# Efficient and Proactive Offloading Techniques for Sustainable and Mobility-aware Resource Management in Heterogeneous Mobile Cloud Environments

by

Shichao Guan

Thesis submitted to the University of Ottawa
In partial fulfillment of the requirements
For the Ph.D. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

# Abstract

To support increasingly sophisticated sensors and resource-hungry applications with the current-used Lithium-based batteries and to augment mobile computing power further, the concept of the Cloudlet-based offloading is proposed which enables to migrate part of application computing tasks from battery-limited low-capacity mobile elements to the local edge. Such Cloudlet-based offloading technologies extend the provisioning of computing and storage capabilities from remote Cloud Data Centers to the proximity of end users via heterogeneous networks. However, Cloudlet-based offloading is required to coordinate among User Equipment, inter-Cloudlet nodes and remote Cloud Data Centers, which emerges new challenges and issues regarding how to enable Cloudlet-based offloading in the context of mobile edge environment and how to achieve execution- and energy-efficient offloading allocation under limited available resources.

In this dissertation, a Cloudlet-based Mobile Cloud offloading prototype is first proposed. A mechanism for handling diverse computing resources is described; by adopting it, idle public resources can be easily configured as additional computing capabilities in the virtual resource pool. A fast deployment model is built to relieve the migration and installation cost when adapting the platform. An energy-saving strategy is utilized to reduce the consumption of computing resources. Security components are implemented to protect sensitive information and block malicious attacks in the cloud.

Concerning the limited processing capability on the edge, a task-centric energy-aware Cloudlet-based Mobile Cloud model is formulated. A Cloudlet task-based offloading mechanism is proposed to achieve energy-aware offloading resource preparation and scheduling on the Cloudlet. A Cloud task-centric scheduling algorithm is presented for the green collaborative offloading processing between Cloudlet and remote Cloud.

Considering the dynamic and heterogeneity of the offloading environment, a hybrid offloading model to solve the heterogeneous resource-constraint offloading issues on the dynamic Cloudlets. A queue-based offloading framework is developed to formulate and analyze the mixed migration-based and partition-based offloading behaviours on the Cloudlet. The execution and energy-aware heterogeneous offloading resource allocation problem is formalized and solved. A time series-based load prediction model is designed on the Cloudlet to achieve fine-grain proactive resource allocation.

Regarding the mobility of User Equipment and the diverse priority of offloading tasks, an edge-based mobility-aware offloading model is modeled to solve the intra-Cloudlet offloading scheduling issue and inter-Cloudlet load-aware heterogeneous resource allocation issue. A priority-based queueing model is designed to formulate the intra-Cloudlet

mobility-aware offloading scheduling problem, resolved by a heuristic solution. The energy-aware inter-Cloudlet resource selection procedure is formalized in a mobility-aware multi-site resource allocation model, which is further solved by lightweight dynamic load balancing.

# List of Publications

## Referred Journal Papers

- Shichao Guan, Azzedine Boukerche: Efficient and Scalable Proactive Offloading Techniques for Sustainable Cloudlet Resource Allocation in Heterogenous Mobile Cloud Environments. IEEE Trans. Sustainable Computing. To appear.
- Shichao Guan, Azzedine Boukerche: A Mobility-aware Energy-efficient Model for MEC-based Offloading. IEEE Trans. Sustainable Computing. To appear.
- Azzedine Boukerche, Shichao Guan, Robson E. De Grande: Sustainable Offloading in Mobile Cloud Computing: Algorithmic Design and Implementation. ACM Comput. Surv. 52(1): 11:1-11:37 (2019).
- Shichao Guan, Robson Eduardo De Grande, Azzedine Boukerche: A Multi-Layered Scheme for Distributed Simulations on the Cloud Environment. IEEE Trans. Cloud Computing 7(1): 5-18 (2019).
- Azzedine Boukerche, Shichao Guan, Robson Eduardo De Grande: A Task-Centric Mobile Cloud-Based System to Enable Energy-Aware Efficient Offloading. IEEE Trans. Sustainable Computing 3(4): 248-261 (2018).
- Robson Eduardo De Grande, Azzedine Boukerche, Shichao Guan, Noura Aljeri: A modular distributed simulation-based architecture for intelligent transportation systems. Concurrency and Computation: Practice and Experience 28(12): 3409-3426 (2016).

## Referred Conference Papers

- Shichao Guan, Azzedine Boukerche, Samaneh Ahmadvand: A Cloudlet-based Mobile Computing Model for Resource and Energy Efficient Offloading. ISCC 2018: 980-985.
- Shichao Guan, Robson Eduardo De Grande, Azzedine Boukerche: A Cloudlet-based task-centric offloading to enable energy-efficient mobile applications. ISCC 2017: 564-569.
- Shichao Guan, Robson Eduardo De Grande, Azzedine Boukerche: An HLA-Based Cloud Simulator for Mobile Cloud Environments. DS-RT 2016: 128-135.
- Shichao Guan, Robson Eduardo De Grande, Azzedine Boukerche: A novel energy efficient platform based model to enable mobile Cloud applications. ISCC 2016: 914-919.
- Shichao Guan, Robson Eduardo De Grande, Azzedine Boukerche: Enabling HLA-based Simulations on the Cloud. DS-RT 2015: 112-119.

**Acknowledgements**

I am very grateful to Prof. Azzedine Boukerche, who has offered me the precious opportunity to study in the Paradise Lab, provided all the equipment that necessary for my experiments, offered me RA funding to support my study, and showed me a way through the mysterious scientific world. Like a mariner, I can chart my course by his lights.

I am also very appreciative of the kind support from Prof. Robson, who has always illuminated me throughout the first three years in my PhD. His rigorous studying attitude and passion encourages me to keep moving forward.

I thank all the members in the Paradise Lab. Thank you for all your help.

Last but not least, I would like to give my thanks to my father, my mother and my wife.

# Abbreviations

**CC** : Cloud Computing
**UE**: User Equipment
**MCC**: Mobile Cloud Computing
**MEC**: Multi-access Edge Computing
**EC**: Edge Computing
**QoS**: Quality of Service
**API**: Application Programming Interface
**SMD**: Smart Mobile Devices
**WLAN**: Wireless Local Area Network
**BS**: Base Station
**RAN**: Radio Access Network
**RPC**: Remote Procedure Call
**DVFS**: Dynamic Voltage and Frequency Scaling
**VM**: Virtual Machine
**VMM**: Virtual Machine Monitor
**JVM**: Java Virtual Machine
**KVM**: Kernel-based Virtual Machine
**CLR**: Common Language Runtime
**RTT**: Round Trip Time
**PSM**: Power Save Mode
**DTMC**: Discrete Time Markov Chain
**MDP**: Markov Decision Process
**ILP**: Integer Linear Programming
**SPNE**: Sub-game Perfect Nash Equilibrium
**MAC**: Medium Access Control
**KKT**: Karush–Kuhn–Tucker
**MPT**: Microwave Power Transfer
**MIMO**: Multi Input Multi Output
**DNN**: Deep Neural Networks
**CDIA**: Cloud-based Distributed Interactive Application
**DIA**: Distributed Interactive Application
**DFGCP**: Data Flow Graph Critical Path
**CPA**: Critical Path Assignment

**ABC**: Artificial Bee Colony
**NIC**: Network Interface Card
**MTC**: Many Task Computing
**AOC**: Ant Colony Optimization
**CP**: Constraint Programming
**GA**: Genetic Algorithm
**PSO**: Particle Swarm Optimization
**HLA**: High Level Architecture
**DIS**: Distributed Interactive Simulation
**DDS**: Data Distributed Service
**NFS**: Network File System
**VT**: Virtualization Technology
**IaaS**: Infrastructure as a Service
**PaaS**: Platform as a Service
**SaaS**: Software as a Service
**WAN**: Wide Area Network
**ACF**: Autocorrelation Function
**PACF**: Partial Autocorrelation Function
**Akaike Information Criterion**: AIC
**BF**: Brutal Force

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

The technological advancements in mobile devices in recent years have enabled users to utilize sophisticated sensors and powerful computing capability ubiquitously. However, the compact size of the mobile elements restricts the battery lifespan and available power to process resource-hungry computational tasks. To deal with this issue, the Mobile Cloud offloading architecture was proposed, which leveraged high-end remote Cloud resources to process the small size computational-intensive mobile tasks externally. The Cloud Computing (CC) has been dramatically developed and widely deployed in the last decade since it was formally defined and standardized [190]. The softwarization and virtualization technologies encapsulate resource management in a centralized manner, which enables elastic and rapid provisioning of computational power, storage space, and network resources throughout distributed locations. Due to the maturity of CC, CC's resources are promising to serve the emergence of diverse and sophisticated mobile applications, which are empowered by the revolution of mobile networks and the capacity of User Equipment (UE). The novel Mobile Cloud Computing (MCC) framework differs from the traditional CC paradigm – MCC visualizes service provisioning via a collection of mobile-aware computational and information utilities in the mobile computing environment, concerning additional features and functionalities that include UE energy consumption evaluation, location and interest awareness, wireless network cost calculation and trace prediction. The MCC paradigm integrates CC and Mobile Computing, seeking to support cross-platform resource provisioning on a considerably reduced cost.

