savings with respect to estimated budget and total expenditure including incentives and bid cost.

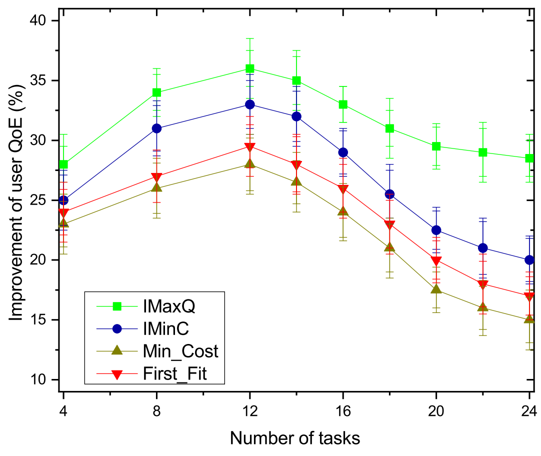
- *User satisfaction:* This is the percentage of successfully executed tasks (without resubmission) out of all offloaded tasks.

4.4.3 Results

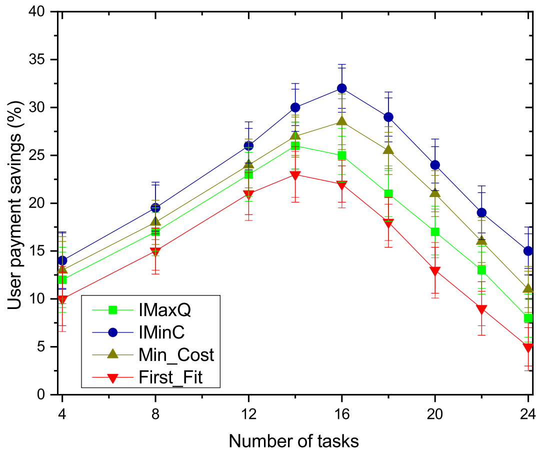
This subsection provides the experimental results and discussion on the comparative performances of the studied systems.

4.4.3.1 Impact of varying number of tasks

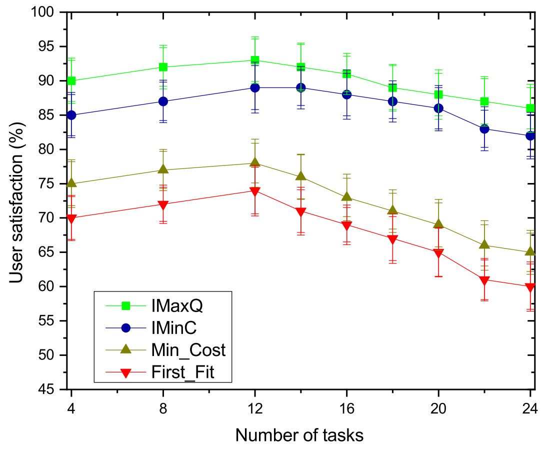
In this experiment, we fix the number of worker devices to $|\mathcal{K}| = 50$ and application size to 200K, and we vary the number of tasks from $4 - 24$. The number of instructions in each task may vary from other tasks and due to fixed size application,



(a) Improvement of user QoE



(b) User payment savings



(c) User satisfaction

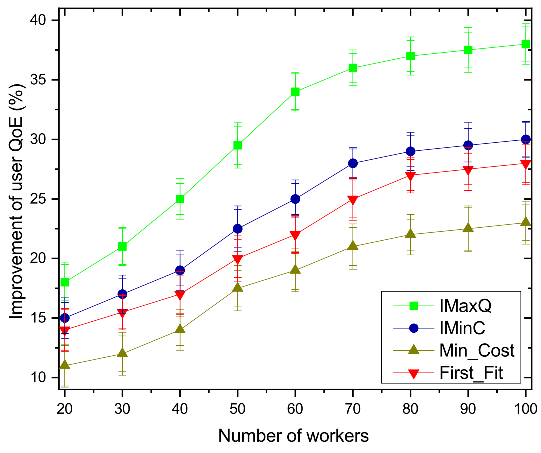
Figure 4.6: Impact of varying number of tasks

the number of instructions of an individual task is reduced with increasing number of tasks.

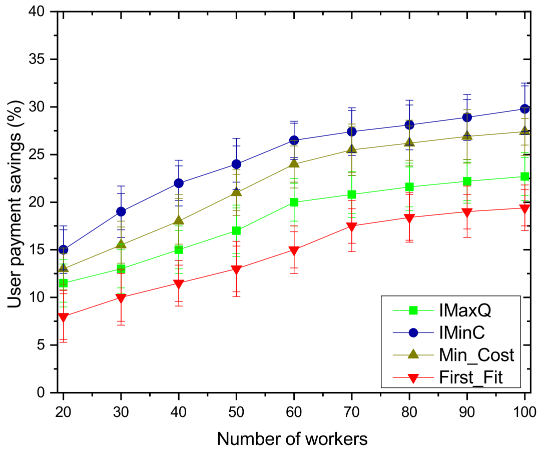
Fig. 4.6(a) shows the improvement of user QoE achieved by the proposed incentive algorithms compared to existing approaches. We observe that the user QoE increases with the number of tasks, and it reaches its peak point when the task population is close to 12. This trend is reasonable since a few capable workers are able to bid and win the tasks at the beginning due to relatively large instruction size of the tasks. When size of an individual task is decreased, the number of capable worker devices for executing the task is increased that facilitated more parallel execution and enabled their executions on highly qualified workers till the saturation point. After that, admission of an additional number of tasks in the system forces it to allocate mid-level or even poor capacity and/or low-quality workers to allocate for the execution of tasks, causing degradation of user-QOE. The graphs also depict that the proposed IMaxQ offers the highest quality (35 %) as it is expected theoretically. Since *IMinC* and *Min-Cost* prefer workers with low cost rather than high quality and thus their performances are significantly less compared to *IMaxQ*. Similarly, due to the random selection of workers, *First-Fit* provides the worst user QoE among all.

On the other hand, in Figure 4.6(b), *IMinC* offers the highest user payment savings with an increasing number of tasks compared to all other incentive algorithms. In the beginning, a fewer number of tasks containing a large number of instructions were bid by a few capable workers at a high cost. Due to this monopolistic competition, such execution leads to small savings for the user. As the number of tasks increases, the amounts of savings also rises in all algorithms. This is due to the fact that more low-cost offering workers were able to compete that assisted better worker selection and execution quality with relatively lower bid cost, resulting higher cost savings for the user. Nevertheless, further increase of tasks ($|M| >= 16$) causes a sharp fall in savings since the system is bound to select workers with high bid costs to accommodate the increasing number of tasks. By comparison, *IMinC* provides the highest payment savings (35 %) and *Min-Cost's* user payment savings percentage is somewhat lower than that of *IMinC* because it pays worker devices with the immediate next bid winner's cost. *IMaxQ* cannot provide low cost execution due to the selection of higher quality workers. Due to the random selection of workers by *First-Fit*, it provides the least payment savings.