However, MCC suffers from long communication delay and limited backhaul bandwidth due to its centralized architecture. Based on Moore's Law, the computational capability in UEs has drastically advanced, which in turn triggers resource-hungrier and

delay-sensitiver applications ubiquitously realized in the current modern life. Besides, the evolution of mobile networks has led to pervasive IoT sensors and massive communication data among the network channels. These trends appeal researchers to reconsider the trade-off between computational gain and connection overhead when accommodating the MCC techniques. The increasingly stringent task deadline and the vast amount of data routed in the network yield revolutionary solutions that can provision CC services and functions more efficiently.

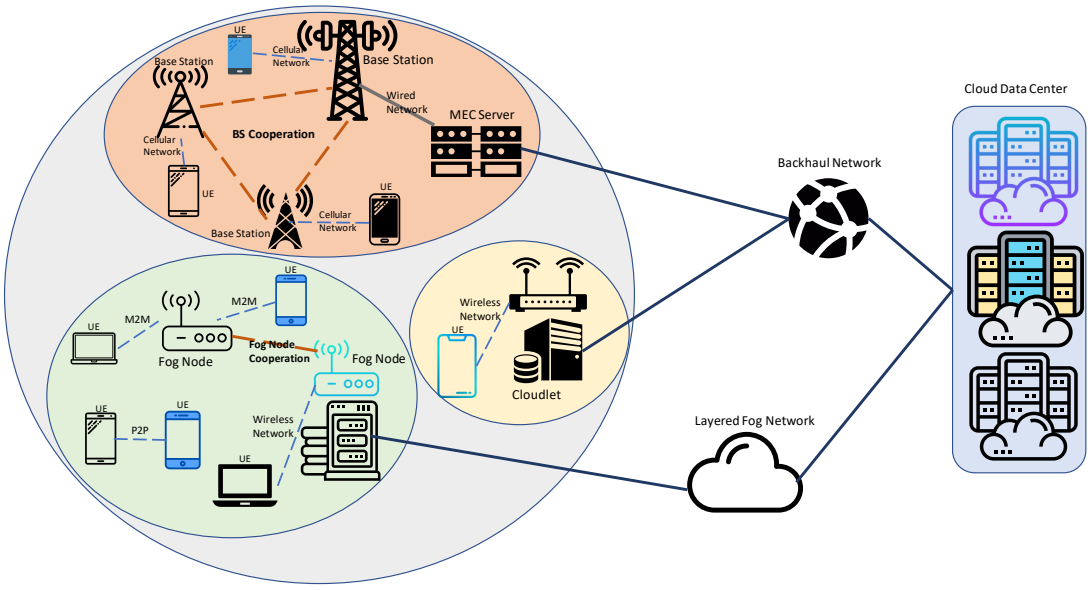



Figure 1.1: Edge Computing Architecture

Concerning such challenges of MCC, the Mobile Edge concept model is emerging as a feasible solution, which shifts the services from centralized CC to the distributed network edge – into the vicinity of end-users. This trend in computing augments the increasing number of UEs by harvesting computation and storage capacity via proximate access, as shown in Figure 1.1. The first edge-based computational model was proposed in the work [220] where the Cloudlet performs as the intermediate platform between UE and CC. This central component extends CC services in a self-managed box, typically deployed at the public edge areas accessible via the one-hop wireless connection. The Multi-access Edge Computing (MEC) [152], initialized known as Mobile Edge Computing, was initially standardized to offer CC services in base stations. Regarding the rapid increasing of non-cellular volume, the default specification was accordingly renamed to incorporate third-party open networks. The Fog Computing [30], as the name suggested,

was proposed as another edge technology trying to provision services closer to UEs on the ground than CC. Compared to MCC, the three edge-based techniques substantially mitigate the end-to-end communication delay and backhaul network load. However, Edge Computing (EC) is still in the infancy and how to realize its potential to support diverse UEs and complex tasks in a large-scale context within time-constraint QoS makes the edge-based efficient service provisioning an open research problem.

## 1.1 Motivation

Therefore, this thesis focuses on the execution- and energy-efficient offloading and resource management issues on one of the edge techniques – the Cloudlet. The Cloudlet typically performs as the edge hosts, which is integrated with Cloud-like virtualization and could receive offloading requests via the cellular networks, local networks or external networks under the coverage, and process offloading tasks in the residing virtual instances.

The Cloudlet-based offloading framework can significantly reduce communication distance and achieve centralized resource management. However, enabling Cloudlet-based energy-efficient offloading remains challenging. The bottleneck resides on the computing and storage capability of the local Cloudlet. Also, although several studies have extended the capability of Cloudlets during run-time by integrating public Cloud resources in a pay-as-you-go manner, most of these current studies assume the offloading is processed under a well-prepared homogeneous environment. In addition, the mobility of UEs may introduce intermittent connectivity during offloading and cause unbalanced load among multiple Cloudlets, which have been proved to be the root cause of offloading failure and service downgrading, as shown in [7] and [259].

Overall speaking, the capacity on Cloudlets, the heterogeneity of environment and the mobility of UE fundamentally influence the performance of Cloudlet-based offloading, in terms of offloading execution time and energy cost on both UE and Cloudlet.

## 1.2 Objectives

This thesis focuses on the offloading and resource allocation in the context of Cloudlet-based MCC. The main objectives of the thesis are to propose analytical models, design and implement protocols and prototypes for execution- and energy-efficient offloading.

Concerning the challenges mentioned in the motivation, the following research problems are defined and investigated in this thesis.

- **How to deploy an offloading-run-ready Cloudlet among different types of computing resources in a hybrid environment?** For offloading managers who may not be familiar with the management of distributed systems, it is challenging to make an offloading-run-ready environment among different types of computing resources and network environments. In addition, the existing Cloudlets are mainly implemented based on local resources. In this case, to end-users, the capabilities available for provisioning are limited to the size of its local resource pool. This is not in accordance with the essential characteristic of elasticity in Cloud Computing, according to the definition from the National Institute of Standards and Technology [190]. In addition, the issue related to Cloudlet energy consumption is needed to be addressed, particularly for offloading applications that may require relatively intensive resources to execute. Therefore, an energy-aware execution-efficient Cloudlet-based hybrid offloading framework is required to tackle the details of underground resource management and environment preparation.

- **How to reduce the execution cost on Cloudlet and Cloud Resources?** The traditional device-centric Cloudlet schemes are restricted to passively tracing the device-level mobility, performing the device-clone local offloading execution and a one-on-one device-instance mapping on the remote Cloud. Thus, in the large-scale multi-user offloading scenario, only a small portion of users can get access to the local offloading resources while a considerable number of offloading requests are rejected or forwarded to the remote Cloud datacenter. These offloading requests may severely suffer from the long-distance communication overhead due to the lack of local computing capability. Also, the traditional device-clone based resource management schemes produce unnecessary resource preparation overhead. The overhead is introduced by preparing the task clones that are out of the local user interests and are never triggered. Moreover, the device-instance-based Cloud resource management is not energy-friendly as the Cloud cannot shutdown scheduling policies due to the running device. In this case, a task-centric fine-grain Cloudlet-based MCC system is required to optimize the utilization of Cloudlet resources and reduce the energy cost on the Cloud with reasonable Quality of Service (QoS).