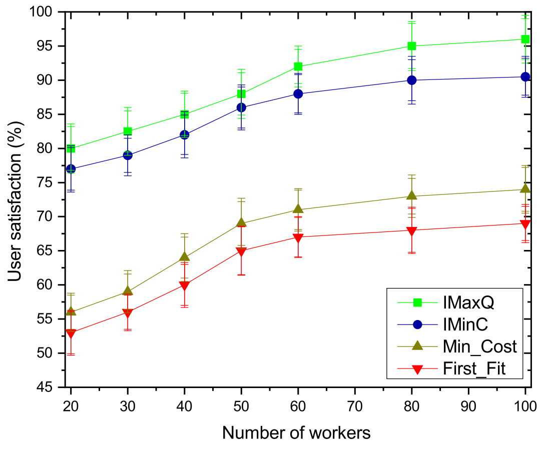
We also measured the satisfaction level of application users offered by the studied systems through on-time execution of tasks, as shown in Fig. 4.6(c). We observed that the user satisfaction rates for all studied algorithms increased with the number of tasks until peaking at approximately $|M| = 12$, and then they started to decrease slowly. Initially, the size of each task was relatively bigger due to a small number of tasks in an application. As a result, a few number of resource-rich workers were able to win the bid and provide successful execution. Dividing the application into more number of tasks instigated more quality workers with a better reputation get a chance to win bids, resulting in higher successful execution. Further increasing the number of tasks implies that each task contains a fewer number of instructions, demanding allocation to more workers of the system. This consequence diminishes user satisfaction due to allocations of tasks to relatively poor workers. Hence, user satisfaction is increased slightly at the beginning and is reduced gradually after reaching a pick point. In comparison, *IMaxQ* and *IMinC* provide better user satisfaction than *Min-Cost* and *First-Fit*, and *IMaxQ* provides the best. This is due to the consideration of worker reputation, which allocates a task to reliable devices in addition to available computation resources. Moreover, lack of consideration of task dependency incurs unnecessary waiting time for a task (dependent on a different one for data input) and hampers user satisfaction in *Min-Cost* and *First-Fit*.



(a) Improvement of user QoE



(b) User payment savings



(c) User satisfaction

Figure 4.7: Impact of varying number of worker devices

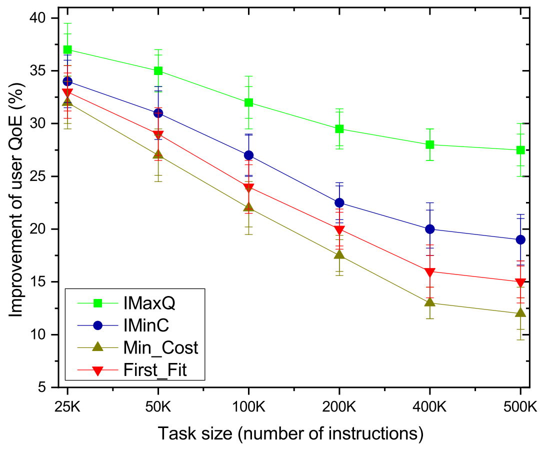
4.4.3.2 Impact of varying number of worker devices

Availability of worker devices creates an opportunity to select more appropriate resources for executing a task, resulting in better system performance. To investigate the impact of worker devices, we fix the number of tasks to $|\mathcal{M}| = 20$ and application size to 200K and vary the number of worker devices from $20 - 100$.

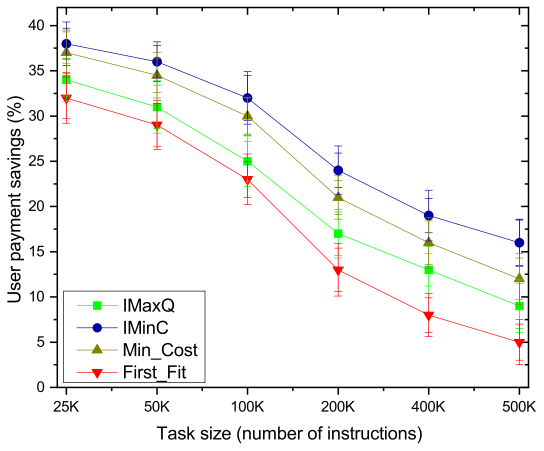
Figure 4.7(a) shows that increasing the number of worker devices drives the average user QoE to rise sharply, and then it slows until it reaches saturation. This is due to the fact that with a fixed set of tasks ($|\mathcal{M}| = 20$), increasing worker devices create an opportunity to allocate resources from more qualified workers, resulting in better user QoE. However, increasing worker devices cannot remarkably improve the user QoE when the system contains many qualified workers ($|\mathcal{K}| > 60$) above the user demand. Our proposed *IMaxQ* and *IMinC* algorithms show better performances compared than the other. Task allocation to reputed workers with rich computing resources provides the finest result for *IMaxQ*, while *IMinC* sacrifices quality slightly to minimize execution cost.

A completely opposite phenomenon is illustrated in Figure 4.7(b), wherein the selection of worker devices with minimum bid cost provides superior performance for *IMinC* and *Min-Cost* compared to *IMaxQ* and *First-Fit*, while *IMinC* provides the maximum savings for a user. This figure also illustrates that user payment savings gradually increase due to the availability of a large number of worker devices offering relatively lower execution cost. However, availability of too many qualified workers ($|\mathcal{K}| > 60$) cannot improve the user savings further as the system reaches saturation.

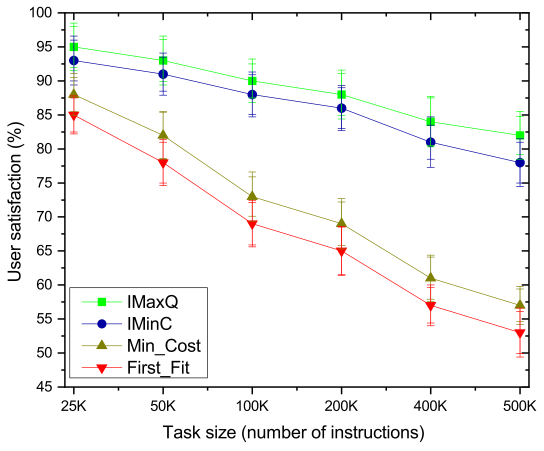
Similarly, increasing worker devices creates more opportunity to improve successful task execution rate, which facilitates higher user satisfaction, as shown in Figure 4.7(c). From this figure, it is also evident that our proposed *IMaxQ* and *IMinC* algorithms maintain significantly superior results than the other algorithms due to the consideration of task dependency, worker reputation, and available computing resources.