- **How to model the dynamic and heterogeneous environment and opti-**

**mize offloading performance in such an environment?** In several offloading models, either partitioning-based task-level offloading [108] or migration-based device-level offloading [220] is utilized as the only offloading protocol supported on the homogeneous resources. As a result, these studies cannot adequately achieve energy-efficient offloading in the real offloading scenarios that involve heterogeneous offloading methods, varying preparation overhead, and diverse offloading resources on the Cloudlet. An analytical model is required to concern the proactive and heterogeneous offloading and resource allocation issue.

- **How to handle the UE mobility during offloading?** Many efforts have recently been made to address the mobility impacts on MEC-based offloading. These studies have extended the offloading decision-making functionality with mobility awareness, integrating strategies such as live migration, and multi-site load balancing. The migration-based work has focused on the connections between the offloading virtual instance and the UE, trying to minimize the communication distance [221] or optimize the communication path [206]. The load balancing approach, as in [125] and [28], targets the mobility issue in the multi-site offloading scenario, seeking to improve offloading efficiency by balancing system load among adjacent resources. Although the inter-Cloudlet mobility issue has been discussed, as shown above, the intra-Cloudlet mobility-aware offloading scheduling issue is not sufficiently concerned. The intra-Cloudlet power control study [181] attempted to dynamically adjust the transmission power of the Cloudlet based on the movements of UEs to reduce the number of overall handovers, but the work did not tackle the resource scheduling details. In terms of the intra-Cloudlet offloading scheduling works, First In First Out (FIFO) or min-min heuristic processing is typically utilized as the default queueing execution strategy, as in the M/M/c/K queue-based work [242] and M/G/1 queue-based work [245]. However, these non-priority protocols cannot adequately handle the heterogeneity in Cloudlet-based offloading that involves different mobility patterns, varying task property, and diverse resource capabilities. As a result, a large amount of offloading requests may be either improperly processed or unnecessarily rejected, due to the lack of prioritizing. A mobility-aware priority-based offloading resource management scheme is needed.

## 1.3 Contribution

Concerning such research problems mentioned in the objectives, the contribution of this dissertation is summarized in this section.

- **A survey and an overview of the state-of-the-art on energy-awareness in the MCC-based offloading and related Green Cloud computing strategies**. It introduces an ample view over the elements that concern energy aspects towards the profiling of applications and resources, partitioning of applications tasks, and the actual decision-making procedures from the perspective of both mobile devices and Cloud systems with discussions on simulation-based experimental evaluations and implementation-based principles. The survey also develops further on the actual open challenges in MCC, delineating the possible research directions in the area.

- **A Cloudlet-based multi-layered prototype for offloading in the MCC environment**. A multi-layered local Cloudlet offloading scheme is proposed, which takes the distributed simulation application as an offloading example, concerning usability, elasticity, energy consumption, and fast deployment. Components are designed and implemented to provide unlimited computing resources to end-users by coordinating public computing resources during run-time. Energy-aware elements are utilized to reduce unnecessary energy cost from computing resources. A deployment model is designed to accelerate the migration and deployment process of the cloud offloading platform. In addition, the scheme contains functions for job scheduling, monitoring, and a friendly web-based graphic interface to ease the configuration, operation, and maintenance of the underlying system.

- **A task-centric mobile Cloud-based system to enable energy-aware fine-grained offloading**. Compared to the conventional device-centric solutions, the task-centric Cloudlet actively traces and analyses the local task execution and perform fine-grain task-level scheduling. The offloading resources can be scheduled based on the priority of local task interest. This fine-grain task-level scheduling can efficiently improve the throughput of the resource-restricted Cloudlet and reduce resource preparation delay. To our knowledge, there is no existing work on the fine-grain Cloudlet-based resource management, which targets on the resource limitation and preparation overhead issues on the Cloudlet and considers the task-level energy-aware offloading collaborative scheduling on the Cloud. A task-centric

Cloudlet-based offloading system is proposed to optimize the task-centric offloading scheduling process. The system performs fine-grain resource preparation and execution, considering energy efficiency, execution efficiency, scalability, security, and availability issues. An energy-aware Cloudlet-caching offloading model is proposed to handle the task-level offloading execution on the virtualized Cloudlet and to optimize the Cloudlet processing throughput and mobile device offloading performance. A Cloud-based task-centric scheduling model is designed for handling the task-based offloading requests and further reduce the offloading energy cost of the remote Cloud resources.

- **An efficient and scalable proactive offloading model for sustainable Cloudlet resource allocation in heterogeneous mobile Cloud environments**. Compared to the conventional Cloudlet-based solutions, the proposed offloading model considers the heterogeneous offloading scenarios with different types of tasks, resources, and mixed offloading methods. It can coordinate with Cloud resources and perform partition-based offloading and migration-based offloading dynamically and simultaneously according to the task load and QoS. In addition, the system is integrated with fine-grain load prediction components to achieve proactive resource allocation, concerning the trend and seasonality of the task load. To our knowledge, there is no existing work on the Cloudlet-based mixed offloading issue, which targets the mixed offloading coordination and fine-grain load prediction. In this part, a queueing-based hybrid Cloudlet model is proposed that enables the heterogeneous offloading behaviors. The model concerns the diversity of application tasks, offloading resources, and offloading methods seeking to achieve fine-grain QoS-aware energy and execution efficiency. The energy-aware deadline-constraint heterogeneous offloading and resource allocation problem is formalized according to the above queueing model. A Particle Swarm Optimization (PSO) heuristic is proposed as the near-optimal solution. In addition, a SARIMA-based load prediction model is designed on the Cloudlet system to achieve proactive offloading resource allocation. The offloading algorithm can adjust the resources based on the trend, seasonality, and fluctuation of the recorded load history for each task.

- **A mobility-aware energy-efficient model for MEC-based offloading**. It concerns the intra- and inter-Cloudlet heterogeneous offloading process in a priority-based queueing system, aimed at optimizing energy and execution efficiency under time constraints while minimizing the number of offloading service rejections intro-

duced by UE mobility. It concerns the mobility-aware priority-based scheduling and multi-offloadee load balancing. In this part, a queueing-based network is proposed to model and analyze mobility-aware heterogeneous offloading in the MEC. The proposed model concerns the impacts of UE mobility, the heterogeneity of tasks, and the dynamics of the system. The intra-Cloudlet mobility- and energy-aware time-constraint heterogeneous offloading scheduling issue is formulated in the priority queueing model with different priority classes. A Particle Swarm heuristic is designed to achieve the optimal or near-optimal task priority scheduling and server utilization efficiency. The inter-Cloudlet resource selection procedure is modeled in a mobility-aware multi-site resource allocation model, which is further solved by lightweight dynamic load balancing.

## 1.4 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2, general background knowledge regarding the topic is introduced with the taxonomy and survey of the state-of-the-art. In Chapter 3, the Cloudlet-based multi-layered offloading prototype is presented. In Chapter 4, the task-centric mobile Cloud-based system is described to enable energy-aware fine-grained offloading. In Chapter 5, the efficient and scalable proactive offloading model is demonstrated for sustainable Cloudlet resource allocation in heterogeneous Mobile Cloud Environments. In Chapter 6, the mobility-aware energy-efficient is proposed for MEC-based Offloading. In the end, Chapter 7 concludes the thesis and directions for future research work are presented.

# Chapter 2

# Background and Literature Review

The constant demand for more sophisticated mobile devices has accentually propelled the industry to produce technology advancements and promote sharp improvements in the capabilities of mobile devices, as well as technology-support devices. Several reports, such as the ones issued by Cisco Systems [129, 194], have demonstrated the rapid growth in the hardware capacities of these devices in recent years. As an example of this steep growth, a recently released mobile phone can easily outperform many static personal computers in the past decade in terms of execution and communication efficiency. On the other hand, these same pocket-size devices integrate more advanced sensors and resource-hungry functions, as well as richer applications associated with such their functionalities and capabilities. Unfortunately, improvements of the battery life-span have not progressed as other parts; the battery of mobile devices is fundamentally limited to the hardware layer. For instance, Lithium-based battery technologies [91] that proposed ten years ago are still used nowadays while new graphene-based solutions [161] may unlikely launch on the market in the foreseeable future. As a result, several works have been putting significant efforts on the development of software-layer optimizations to support resource-hungry tasks so that they can execute on resource-constraint devices.

MCC rises in a convenient moment where it, in a simplistic viewpoint, intends to handle the conflict between mobile applications and mobile devices. MCC envisions the use of Cloud resources for the provisioning of external computing and storage utilities to mobile entities. When dealing with the provisioning and allocation of distributed resources, Grid computing [131] has been as a prominent solution and a mature standard for the dynamic control and management of processing and computing resources. The Grid middleware defined standardized internal and external Application Programming

Interface (API) and provided functions, such as security, resource management, information queue, and data management. Since the approach of Grid is proposed, many works [100, 87, 139, 201, 31] endeavor to augment the mobile device capability and to enable MCC offloading by leveraging Grid resources.

Even though studies have questioned the distinction between Grid and Cloud due to their similar goals [233, 16], as well as the maturity [132] and performance [113] of Cloud Computing, we must point out that the Cloud suits well to offloading because it enables customization, utilization, load balancing, scalability, and less energy consumption when compared with previous standards [239, 215]. Therefore, Cloud Computing can enable adequate service availability through its internal redundancy-based resource scheduling frameworks; Amazon [112] and Google [92] public Cloud infrastructures have already included this feature to support services. In addition to the high availability, Cloud Computing also grants elastic provisioning models [190]. These flexible service models can particularly suit well to requirements on different granularities by varying resource capabilities and management levels. As a result, when compared to other large-scale systems, such as Cluster or Grid computing systems, CC stands out by genuinely empowering real-time resource provisioning in a self-manageable pay-as-per-use manner [10, 88, 132].

MCC then bridges Mobile and Cloud Computing through effective offloading. Certainly, connectivity and availability of communication channels play a fundamental role in enabling this offloading as a promising alternative which outsources mobile application tasks or parts of application components from mobile devices to external Cloud utilities. Inevitably, together with offloading, a substantial amount of energy can be preserved on mobile devices by alleviating computing loads from mobile elements. However, preserving energy can only be achieved if the communication channel management overhead is observed and properly controlled. However, preserving energy can only be achieved if the communication channel management overhead is observed and properly controlled. Consequently, the life-span of the mobile devices can be prolonged without hurting their mobility and portability.

Unfortunately, offloading-based energy reservation is not necessarily costless. The offloading process inevitably includes several issues that must be considered for achieving improvement of performance or reduction of energy consumption. For instance, the process of migrating application load might add computation and communication overhead, extra energy consumption of the external resources, usability concerns on users and applications, or QoS requirements of applications and services. All these aspects influence,

or even impede, achieving energy efficiency improvement during the task offloading execution scenario. However, even truly maximizing energy reservation in mobile devices, the offloading comes to the expense of Cloud resources, missing, at some extent, the purpose of *Green Computing* [170]. As result, the scale of the urban environment where mobile devices most likely reside incurs a high volume of offloading requests, exposing and deepening issues involved with unattainable scheduling, imbalance, and management costs, as well as inevitably increasing Cloud-side energy consumption.

The high interest in MCC have motivated the development of many works, so several recent survey studies have collected and classified existing works, focusing on architectural aspects, QoS constraints, augmentation of mobile device capabilities, and decision-making policies. As for defining a general layout for MCC, some surveys have regarded the overall challenges and general architectures of MCC [209, 208, 200, 238]. Other surveys have observed MCC solutions that have targeted constraints on QoS [6]. Some other survey works have observed the Cloud-based augmentation of mobile elements [2], focused on computation and storage enhancement ("augmentation") through offloading [260], or they have concentrated on producing an understanding of proposed policies and procedures on the decision-making process within the offloading [255]. Other survey studies have centralized in the synergy of novel technologies with MCC, such as Edge Computing [223, 224], IoT [1], and Big Data Stream Mobile Computing [11], or they have focused on scheduling and load balancing in Datacenters towards energy efficiency [149]. Contrary to these existing surveys, this study focuses on the particular scope of sustainable computing; it provides an overview of the state-of-the-art on energy-awareness in the MCC-based offloading and related Green Cloud computing strategies. This survey thus classifies related works regarding the problematic models and strategies of preserving the energy of mobile devices while giving a Cloud-based perspective on energy preserving techniques.

As a result, this survey introduces an ample view over the elements that concern energy aspects towards the profiling of applications and resources, partitioning of applications tasks, and the actual decision-making procedures from the perspective of both mobile devices and Cloud systems with discussions on simulation-based experimental evaluations and implementation-based principles. This work also develops further on the actual open challenges in MCC, delineating the possible research directions in the area. As described in Figure 2.1, this study splits Sustainable MCC in two major groups: (i) one that concerns the works focused on promoting energy saving in mobile devices, represented by *Energy-aware MCC* in Figure 2.2, and (ii) another that concerns the related

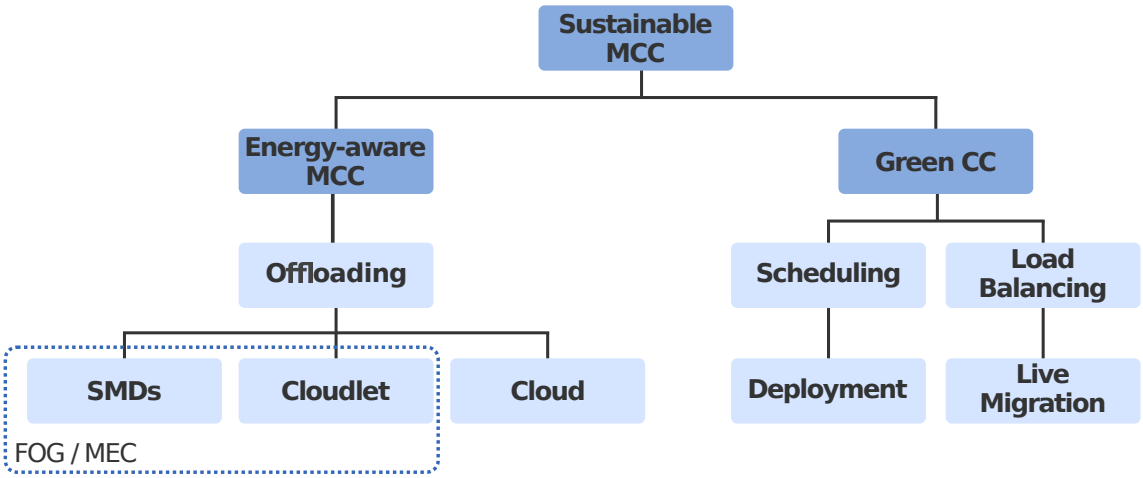



Figure 2.1: Sustainable Mobile Cloud Computing Interest Areas

approaches to contain energy consumption in Cloud resources, represented by *Green CC*.

The organization of the remaining sections is as follows. Section 2.1 introduces the concept of MC and MCC systems by detailing the components of their architectures and then illustrates the entire offloading process in the scope of energy-aware MCC systems. Section 2.2 describes the existing Mobile device based works regarding the issue of energy reservation and tradeoffs. Section 2.3 presents the revolution and solutions towards Cloud-based energy-aware resource management. Section 2.4 describes actual open challenges and presents future research directions. Finally, Section 2.5 briefly summarizes this survey study.

## 2.1 Background on Energy-aware Mobile Cloud Computing

Even though offloading of computation and data may be considered a complementary feature to mobile computing systems, it has proven to become an essential tool for increasing the autonomy of smart mobile devices and enabling the delivery of rich applications. A thorough understanding of mobile computing and MCC Systems enables an efficient design and implementation of offloading strategies, as well as awareness of involved limitations. Therefore, this section presents a brief background on MCC, including motivation, architecture, classification, and offloading basics.