(a) Improvement of user QoE



(b) User payment savings



(c) User satisfaction

Figure 4.8: Impact of varying task sizes

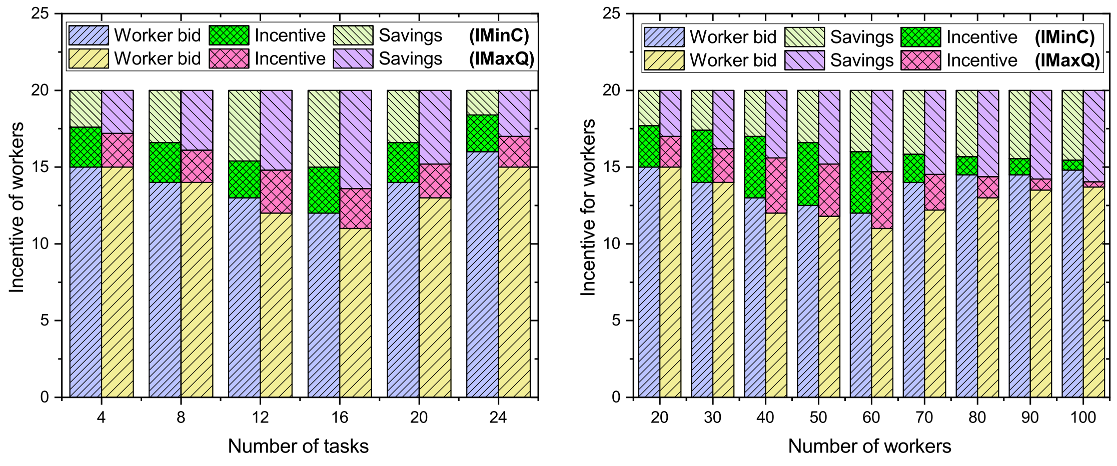
4.4.3.3 Impact of varying task sizes

In this experiment, we fixed the number of worker devices to $|\mathcal{K}| = 50$ and the number of tasks to $|\mathcal{M}| = 20$ and classified the application size into three categories: small $(25K - 50K)$, medium $(100K - 200K)$ and large $(400K - 500K)$ tasks.

As shown in Figure 4.8(a) and Figure 4.8(b), increasing the size of a task results in degradation of both user QoE and payment savings. This is due to the fact that increasing task size reduces the percentage of qualified worker devices to execute the task, resulting in relatively resource-poor workers being hired. For this reason, user QoE and user payment savings decrease with growing task size. However, due to the consideration of worker reputation, task dependency, and available computation resources, our proposed *IMaxQ* and *IMinC* algorithms still outperform the *Min-Cost* and *First-Fit* models, while *IMaxQ* has superior performance for user QoE and *IMinC* provides maximum cost savings. For the same reason, user satisfaction also decreases with growing application task size, as illustrated in Fig. 4.8(c). However, for allocation of tasks to reputed worker devices, our proposed *IMaxQ* and *IMinC* provide significantly better results compared to other algorithms.

4.4.3.4 Incentives for worker devices

In this experiment, we plotted the average amount of incentives received by different winning worker devices for executing a single application having 20 units of budget with fifty worker devices ($|\mathcal{K}| = 50$). Figure 4.9(a) shows the impact of increasing the number of tasks on the distribution amounts of the worker bids, incentives, and savings. Initially, size of an individual task was relatively bigger. As a result, a few workers were able to win the bid with high bid cost, resulting small payment savings and incentives. Dividing the application into more number of small-sized tasks invited more quality workers with lower bid cost, resulting higher payment savings, and incentives. However, further increasing number of tasks diminishes payment savings and incentives due to execution of tasks with relatively high bid cost workers. More specifically, in such situation, number of good quality workers is not enough to execute so many tasks. For the same reason, the graph follows similar trend for user payment savings. Comparatively, *IMaxQ* pays more incentive than *IMinC* as it prefers higher quality. Incentive performances for the *Min-Cost* and *First-Fit* algorithms are not presented here because these algorithms do not pay any additional incentives to workers above their bid amounts.



(a) Impact of increasing number of tasks (b) Impact of increasing number of workers

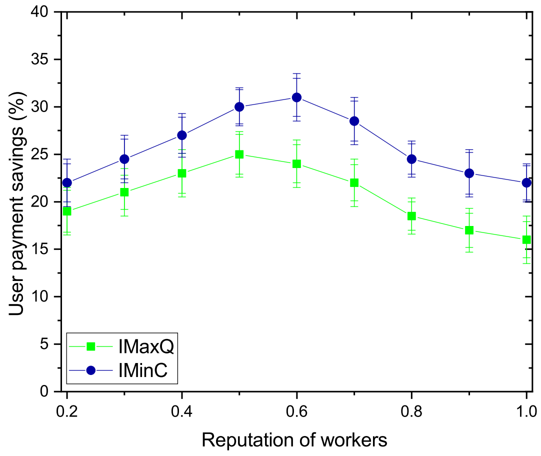
Figure 4.9: Incentive for workers

Similarly, increasing the number of worker devices causes more competition among the workers with fixed number of tasks ($|\mathcal{M}| = 20$), resulting in higher user QoE with small bid cost, as illustrated in Figure 4.9(b). Therefore, the user gets the opportunity to pay a higher incentive for the worker resources. However, further increasing the number of workers (> 70) resulted high quality execution from the worker devices though the incentive amount is decreased. At this point, due to high competition among the workers, the difference between the SLA quality and the provided quality of the workers becomes very little and hence the incentive amount is decreased even though the payment savings is increased. Though *IMaxQ* pays higher incentives than *IMinC*, the savings amount in Figure 4.9(b) is much higher compared to its counterpart shown in Figure 4.9(a). It is worth noting that although our proposed *IMaxQ* and *IMinC* algorithms provide incentives to workers, the savings are still higher than from other state-of-the-art-works due to the payment strategy. Our proposed resource allocation algorithms provide payment as the workers bid price, whereas other mechanisms use the next winner's bid cost to pay a worker, causing higher payment. However, our proposed allocation algorithms incentivize only the qualified workers, resulting in savings. Our in-depth look into the simulation trace files also reveals that, due to the payment of additional incentives, the participation of resourceful workers also increases gradually with the growing number of worker devices and execution of tasks with these worker devices upswing user-QoE.

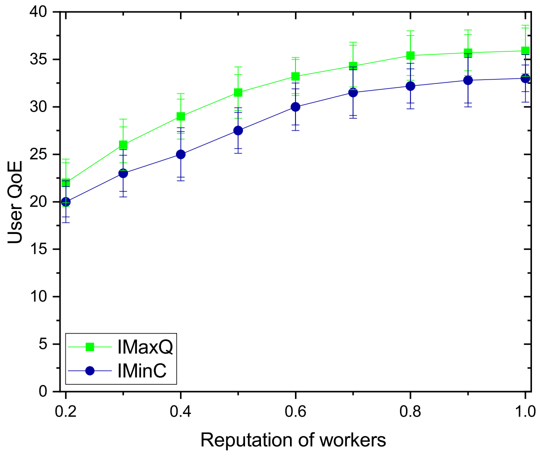
In summary, the amount of incentive received by a worker device is proportional to how much less time it takes for it to execute a task than the user deadline. Our in-depth analysis on results in the simulation trace file revealed that approximately 40 % of the workers are incentivized with an ample amount for offering excellent qualities of execution. Moreover, approximately 30 % of the workers are incentivized insignificantly due to bidding higher cost and offering just above SLA quality. The remaining workers are given negligible (or zero) incentives for just-in-time execution. Such a mechanism develops a win-win environment for users and workers, increasing the sustainability of the MDC system.

4.4.3.5 Impact of worker reputations

The impact of worker reputations on user payment savings and QoE are plotted in Figure 4.10. In these experiments, we execute an application having 200K instructions where the number of tasks and workers are fixed at 20 and 50 respectively.



(a) User payment savings



(b) Improvement of user QoE

Figure 4.10: Impact of worker reputation

Figure 4.10(a) shows that with the increasing number of reputation the user payment savings increases initially reaches a pick and starts to decrease after reaching the saturation. This is due to execution with non-reputed or untrustworthy workers, the user payment saving is very low at the beginning but increases with the growing reputation of the workers. With a very high reputation, the execution cost of the tasks also increases hence the payment saving decreases. Similarly, user QoE also increases with growing user reputation as timely execution with required quality are ensured with reputed workers as depicted in Figure 4.10(b). However, with very high reputation (≥ 0.6) the user QoE improvement is very little as the workers provide almost similar quality execution.

4.5 Summary

In this chapter, we have developed a novel computation framework for MDC, where mobile worker devices are incentivized in line with their execution qualities of user application tasks. Furthermore, consideration of resource capacity, reputation, and bid cost of worker devices helped our incentive mechanisms to harvest maximum benefits (in terms of quality or cost) through provisioning of user application tasks to high performing workers. The simulation results clearly indicate that the proposed system can maximize user QoE by as much as 35 % while minimizing cost by up to 30 %. The results also reveal that increasing the number of workers in the system can steadily improve user QoE, payment savings, and satisfaction, whereas larger task sizes can adversely affect MDC performance. Such a collaborative and distributed platform might play a vital role for developing a large machine learning model so as to further improve the performances.

Chapter 5
Conclusion

In this chapter, we summarize the research results presented in this thesis and state few directions for future works

5.1 Summary of Research

With the advancement of mobile computing technologies, the demand of mobile device users for better performance on the functionality and quality of experience of mobile applications has increased significantly. To meet the user demand, nowadays, each mobile device is equipped with a number of high-quality sensors, and complex artificial intelligence algorithms to process a huge volume of multimedia data. Significant research has been conducted recently to minimize the execution latency of compute-intensive applications through forming a Mobile Device Cloud (MDC), a collaborative cloud computing platform over which neighboring smart devices form an alliance of shared resources to mitigate resource-scarcity of an individual user device for running compute-intensive applications. An MDC supports the execution of resource-intensive and interactive mobile applications with lower latency by providing powerful computing resources to mobile devices through exploiting the shared resources of the nearby devices. An MDC can be envisioned in large scale stationary places (i.e. stadium, shopping mall, coffee shop or Movie Theater) or during traveling (air, bus, train) or scenarios where infrastructure may be infeasible and/or inefficient (e.g., natural disasters areas, heavily crowded regions, remote areas in military operations). However, the self-limitations of MDC technology restricts the widespread deployment. Our advanced study shows that there are a number of concerns that stand as vulnerabilities for implementing a sustainable MDC technology, including task interdependency, application deadline, resource heterogeneity, available energy, and compensation of worker devices, etc. In this thesis, we have focused on reliable worker device selection to speedup the execution performance of offloaded applications and providing payment to the worker devices according to their execution qualities. The different approaches available in state-of-the-art-works have considered as motivations and inspirations for developing solutions to such problems.

In this thesis, we first developed a code offloading framework named TESAR to provide improved performance for compute-intensive applications to the end-users. We formulate a mixed-integer linear programming (MILP) objective function with necessary constraints for assigning the computation tasks of an application on nearby worker devices so that the performances are optimized. We also provide an algorithm for developing a dependency tree among the tasks of an application so as to allow a higher number of parallel executions, wherever and whenever it is possible. The dependency tree aids the optimization framework in the ordering of tasks while scheduling. The optimization function helps to tradeoff between application execution speedup and reliability while maintaining device energy within a predefined range. Finally, the conducted emulation implemented in the android platform depicts that task execution with reliable workers can significantly speedup the execution performance while minimizing the application completion time, communication latency, and rescheduling overhead.

Next, we have addressed the joint problem of maximizing user quality-of-experience (QoE) at minimum cost while providing attractive incentives to workers' mobile devices. To the best of our knowledge, this is the first work where worker devices are incentivized in addition to their regular bid payment, according to their task execution quality to improve the user-QoE. We update the code offloading framework proposed in section 3.2.1 and incorporate a payment module to facilitate the payment and incentive related functionalities. The goal of the updated computational framework is to minimize the execution cost of user codes while supporting sufficient payments and incentives to the workers for their used resources. To increase the user-QoE with reduced cost, we formulated multi-objective linear programming (MOLP) optimization function that exploits reverse-auction bidding policy to provision the application tasks on workers with rich computation resources. Due to resource heterogeneity, the optimization formulation also ensures the selection of reputed workers in the execution of tasks. Due to the NP-hardness of MOLP, we offer two greedy worker selection algorithms for maximizing user QoE or minimizing execution cost. In both algorithms, the amount of incentive awarded to a worker is determined following the QoE offered to a user. Thus, the proposed algorithms motivate reliable and qualified worker's participation while rejects felonious workers in the system and ensures a sustainable system. We also conducted theoretical proofs of desirable properties of the proposed incentive mechanisms to ensure the correctness of the proposed algorithms. Simulation results illustrate that our incentive algorithms offer higher user-QoE, and payment savings while providing attractive incentives to qualified worker devices.

5.2 Discussion

Over the last few years, the interest in utilizing the computational resources of nearby mobile devices has been increased significantly. The MDC is such a collaborative environment where tasks of an application are executed collaboratively with the help of the resources of nearby mobile devices. A crucial benefit of MDC is that it does not relies on any preexisting infrastructure and can be formed easily in a large scale stationary place. We have explored the current task offloading mechanism and found that at present the tasks are offloaded to available worker devices without considering their capacity, and/ or reliability. We have also investigated that worker devices are not motivated in resource sharing due to a lack of compensation. We have reconnoitered that a payment mechanism can substantially increase the participation of workers with high computation resources, and increase the quality of the executed tasks.

My Ph.D. journey started with the aim to work with code offloading mechanisms in mobile cloud computing (MCC). Studying with the state-of-the-art works, we found the promising field of mobile device cloud (MDC). Although MDC, is a special kind of MCC, the formation and working functionality is significantly different from the regular MCC and thus the solution strategies are different for the same problem. It is also observed that MDC can be a momentous supplement to the regular cloud-based offloading mechanisms to offer enhanced performance for our real-life applications. Hence, we opt to develop a code offloading framework and formulated an optimization function to select reliable worker devices that speedup the computation of tasks and delivers an improved execution result. Later we found that a payment mechanism is necessary to compensate the worker devices for the used resources and to encourage their participation in the system. For this reason, we concentered on developing an incentive mechanism for the worker devices to acknowledge their qualified execution support.

To find an efficient solution to a problem, we have studied the state-of-the-art tools and techniques and acquainted with them. We explored different optimization techniques for modeling our code offloading framework to find an optimal solution. After the mathematical formulation, we had to go for simulation to evaluate the performance of the proposed model. Since MDC is a new technology, simulation tools are not equipped with the necessary features to conduct the performance analysis and hence we choose to develop an emulation testbed in the Android platform. For this reason, we had to spend a significant amount of time in learning Android and implementing the testbed. For the numerical evaluation, we had to learn MATLAB. Finally, after spending a significant amount of time in the laboratory, we had been able to implement our testbed which was tested with a good number of state-of-the-art-works to deliver satisfactory outcomes.