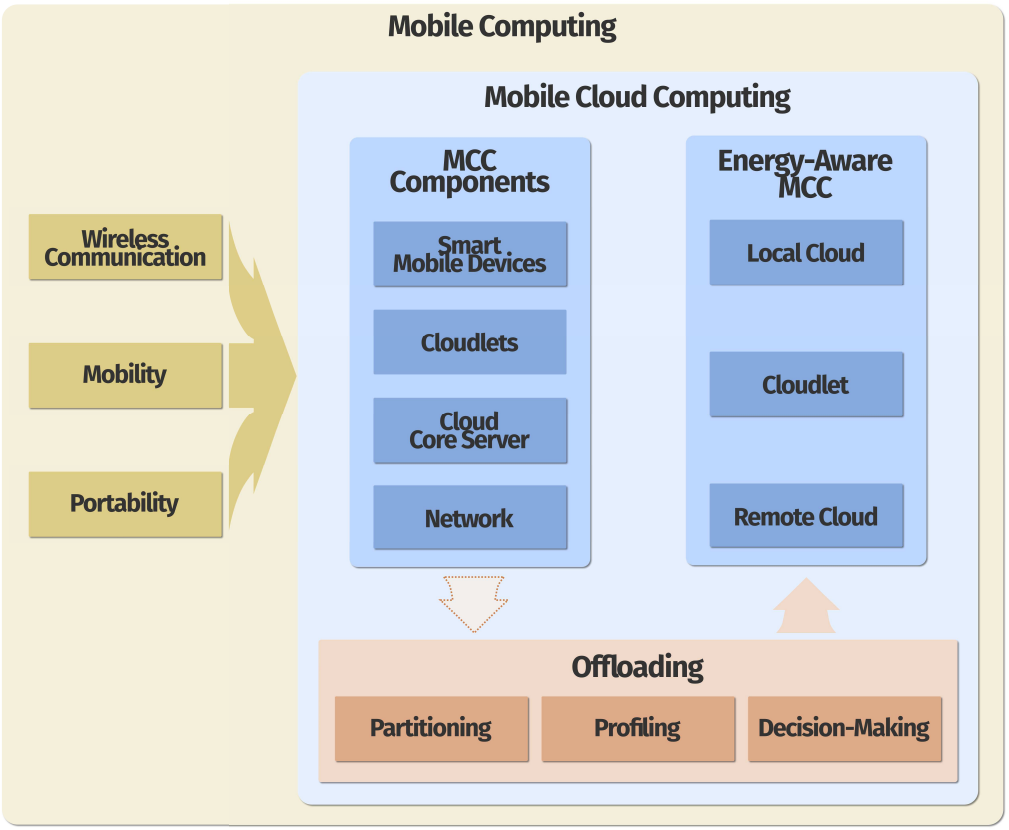



Figure 2.2: Energy-aware Mobile Cloud Computing in a Big Picture

### 2.1.1 Mobile Computing

Mobile Computing closely relates to Ubiquitous Computing, which initially originated from works presented in the early 1990's [241, 240]. These works were the first to delineate ubiquitous computing, describing scenarios that have become common-place nowadays; they also forecast the aggressive development of mobile hardware and introduced the perception of everyday computing, which represents the unstructured technology-aided activities of the daily routine of common individuals. Stemming from this concept, we can easily conclude that Mobile Computing comprehends a combination of wireless communication, mobility, and portability [130, 219].

These factors have been presented as the ones responsible for defining Mobile Computing in its early years. Even in recent years, we can observe that this representation still stands true for shaping challenges, motivations, and research works behind Mobile Cloud Computing. When it comes to Smart Mobile Devices (SMDs) and MCC, a large number of works, many listed in this survey study, showed the substantial importance

and interest in exploring energy consumption in mobile environments.

#### Wireless Communication

Wireless communication technology is fundamental for enabling mobile systems and permitting SMDs to have access to the Internet; on the other hand, it comprises the major concern in realizing such systems due to the already-existing physical challenges for maintaining connectivity and communication quality. In mobile systems, SMDs make use of wireless communication, typically via Wireless Local Area Network (WLAN) or Cellular network, to attach themselves to a stationary physical network.

#### Mobility

The free, independent movement of devices consists of the fundamental nature of Mobile Computing, which refers to support network availability independent of the location, as well as the physical displacement, of devices. As a vital requirement to enable mobility, SMDs usually need to follow protocols and conduct registration procedures whenever entering new environments and areas of interest. Many protocols, such as communication handoff, have been developed as a way to minimize communication interruptions and maintain connectivity as a much as possible.

#### Portability

In broad terms, portability is one of the critical factors in enabling mobility of devices. The non-attachment of devices to a stationary power supply amplifies their independent movement capabilities. Consequently, the use of batteries consists of the feature that allows mobile devices to be carried around, directly enabling, at the same restricting, mobility. The autonomy, lifespan, or charge of batteries immediately influences mobility, as well as capabilities of SMDs. Portability unconditionally suffers from the lack of continuous power source.

### 2.1.2 MCC System Components

Besides the three major features described in the previous section, heterogeneity is also an inherent factor in MCC. Apart from the large, and growing, a variety of services and applications, we can notice that the models, capacity, and behavior of SMDs, Cloud servers, and network infrastructure differ widely. However, the elements that compose

an MCC system, in terms of supporting offloading and device extensibility, are classified into major categories: smart mobile devices, Cloudlets, a remote Cloud, and networking. These elements play an essential role into MCC and must be considered when offloading computation and data from devices.

### Smart Mobile Devices

SMDs are the original motivators for designing and implementing MCC. In the whole architecture, they are the entities with enhanced capabilities through offloading techniques, and they are responsible for the original local task execution and data storage. As described later in this study, a local execution is preferred when conditions indicate that offloading to Cloudlets or remote Clouds cannot be performed efficiently, in terms of computation gain, data storage, and energy consumption. Such conditions are dynamic and dictated by evaluations obtained through profiling towards energy and computing efficiency, which includes task patterns, computing capabilities, and the networking environment.

### Cloudlets

Cloudlets are the unique, distinct components in the Cloudlet-based architectural model. These elements in the model perform as local resource managers, responsible for connecting local devices, executing offloading requests, and forwarding offloading tasks to remote Cloud resources in case the specific local Cloudlet capabilities are insufficient. Based on the granularity of offloading requests, Cloudlets are designed and implemented differently to cater device-level offloading or thread-level offloading.

### Remote Cloud

The remote Cloud primarily handles the offloading of task executions and data storage, which might be forwarded from Cloudlets or directly forwarded by mobile elements. Based on the definition presented in [190], a remote Cloud performs as a resource pool that contains clones of all mobile tasks. This resource pool can also provide sufficient "unlimited" computing capability to process different types of tasks.

#### Networking

Networking plays a crucial role in defining the feasibility of offloading in the MCC model. Mobile devices, Cloudlets, and Cloud resources present a considerable variability of capacities, features, and configurations. This diversity, when it comes to the variety of communication interfaces and means, promotes a diversity of connections among these elements.

### 2.1.3 Taxonomy of Energy-aware MCC

The great motivator behind Energy-aware MCC comprehends SMDs becoming increasingly more complex and powerful while recent batteries still being with low capacity. In order to achieve portability and availability for mobile devices, offloading has risen as the most effective approach, always seeking to transfer the tasks from mobile elements to externally accessible alternatives in the most efficient manner in the MCC architecture.

Based on the types of external offloading resources, the MCC offloading strategy can be briefly categorized into Local MCC, Remote Cloud MCC, and Cloudlet MCC respectively. As summarized in Table 2.1, the local MCC, which consists of only local mobile elements, leads to small overhead, but it is limited to its available computing capability. The remote MCC ensures sufficient computing power, but it introduces the long-distance communication that will lead to larger communication overhead. Subsequently, the Cloudlet MCC enhances both computing and communication efficiency, yet it is restricted by the availability.