5.3 Future Works

In this thesis, we have presented the techniques for an improved code offloading mechanism in an MDC system. The developed framework and algorithms intend to build a collaborative code offloading environment that takes into account worker capacity, reliability, available energy, and interdependency among the tasks. Android platform was used to implement the experimental testbed while the MATLAB tool was used to investigate the numerical performance of the proposed algorithms. While the preliminary results of the performance looked encouraging, there are ample issues and open challenges that require more investigation to improve the performance further.

Although our proposed worker selection algorithms provide higher user-QoE with significant payment savings, further theatrical and experimental expansion are still possible. The present formulation is intended to work for a single application from a single user. In the future, investigating the problem of selecting the worker devices while tasks are offloaded from multiple users would be an interesting problem.

In this work, we have considered a fixed budget for the execution of an application that shrinks the worker selection due to budget constraints. Moreover, the work is focused on minimizing the execution cost of the user that hinders the profit of the worker. In the future, this work can be enhanced further by establishing an equilibrium between user budget and worker bids during on-demand provisioning of application tasks to workers.

Bibliography

[1] "Mobile app usage overview," Access Date: 19/01/2021. [Online]. Available: http://www.statista.com/topics/1002/mobile-app-usage/

[2] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices," in *MobiSys' 07: Proceedings of the 5th International Conference on Mobile Systems, Applications and Services.* ACM, 2007, pp. 272–285.

[3] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *2012 IEEE symposium on computers and communications (ISCC).* IEEE, 2012, pp. 59–66.

[4] Q. Shen, X. Liang, X. S. Shen, X. Lin, and H. Y. Luo, "Exploiting geo-distributed clouds for a e-health monitoring system with minimum service delay and privacy preservation," *IEEE journal of biomedical and health informatics*, vol. 18, no. 2, pp. 430–439, 2014.

[5] M. A. Habib, S. Saha, F. N. Nur, and M. A. Razzaque, "Starfish routing for wireless sensor networks with a mobile sink," in *Region 10 Conf (TENCON).* IEEE, 2016, pp. 1093–1096.

[6] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*, 2012, pp. 145–154.

[7] M. Satyanarayanan, "Fundamental challenges in mobile computing," in *Proc of the 15th annual symp on Principles of distributed computing.* ACM, 1996, pp. 1–7.

[8] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing." *HotCloud*, vol. 10, pp. 4–10, 2010.

[9] 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 *Proc of the 8th Int'l Conf on Mobile systems, applications, and services.* ACM, 2010, pp. 49–62.

[10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE Infocom.* IEEE, 2012, pp. 945–953.

[11] P. Balakrishnan and C.-K. Tham, "Energy-efficient mapping and scheduling of task interaction graphs for code offloading in mobile cloud computing," in *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing.* IEEE, 2013, pp. 34–41.

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

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

[14] N. Fernando, S. W. Loke, and W. Rahayu, "Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing.* IEEE, 2011, pp. 281–286.

[15] E. Miluzzo, R. Cáceres, and Y.-F. Chen, "Vision: mclouds-computing on clouds of mobile devices," in *Proc of the 3^{rd} workshop on Mobile cloud computing and services.* ACM, 2012, pp. 9–14.

[16] A. Mtibaa, K. A. Harras, and A. Fahim, "Towards computational offloading in mobile device clouds," in *Cloud Computing Technology and Science (CloudCom), 5^{th} Int'l Conf on*, vol. 1. IEEE, 2013, pp. 331–338.

[17] T. Penner, A. Johnson, B. Van Slyke, M. Guirguis, and Q. Gu, "Transient clouds: Assignment and collaborative execution of tasks on mobile devices," in *Global Commun Conf (GLOBECOM).* IEEE, 2014, pp. 2801–2806.

[18] X. Zhou, Y. Zhang, and T. Ma, "Computing offloading to save energy under time constraint among mobile devices," in *International Conference on Geo-Spatial Knowledge and Intelligence.* Springer, 2017, pp. 383–391.

[19] B. Li, Z. Liu, Y. Pei, and H. Wu, "Mobility prediction based opportunistic computational offloading for mobile device cloud," in *2014 IEEE 17th In-*

ternational Conference on Computational Science and Engineering. IEEE, 2014, pp. 786–792.

[20] Y. Li, L. Sun, and W. Wang, "Exploring device-to-device communication for mobile cloud computing," in *2014 IEEE international conference on communications (ICC).* IEEE, 2014, pp. 2239–2244.

[21] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *8^{th} Int'l Conf on cloud computing.* IEEE, 2015, pp. 9–16.

[22] X. Wang, X. Chen, W. Wu, N. An, and L. Wang, "Cooperative application execution in mobile cloud computing: A stackelberg game approach," *IEEE Commun Letters*, vol. 20, no. 5, pp. 946–949, 2016.

[23] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).* IEEE, 2014, pp. 352–357.

[24] H. Qian and D. Andresen, "Extending mobile device's battery life by offloading computation to cloud," in *Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on.* IEEE, 2015, pp. 150–151.

[25] Z. Cheng, P. Li, J. Wang, and S. Guo, "Just-in-time code offloading for wearable computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 1, pp. 74–83, 2015.

[26] M. E. Khoda, M. A. Razzaque, A. Almogren, M. M. Hassan, A. Alamri, and A. Alelaiwi, "Efficient computation offloading decision in mobile cloud computing over 5g network," *Mobile Networks and Applications*, vol. 21, no. 5, pp. 777–792, 2016.

[27] A. Mtibaa, K. A. Harras, K. Habak, M. Ammar, and E. W. Zegura, "Towards mobile opportunistic computing," in *8th Int'l Conf on Cloud Computing (CLOUD).* IEEE, 2015, pp. 1111–1114.

[28] 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.

[29] Y. Jararweh, L. Tawalbeh, F. Ababneh, and F. Dosari, "Resource efficient mobile computing using cloudlet infrastructure," in *2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks.* IEEE, 2013, pp. 373–377.

[30] J. Li, K. Bu, X. Liu, and B. Xiao, "Enda: Embracing network inconsistency for dynamic application offloading in mobile cloud computing," in *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, 2013, pp. 39–44.

[31] Y. Zhang, D. Niyato, P. Wang, and C.-K. Tham, "Dynamic offloading algorithm in intermittently connected mobile cloudlet systems," in *2014 IEEE international conference on communications (ICC).* IEEE, 2014, pp. 4190–4195.

[32] M. R. Mahmud, M. Afrin, M. A. Razzaque, M. M. Hassan, A. Alelaiwi, and M. Alrubaian, "Maximizing quality of experience through context-aware mobile application scheduling in cloudlet infrastructure," *Software: Practice and Experience*, vol. 46, no. 11, pp. 1525–1545, 2016.

[33] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2015.

[34] C. Wang, Y. Li, and D. Jin, "Mobility-assisted opportunistic computation offloading," *Communications Letters, IEEE*, vol. 18, no. 10, pp. 1779–1782, 2014.

[35] X. Lin, J. Jiang, C. H. Y. Li, B. Li, and B. Li, "Circa: collaborative code offloading among multiple mobile devices," *Wireless Networks*, pp. 1–19, 2018.

[36] M. Guiguis, Q. Gu, T. Penner, L. Tammineni, T. Langford, A. Rivera-Longoria, A. Johnson, and B. Van Slyke, "Assignment and collaborative execution of tasks on transient clouds," *Annals of Telecommunications*, vol. 73, no. 3-4, pp. 251–261, 2018.

[37] M. Le and Y.-W. Kwon, "Utilizing nearby computing resources for resource-limited mobile devices," in *Proceedings of the Symposium on Applied Computing*, 2017, pp. 572–575.

[38] V. Balasubramanian and A. Karmouch, "An infrastructure as a service for mobile ad-hoc cloud," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2017, pp. 1–7.

[39] B. Venkatraman, F. A. Zaman, and A. Karmouch, "Optimization of device selection in a mobile ad-hoc cloud based on composition score," in *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA).* IEEE, 2017, pp. 257–262.

[40] U. Drolia, R. Martins, J. Tan, A. Chheda, M. Sanghavi, R. Gandhi, and P. Narasimhan, "The case for mobile edge-clouds," in *2013 IEEE 10th Int'l Conf on Ubiquitous Intelligence and Computing and 2013 IEEE 10th Int'l Conf on Autonomic and Trusted Computing*, 2013, pp. 209–215.

[41] N. Fernando, S. W. Loke, and W. Rahayu, "Computing with nearby mobile devices: a work sharing algorithm for mobile edge-clouds," *IEEE transactions on cloud computing*, vol. 1, no. 99, pp. 1–14, 2017.

[42] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proc of the 1st Workshop on MCC & Services: Social Networks and Beyond.* ACM, 2010, pp. 6–10.

[43] L. A. BARROSO and U. HÖLZLE, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[44] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone." in *USENIX annual technical conference*, vol. 14. Boston, MA, 2010.

[45] J. M. Rodríguez, C. Mateos, and A. Zunino, "Are smartphones really useful for scientific computing?" in *International Conference on Advances in New Technologies, Interactive Interfaces, and Communicability.* Springer, 2011, pp. 38–47.

[46] Y.-S. Chang, S.-H. Hung, N. J. Wang, and B.-S. Lin, "Csr: A cloud-assisted speech recognition service for personal mobile device," in *2011 International Conference on Parallel Processing.* IEEE, 2011, pp. 305–314.

[47] B. Tatomir and L. Rothkrantz, "Crisis management using mobile ad-hoc wireless networks," in *Proceedings of the second international ISCRAM conference.* Citeseer, 2005.

[48] H. Chenji, W. Zhang, R. Stoleru, and C. Arnett, "Distressnet: A disaster response system providing constant availability cloud-like services," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2440–2460, 2013.

[49] Y.-D. Lee and W.-Y. Chung, "Wireless sensor network based wearable smart shirt for ubiquitous health and activity monitoring," *Sensors and Actuators B: Chemical*, vol. 140, no. 2, pp. 390–395, 2009.

[50] X. Li, X. Huang, C. Li, R. Yu, and L. Shu, "Edgecare: leveraging edge computing for collaborative data management in mobile healthcare systems," *IEEE Access*, vol. 7, pp. 22 011–22 025, 2019.

[51] B. Richerzhagen, D. Stingl, R. Hans, C. Gross, and R. Steinmetz, "Bypassing the cloud: Peer-assisted event dissemination for augmented reality games," in *14-th IEEE International Conference on Peer-to-Peer Computing.* IEEE, 2014, pp. 1–10.

[52] B. Shi, J. Yang, Z. Huang, and P. Hui, "Offloading guidelines for augmented reality applications on wearable devices," in *Proceedings of the 23rd ACM international conference on Multimedia*, 2015, pp. 1271–1274.

[53] T. Soyata, R. Muraleedharan, J. Langdon, C. Funai, S. Ames, M. Kwon, and W. Heinzelman, "Combat: mobile-cloud-based compute/communications infrastructure for battlefield applications," in *Modeling and Simulation for Defense Systems and Applications VII*, vol. 8403. International Society for Optics and Photonics, 2012, p. 84030K.

[54] T. Nishio, R. Shinkuma, T. Takahashi, and N. B. Mandayam, "Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud," in *Proceedings of the first international workshop on Mobile cloud computing & networking*, 2013, pp. 19–26.

[55] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 2, pp. 319–333, 2018.

[56] L. D. Xu, E. L. Xu, and L. Li, "Industry 4.0: state of the art and future trends," *International Journal of Production Research*, vol. 56, no. 8, pp. 2941–2962, 2018.

[57] M. A. Habib, S. Saha, M. A. Razzaque, M. Mamun-or Rashid, G. Fortino, and M. M. Hassan, "Starfish routing for sensor networks with mobile sink," *Journal of Network and Computer Applications*, vol. 123, pp. 11–22, 2018.

[58] X. Wang, X. Chen, and W. Wu, "Towards truthful auction mechanisms for task assignment in mobile device clouds," in *INFOCOM-IEEE Conf on Computer Communications*, 2017, pp. 1–9.

[59] S. Al Noor, R. Hasan, and M. M. Haque, "Cellcloud: A novel cost effective formation of mobile cloud based on bidding incentives," in *2014 IEEE 7th International Conference on Cloud Computing.* IEEE, 2014, pp. 200–207.

[60] L. Tang, S. He, and Q. Li, "Double-sided bidding mechanism for resource sharing in mobile cloud," *IEEE Trans on Vehicular Technology*, vol. 66, no. 2, pp. 1798–1809, 2017.

[61] S. Saha, M. A. Habib, and M. A. Razzaque, "Compute intensive code offloading in mobile device cloud," in *Region 10 Conf (TENCON).* IEEE, 2016, pp. 436–440.

[62] S. Saha, M. A. Habib, T. Adhikary, M. A. Razzaque, and M. M. Rahman, "Tradeoff between execution speedup and reliability for compute-intensive code offloading in mobile device cloud," *Multimedia Systems*, pp. 1–13, 2017.

[63] S. Saha, M. A. Habib, S. Sarkar, M. A. Razzaque, and M. M. Rahman, "Minimizing execution cost of user application codes in mobile device cloud," in *1st International Conference on Sustainable Technologies for Industry 4.0 (STI).* IEEE, 2019, pp. 90–95.

[64] N. Fernando, S. W. Loke, and J. W. Rahayu, "Mobile crowd computing with work stealing." in *NBiS*, 2012, pp. 660–665.

[65] T. Truong-Huu, C.-K. Tham, and D. Niyato, "To offload or to wait: An opportunistic offloading algorithm for parallel tasks in a mobile cloud," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science.* IEEE, 2014, pp. 182–189.

[66] H. Gao, J. Feng, R. Wang, and W. Wang, “A multi-task oriented selection strategy for efficient cooperation of collocated mobile devices,” in *International Conference on Algorithms and Architectures for Parallel Processing.* Springer, 2017, pp. 705–714.

[67] H. Viswanathan, P. Pandey, and D. Pompili, “Maestro: Orchestrating concurrent application workflows in mobile device clouds,” in *2016 IEEE International Conference on Autonomic Computing (ICAC).* IEEE, 2016, pp. 257–262.