Table 2.1: MCC Deployment Model and Energy-aware Issues

| **Deployment Type** | **Pros** | **Cons** |
|---|---|---|
| Local MCC | LAN Level Latency<br>Low Communication Overhead | Restricted Computing Resources<br>Mobile Device Hotspot |
| Remote Cloud MCC | *Unlimited* Computing Resource<br>High Computing Efficiency | WAN Level Latency<br>High Communication Overhead |
| Cloudlet-based MCC | Flexible Energy-aware Scheduling<br>Low Latency<br>High Computing Efficiency | High Scheduling Load<br>Cloudlet Bottleneck |

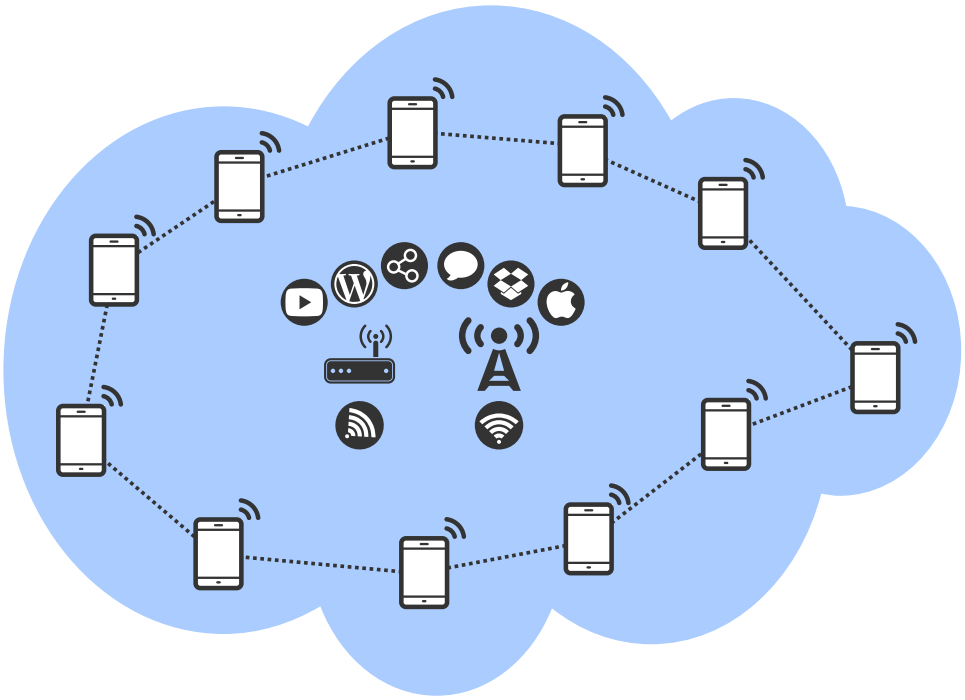


Figure 2.3: Energy-aware Local Offloading MCC Deployment Model

### Local MCC

As shown in Figure 2.3, in this deployment model, mobile devices in the local area initiate and maintain a wireless ad hoc network to form a Mobile Ad Hoc Cloud. SMDs can collaboratively help each other in ad hoc manner. Some designs have envisioned offloading and task delegation into Mobile Ad Hoc Clouds [209], following the most recent trends where MEC is implemented. This approach can drastically reduce communication latency but faces more constrained scenarios due to limited SMD capabilities. Within the network, idle mobile elements can provide the computing power for others. The devices can either trigger the offloading process as resource requesters or execute offloading tasks as resource vendors.

### Remote Cloud MCC

The deployment model is based on remote public Cloud resources as described in Figure 2.4, typically provided and maintained by public cloud service providers, such as Amazon AWS, Google Cloud Platform, and Microsoft Azure. In order to enable task offloading for mobile devices, virtual instances are established and mapped to mobile

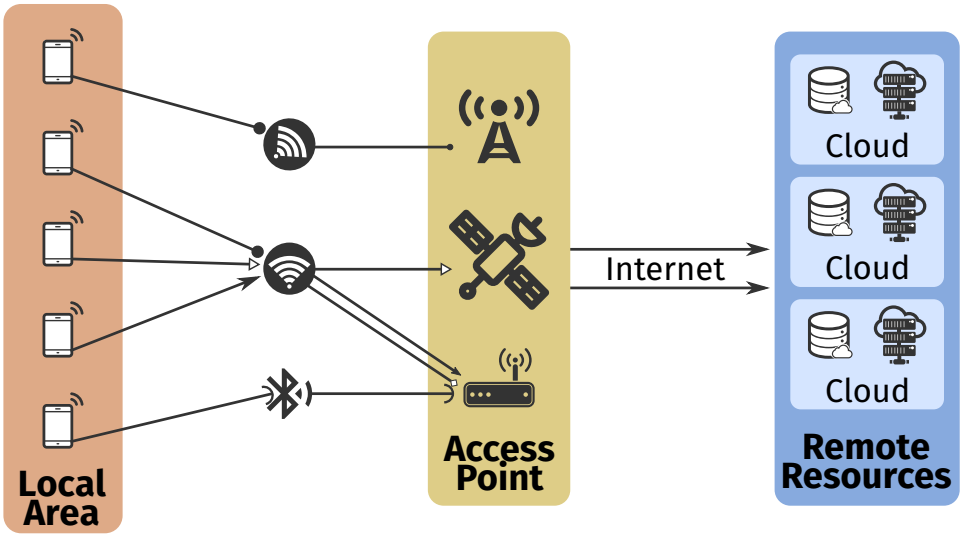



Figure 2.4: Energy-aware Remote Offloading MCC Deployment Model

users in an always-on mode. Each virtual instance hosts a virtual clone of the specific user device, capable of receiving offloading tasks, scheduling offloading executions and returning results via a proxy. Due to the data center based resource integration and energy optimization technologies, the model based on remote resources can support "unlimited" computing power and storage for the end mobile users and trivially outperform the local-based counterpart in terms of computing efficiency.

### Edge-based MCC

The concept of EC is motivated by MCC, which advocates deploying computing, storage, and communication resources in the vicinity of UE at the edge instead of remote CC. Provisioning CC services at the edge innovatively facilitates an intermediate layer between UE and CC, which promotes tasks to be processed locally; contents to be cached, prefetched and shared with the awareness of local context; raw data to be preprocessed and filtered before forwarding to CC via backhaul networks. Such capabilities, in turn, has fundamentally changed the way services are arranged, contents are delivered, and data is analyzed when comparing to MCC, leading to higher QoS and less backhaul network pressure.

**EC General Architecture** As shown in Figure 2.5, although edge technologies adapt different deployment strategies, the EC architecture (MEC, Cloudlet, and Fog) typically

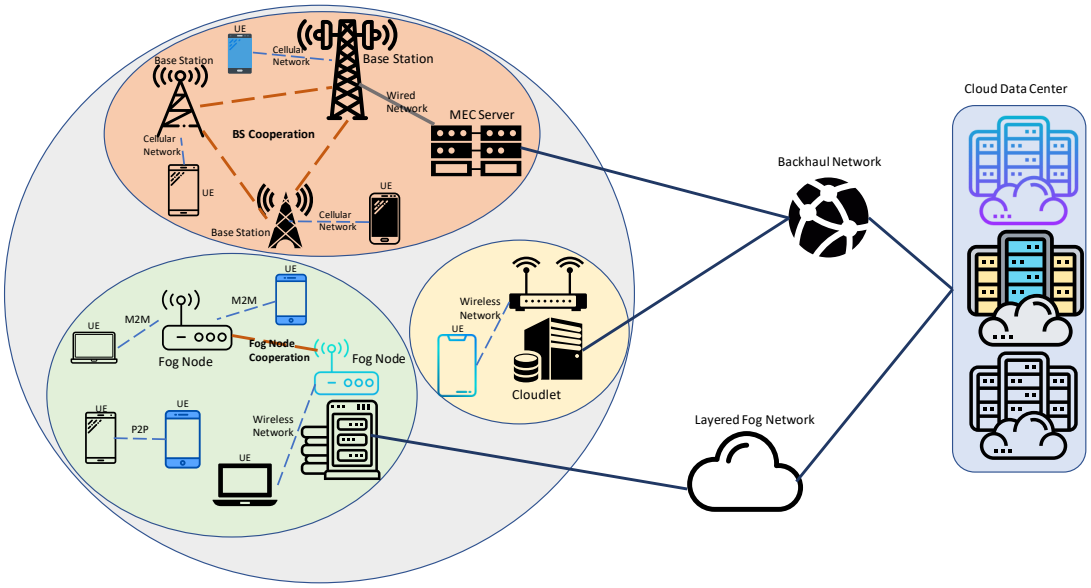



Figure 2.5: Edge Computing Architecture

involves three main hierarchy in common: end UE, middle EC platform and remote CC. UEs perform as service requesters that trigger computational tasks to be offloaded, contents to be shared or delivered. The initiation of service requests necessitates fundamental decision-making from UE regarding W&W – whether to use edge or not and which edge platform to connect, depending on factors such as network topology, UE energy situation, and task partitioning dependency. The EC platform bridges UE and remote CC, which sustains the capability to provide services locally and cooperate with geographically distributed EC platforms and centralized CC. The functioning of the EC platform accordingly involves two aspects – intra-edge service orchestration and inter-edge coordination. The remote CC adapts the same principles as defined in [190] to fulfill requirements forwarded from the edge.