[68] S. M. Asghari, Y.-H. Kao, M. H. Lotfi, M. Noormohammadpour, B. Krishnamachari, B. H. Khalaj, and M. Katz, “Lord: Leader-based framework for resource discovery in mobile device clouds,” *arXiv preprint arXiv:1308.0959*, 2013.

[69] L. Li, J. Peng, W. Liu, F. Jiang, and S. Li, “Energy optimization for task offloading based on auction mechanism in ad hoc mobile cloud,” in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC).* IEEE, 2017, pp. 245–250.

[70] X. Wang, Y. Sui, J. Wang, C. Yuen, and W. Wu, “A distributed truthful auction mechanism for task allocation in mobile cloud computing,” *IEEE Trans on Services Computing*, 2018.

[71] J. He, D. Zhang, Y. Zhou, X. Lan, and Y. Zhang, “Towards a truthful online auction for cooperative mobile task execution,” in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced &*

Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). IEEE, 2018, pp. 546–553.

[72] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.

[73] 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*, 2011, pp. 43–56.

[74] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2253–2266, 2014.

[75] A.-L. Jin, W. Song, P. Wang, D. Niyato, and P. Ju, "Auction mechanisms toward efficient resource sharing for cloudlets in mobile cloud computing," *IEEE Transactions on Services Computing*, vol. 9, no. 6, pp. 895–909, 2015.

[76] A.-L. Jin, W. Song, and W. Zhuang, "Auction-based resource allocation for sharing cloudlets in mobile cloud computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 1, pp. 45–57, 2015.

[77] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *Proceedings of the 12th conference on Hot topics in operating systems.* USENIX Association, 2009, pp. 8–8.

[78] E. E. Marinelli, “Hyrax: cloud computing on mobile devices using mapreduce,” Carnegie-mellon univ Pittsburgh PA school of computer science, Tech. Rep., 2009.

[79] C. Shi, M. H. Ammar, E. W. Zegura, and M. Naik, “Computing in cirrus clouds: the challenge of intermittent connectivity,” in *Proceedings of the 1th edition of the MCC workshop on Mobile cloud computing.* ACM, 2012, pp. 23–28.

[80] N. Fernando, S. W. Loke, and W. Rahayu, “Honeybee: A programming framework for mobile crowd computing,” in *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services.* Springer, 2012, pp. 224–236.

[81] P. Pandey, H. Viswanathan, and D. Pompili, “Robust orchestration of concurrent application workflows in mobile device clouds,” *Journal of Parallel and Distributed Computing*, vol. 120, pp. 101–114, 2018.

[82] K. Habak, C. Shi, E. W. Zegura, K. A. Harras, and M. Ammar, “Elastic mobile device clouds: Leveraging mobile devices to provide cloud computing services at the edge,” *Fog for 5G and IoT*, pp. 159–188, 2017.

[83] V. Balasubramanian, F. Zaman, M. Aloqaily, S. Alrabaee, M. Gorlatova, and M. Reisslein, “Reinforcing the edge: autonomous energy management for mobile device clouds,” in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops.* IEEE, 2019, pp. 44–49.

[84] J. He, D. Zhang, Y. Zhou, and Y. Zhang, “A truthful online mechanism for collaborative computation offloading in mobile edge computing,” *IEEE Transactions on Industrial Informatics*, 2019.

[85] E. H. Clarke, “Multipart pricing of public goods,” *Public choice*, pp. 17–33, 1971.

[86] V. Krishna, *Auction theory.* Academic press, 2009.

[87] V. Kulkarni, A. Moro, and B. Garbinato, “Mobidict: a mobility prediction system leveraging realtime location data streams,” in *Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoStreaming.* ACM, 2016, pp. 1–10.

[88] “N. optimization server,” Access Date: 06/08/2016. [Online]. Available: http://www.neos-server.org/neos/

[89] M. M. Hassan, B. Song, M. S. Hossain, A. Alamri, M. A. Alnuem, M. M. Monowar, and M. A. Hossain, “Efficient virtual machine resource management for media cloud computing.” *TIIS*, vol. 8, no. 5, pp. 1567–1587, 2014.

[90] “Tp-link td-w8901n wireless router specification,” Access Date: 10 Feb 2017. [Online]. Available: http://www.tp-link.com.bd/products/details/cat-15-TD-W8901N.html

[91] K. Xie, X. Wang, G. Xie, D. Xie, J. Cao, Y. Ji, and J. Wen, “Distributed multi-dimensional pricing for efficient application offloading in mobile cloud computing,” *IEEE Trans on Services Computing*, 2016.

[92] Y. Zhang, J. Yan, and X. Fu, "Reservation-based resource scheduling and code partition in mobile cloud computing," in *2016 IEEE Conference on Computer Communications Workshops.* IEEE, 2016, pp. 962–967.

[93] M. Golkarifard, J. Yang, Z. Huang, A. Movaghar, and P. Hui, "Dandelion: A unified code offloading system for wearable computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 3, pp. 546–559, 2018.

[94] A. Shye, B. Scholbrock, G. Memik, and P. A. Dinda, "Characterizing and modeling user activity on smartphones: summary," in *SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1. ACM, 2010, pp. 375–376.

[95] X. Guo, L. Liu, Z. Chang, and T. Ristaniemi, "Data offloading and task allocation for cloudlet-assisted ad hoc mobile clouds," *Wireless Networks*, vol. 24, no. 1, pp. 79–88, 2018.

[96] F. Lu, L. Gu, L. T. Yang, L. Shao, and H. Jin, "Mildip: An energy efficient code offloading framework in mobile cloudlets," *Information Sciences*, vol. 513, pp. 84–97, 2020.

[97] H. Moore, *MATLAB for Engineers.* Pearson, 2017.

[98] K. Xu, Y. Zhang, X. Shi, H. Wang, Y. Wang, and M. Shen, "Online combinatorial double auction for mobile cloud computing markets," in *IEEE 33rd Int'l Performance Computing and Communications Conf (IPCCC).* IEEE, 2014, pp. 1–8.

[99] S. Sarker, M. A. Razzaque, M. M. Hassan, A. Almogren, G. Fortino, and M. Zhou, "Optimal selection of crowdsourcing workers balancing their utili-

ties and platform profit," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8602–8614, 2019.

[100] B. Kantarci and H. T. Mouftah, "Reputation-based sensing-as-a-service for crowd management over the cloud," in *Int'l Conf on Communications (ICC).* IEEE, 2014, pp. 3614–3619.

[101] Q. Wu, X. Zhang, M. Zhang, Y. Lou, R. Zheng, and W. Wei, "Reputation revision method for selecting cloud services based on prior knowledge and a market mechanism," *The Scientific World Journal*, vol. 2014, 2014.

[102] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi, "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions," *Journal of Network and Computer Applications*, vol. 48, pp. 99–117, 2015.

[103] S. Saha and M. S. Hasan, "Effective task migration to reduce execution time in mobile cloud computing," in *Automation and Computing (ICAC), 2017 23^{rd} Int'l Conf on.* IEEE, 2017, pp. 1–5.