**Cloudlet** The Cloudlet platform is shown in the yellow region in Figure 2.5, which performs as a box-size DC in the local domain. The main focus on Cloudlet is to provision Cloud services with crisp responsiveness and high quality. Concerning the communication overhead, especially the propagation and queueing delay, the Cloudlet server is generally deployed one hop away from UEs at the Internet edge, which adopts a star-type centralized topology to support high bandwidth wireless area network.

To alleviate the resource preparation delay introduced by virtualization, virtual ma-

chine overlay [220] is commonly utilized to synchronize soft data between UE and Cloudlet, which drastically reduces the cost on initializing base image. Another virtualization-related concern is virtual machine migration, which plays a crucial role in supporting seamless service provisioning. Unlike the remote CC that is connected with the back-haul network and maintains literally 'unlimited' resources, Cloudlet manages relatively restricted resources and signal coverage. In this case, migration provides a novel solution to balance the load between EC platforms and achieve UE mobility-awareness. Concerning the potential fails introduced by mobility, live synchronization replica schemes are generally adjusted to incrementally backup the soft overlay image so that interrupted services can continue if intermittent connections occur.

**Fog** The Fog nodes are depicted in the green area in Figure 2.5. The concept of Fog is first introduced by CISCO to address primarily IoT-related end-to-end services, which enables resources and services to be distributed horizontally between service requesters and service providers. Unlike the Cloudlet that relies on centralized servers, Fog nodes can alternatively cooperate and support D2D-based service provisioning in a distributed manner, which can substantially improve resource utilization efficiency and QoS.

However, the distributed network architectures require additional management costs on different layers of the system. In the UE layer, the incentive to share individual resources keeps the primary issue, which involves the design and implementation of pricing and auction schemes, and security mechanisms. Big data analysis on the UE behaviors can potentially discover possible resource providers and identify their corresponding trust level. In the Fog node layer, the cooperation of inter-Fogs relies highly on the awareness of the load and network context information among the Fog nodes, which demands real-time monitoring and proper prediction. Such system information can be further utilized to schedule the appropriate service provider and interrelated communication path via the network function virtualization. To reduce the amount of transmitted data among Fogs, caching and filtering protocols are commonly adapted to minimize the communication overhead for downlink and uplink.

**MEC** MEC was defined and maintained by ETSI and ISG as an emerging technology for 5G networks, shown in the red zone of Figure 2.5. The MEC platform specifically targets the edge service scenarios where service providers reside by cellular base stations (BS) to offer the cloud services directly via the radio access network (RAN). The MEC platform focuses on the open service provisioning issues at the edge, trying to integrate

third-party members to afford computation, storage, and networking capabilities on the platform level. Such accommodation drastically promotes local service enhancements and alleviates core network congestions by shifting resource-rich cloud services to the edge of RAN.

However, resource consolidation introduces new issues on different layers of the system. For the UEs, due to the availability of multiple networks, network selection draws increasing attention in the literature. The election strategies intensely depend on the profiling of dynamic information on the network, requiring timely responsiveness and satisfactory accuracy. Also, compared to the other wireless networks, RAN-based network typically consumes more energy during data transmission, making the energy reservation another impacting parameter in the network selection process. For the MEC servers, unlike the Fog or Cloudlet that utilizes existing infrastructures, these centralized servers are required to be first invested and configured at the edge, leading to the placement problem. The initialization of MEC servers involves the evaluation of multi-dimension variables such as price, coverage, and throughput to maximize the deployment profit. Due to the usage of RAN, MEC servers are sensitive to UE mobility. Several individual mobility models are adapted to estimate the contract time between UE and MEC servers, reducing the service failure rate. Besides the single mobility models, group mobility patterns are investigated to predict possible load balancing actions in the decision interval.

**Comparison** All of these technologies adapt hierarchical structures for resource provisioning but are initially motivated by different interests. Cloudlet is designed initially as a virtualized box that can support elastic resource provisioning similar to CC. The concept of Fog is first realized for IoT scenarios, which concerns the distributed resource sharing and horizontal processing in the multi-tie network systems. MEC is proposed with the development of the next-generation cellular network that targets RAN-based resource consolidation and open access issues.

In the remaining chapters, we use the term Cloudlet, as depicted in Figure 2.6 to refer to all MEC host counterparts in the edge environment, specifically targeting the resource allocation issue on the MEC host layer.

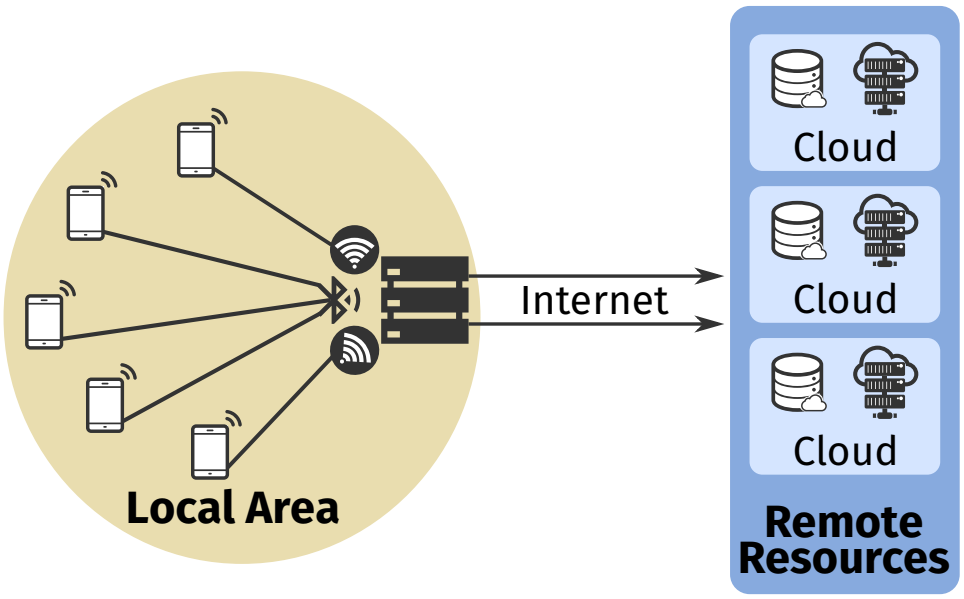



Figure 2.6: Energy-aware Cloudlet Offloading MCC Deployment Model

### 2.1.4 MCC-based Offloading

Understanding the major elements that compose the MCC environment and the essential targeted destinations for energy-aware offloading, we dissect the offloading process into three principal parts. First, in some general sense, offloading extends Remote Procedure Call (RPC) approach [27], which allows a program, or parts of a program, to run across one or multiple communication networks. Differently from the traditional RPC model, the MCC offloading primarily concentrates on maximizing energy reservation on the mobile elements, which further introduces additional issues, such as coordinating heterogeneous resources on the unstable network.

An entire MCC-based offloading process is suggested to be decoupled into three relatively independent sub-tasks – partitioning, environment profiling, and decision&execution respectively.

#### Task Partitioning

Offloading is performed at either the application level or device level based on the diversity of applications and the size of the system. With respect to the application requirements, tasks may involve specific-purpose equipment, which may be not available in the Cloud resources, such as GPS, camera, and other sensors. In this case, offloading cannot consider these mobile application tasks due to the lack of corresponding hardware

and capabilities in the Cloudlet or remote Cloud MCC solutions. Also, for security concerns, as shown in [226] and [262], applications with sensitive personal information may not be offloaded to a third-party Cloud as a matter of preventing data loss or inter-Cloud malicious attacks.