[104] T. Dbouk, A. Mourad, H. Otrok, H. Tout, and C. Talhi, "A novel ad-hoc mobile edge cloud offering security services through intelligent resource-aware offloading," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1665–1680, 2019.

[105] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Trans on Mobile Computing*, vol. 16, no. 11, pp. 3056–3069, 2017.

[106] D. Dereniowski and W. Kubiak, “Shared multi-processor scheduling,” *European Journal of Operational Research*, vol. 261, no. 2, pp. 503–514, 2017.

[107] T. Sigwele, A. S. Alam, P. Pillai, and Y. F. Hu, “Evaluating energy-efficient cloud radio access networks for 5g,” in *2015 IEEE international conference on data science and data intensive systems.* IEEE, 2015, pp. 362–367.

Appendix A
List of Notations

$\mathcal{M}$	Set of advertised tasks in the application
$\mathcal{K}$	Set of candidate worker devices
$\mathcal{V}$	Set of winning worker devices
$\mathcal{B}$	Set of bids submitted by worker devices
$\mathcal{W}$	Set of winning bids
Π_m	Set of parents of task m
L	Set of Leaf nodes
$\mathbb{R}$	Set of tasks executed Remotely
$\mathbb{L}$	Set of tasks executed Locally
$\mathcal{S}_m^o$	Instruction size of task $m \in \mathbb{R}$
$\mathcal{S}_m^o\prime$	Instruction size of task $m \in \mathbb{L}$
$\mathbb{V}_m$	Output instruction size of task $m \in \mathcal{M}$
μ_k	Execution speed of device $k \in \mathcal{K}$
$\mathcal{T}_m^l(o)$	Local device execution time of task $m \in \mathbb{R}$
$\mathcal{T}_m^l(o\prime)$	Local device execution time of task $m \in \mathbb{L}$
$\mathcal{T}_{m,k}^x$	Remote execution time of task $m \in \mathbb{R}$ in device $k \in \mathcal{K}$
$\mathcal{T}_{m,k}^t$	Input transmission time of task $m \in \mathbb{R}$
$\mathcal{T}_{m,k}^r$	Output reception time of task $m \in \mathbb{R}$

$\mathbb{B}_k^d$	Uplink bandwidth between cloudlet and device $k \in \mathcal{K}$
$\mathbb{B}_k^d{\prime}$	Downlink bandwidth between cloudlet and device $k \in \mathcal{K}$
$\mathcal{E}_{m,k}^x$	Execution energy of task $m \in \mathcal{R}$ in $k \in \mathcal{K}$
$\mathcal{E}_{m,k}^t$	Input transmission energy of task $m \in \mathcal{R}$ in $k \in \mathcal{K}$
$\mathcal{E}_{m,k}^r$	Output transmission energy of task $m \in \mathcal{R}$ in $k \in \mathcal{K}$
γ_k	Signal strength of device $k \in \mathcal{K}$
η_k	Associativity time of device $k \in \mathcal{K}$
E_k	Available energy of device $k \in \mathcal{K}$
Ω_k	Reputation of device $k \in \mathcal{K}$
$\phi_{n,m}$	Percentage of dependency of a child task $m \in \mathcal{M}$ on its parent $n \in \mathcal{M}$
$\mathcal{S}_m$	Number of instructions in task $m \in \mathcal{M}$
$\mathcal{T}_m$	Execution deadline of task $m \in \mathcal{M}$
$\mathcal{R}_m$	Resource requirement for executing task $m \in \mathcal{M}$
$\mathcal{C}_m^k$	Claimed cost of worker $k \in \mathcal{K}$ for task $m \in \mathcal{M}$
$\mathcal{Q}_m^k$	Offered quality of worker $k \in \mathcal{K}$, $m \in \mathcal{M}$
Ω_m^k	Earned reputation of worker $k \in \mathcal{K}$, $m \in \mathcal{M}$
$\mathcal{P}_m^k$	Payment of worker $k \in \mathcal{K}$ for task $m \in \mathcal{M}$
$\mathcal{U}_m^0$	Utility of cloudlet for serving task $m \in \mathcal{M}$
$\mathcal{U}_m^k$	Utility of worker $k \in \mathcal{K}$ for task $m \in \mathcal{M}$

Appendix B
List of Acronyms

AI	Artificial Intelligence
AP	Access Point
CPU	Central Processing Unit
D2D	Device to Device
GPS	Global Positioning System
ILP	Integer Linear Programming
IMaxQ	Incentive for Maximizing Quality
IMinC	Incentive for Minimizing Cost
IoT	Internet of Things
MCC	Mobile Cloud Computing
MDC	Mobile Device Cloud
MILP	Mixed-Integer Linear Programming
MOLP	Multi Objective Linear Optimization
NP	Non-deterministic Polynomial
PM	Payment Manager
PRL	Performance Resource-Load
QoE	Quality of Experience
QoS	Quality of Service

RAM	Random Access Memory
SLA	Service Level Agreement
TESAR	Tradeoff between Execution Speedup And Reliability
TP	Task Profiler
TWA	Task-Worker Allocator
EC	Execution Coordinator
VCG	Vickrey-Clarke-Groves
VM	Virtual Machine
Wi-Fi	Wireless Fidelity

Appendix C
List of Publications

International Journal Papers (SCI-indexed)

1. __"Quality-of-Experience-Aware Incentive Mechanism for Workers in Mobile Device Cloud." IEEE Access 9 (2021): 95162-95179.

2. ______________________________"Tradeoff between execution speedup and reliability for compute-intensive code offloading in mobile device cloud", Multimedia Systems, Springer, pp.1-13, 2017.

3. "Starfish routing for sensor networks with mobile sink.", Journal of Network and Computer Applications, Elsevier, Volume 123, pp.11-22, 2018.

4. ________________"Lifetime Maximization of Sensor Networks Through Optimal Data Collection Scheduling of Mobile Sink." IEEE Access 8 (2020): 163878-163893.

International Conference Papers

5. ________"Minimizing Execution Cost of User Application Codes in Mobile Device Cloud" In 2019 International Conference on Sustainable Technologies for Industry 4.0 (STI), pp. 1-5. IEEE, 2019. (Best Paper Award)

6. ____"Effective task migration to reduce execution time in mobile cloud computing", in Automation and Computing (ICAC), 2017 23rd Intl Conf on. IEEE, 2017, pp. 15. (Best Paper Award)

7. _____________"Compute Intensive Code Offioading In Mobile Device Cloud" In IEEE TENCON 2016 - Technologies for Smart Nation,22 - 25 November 2016, Marina Bay Sands, Singapore.

8. _________"On The Optimal Size of Ring Canal in Starfish Routing." In 2019 International Conference on ICIEV, IEEE, 2019.

9. ________________"An Efficient Mobile-Sink Trajectory to Maximize Network Lifetime in Wireless Sensor Network." In 2018 International Conference on Innovation in Engineering and Technology (ICIET), pp. 1-5. IEEE, 2018.

10. _______"Starfish Routing for Wireless Sensor Networks with a Mobile Sink." In IEEE TENCON 2016 - Technologies for Smart Nation, 22 - 25 November 2016, Marina Bay Sands, Singapore.