### Profiling

Because of the nature of mobility, the networking and computing environments may change immensely during application execution, under which the offloading efficiency may also vary largely during run-time. In order to decrease the probability that offloading overhead exceeds its benefits, during a certain interval, the execution environment information is necessarily updated as references to make the real-time offloading scheduling decision. The profiled information involves three aspects: the current task load, the computing status of mobile devices and offloading resources, and the networking between them.

### Decision-making & Execution

This step consists of the final part in the offloading procedure, which makes the distributed task scheduling decision from a global point of view. Although each task is already partitioned, tagged, and profiled, the entire task set needs a global evaluation to optimize the executions. This optimization consists of one the most extensively explored aspect in recent works, listed in the following sections.

- **Evaluation Model**

As this work will describe in the subsequent sections, which summarily enlists recent studies in Tables 2.2 and 2.3, an MCC-based offloading approach cannot suit all mobile applications; performance profoundly relies on the execution environments. However, all these works share similar offloading fundamentals: reduction of application execution time and reservation of energy can be achieved if the computational gain and the communication overhead are carefully considered and evaluated. We can then define, in general terms, energy consumption involved with local processing and data transfers and energy reservation, or gain, related to offloading.

Computational energy gain refers to the difference of computing capability between the mobile device and the offloading destination resource for the targeting task set. According to the Dynamic Voltage and Frequency Scaling (DVFS) model, as in Formula 2.1

$$Power_{dvfs}() = Power_{static} + Power_{dynamic} = c * f * V * V \tag{2.1}$$

where $Power_{static}$ stands for the leakage power consumption and $Power_{dynamic}$ represents the dynamic power consumption in the scope of minimum CPU frequency and maximum CPU frequency $[f(min), f(max)]$, the computational energy gain during the task execution interval $\Delta t$ can be defined as in Formula 2.2:

$$E_{\Delta t}(gain) = \int_0^{\Delta t} (Power_{dvfs}(offload) - Power_{dvfs}(mobile))d(\Delta t) \tag{2.2}$$

where $Power_{dvfs}(offload)$ is the offloading resource power matrix and $Power_{dvfs}(mobile)$ is the mobile device power matrix.

The calculation of communication overhead is even more complex due to the mix of different communication media and transmission protocols. Given a simple client/server based offloading application, The overall communication energy overhead from the perspective of the mobile device can be defined as in Formula 2.3:

$$E(cost) = E(upload) + E(download) + E(wait) + E(extra) \tag{2.3}$$

where $E(upload)$ and $E(download)$ calculates the energy utilized for uploading task inputs and downloading process results that are highly related to the size of the task and the real-time available bandwidth; $E(wait)$ is the extra energy consumed by the mobile device while waiting for the calculation results of the remote resource; E (extra) holds the extra energy cost of the mobile device when maintaining the communication channel.

- **Implementation Methodology**

Profiled information is typically abstracted as different types of parameters in mathematical models and the decision making process is to find the best value in these models. Even computation-only offloading scenarios involve several parameters and factors that substantially increase their modelling complexity. In more holistic approaches, energy, computation, and mobility factors are accounted. New designs require more sophisticated evaluation tools to implement and analyze different experiment scenarios to compare to other works. Throughout the extensive study in this survey, we could observe that the experimental-based comparison is typically oriented to two directions – simulation-based and implementation-based, both of which concentrates on the calculation of the task execution efficiency and device energy consumption.

In simulation-based studies, the evaluation typically deploys simulators to support the targeting experiment scenarios. The mobile simulators, such as OMNet++ [234] and ONE [160], are utilized to investigate local mobile devices and communications while the Cloud-based simulators [89, 164] handle Cloudlets and remote Cloud properties and their respective behaviors.

On the other hand, implementation-based evaluations lack standard MCC definition, adopting different implementing strategies and system architectures. In general, to enable mobile application execution externally involves the configuration of mobile devices, offloading resources and the channeling between them. SMDs are primary smartphones and tablets that execute Android OS [99, 15] or Window OS [108], which can utilize Java Virtual Machine (JVM) or .Net Common Language Runtime (CLR) to solve the cross-platform issue. The lack of iOS-based Apple devices is mainly due to the insufficient support of MacOS-based virtualization on the Cloud side. The offloading resources differ based on the granularity of the corresponding topics. For instance, Android-based image manipulation application scenario [15] is built on the Amazon public EC2 Cloud for considering and testing access to remote resources. A VM task overlay [220] is implemented on local servers in order to manipulated local resources. VirtualBox-based Android x86 image processing [166] is deployed on private and public Clouds for testing offloading approaches with hybrid resources. The communication between mobile devices and offloading resources are usually via proxies or middleware. In terms of proxy-based communication, application servers in the middle of mobile devices [166] have been used to dispatch application tasks to corresponding computing units. For the middleware-based solution, MAUI runtime is deployed inside smartphones and servers, which maintain the client proxy and server proxy internally [108]. In the context of communication, Kimberley Control Manager [220] has been adopted to provide service discovery and message transmission.

## 2.2 Energy-aware MCC Offloading

As discussed in the previous section, the mobile device based components concern the offloading issues in the scope of the local region, which involve the handling of task partitioning, profiling, offloading scheduling, and interactions among local components. The feasibility of MCC-based offloading must be first discussed to identify the offloading benefits. Considering that the offloading decision on partitioned tasks is well-established, the sub-offloading technologies and studies are then described further in details.

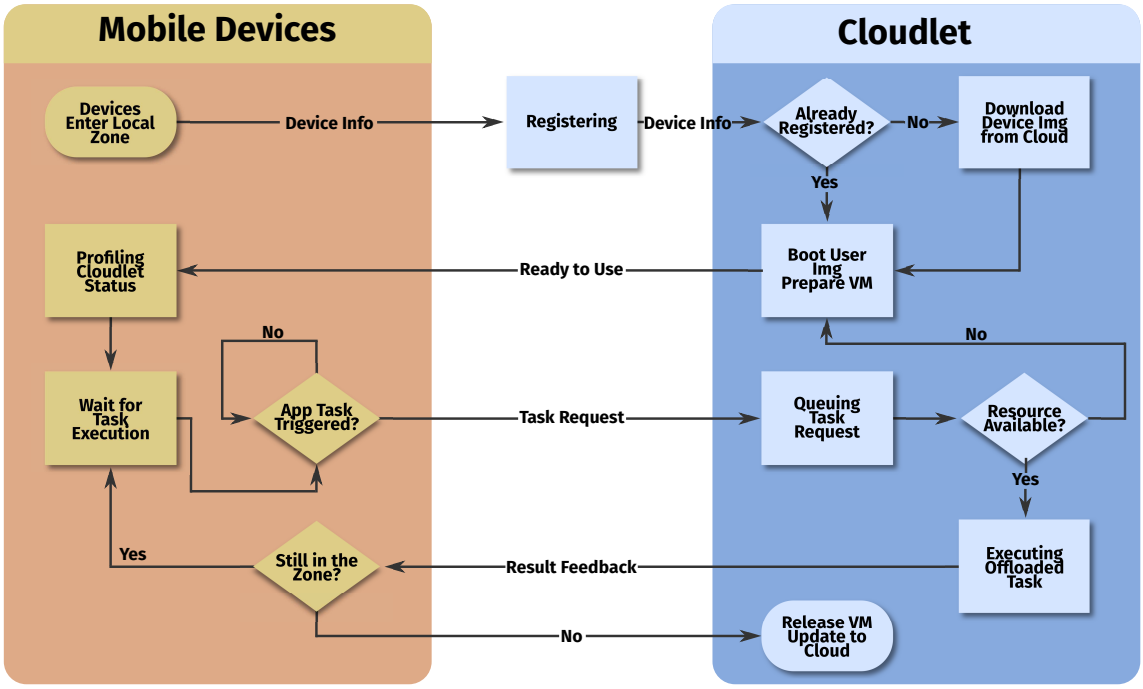



Figure 2.7: Energy-aware Local Offloading Flow

Figure 2.7 describes the general process of energy-aware task offloading. The typically local offloading process starts with the target devices enter the coverage zone. The devices first register themselves with the Cloudlet manager so that the Cloudlet can download user images from their remote repository if no local copies are available. After the virtual instances are fully booted, the Cloudlet acknowledges the device about the virtual instance information and QoS information. Based on these profiling data, the devices trigger offloading tasks or only execute the tasks on the mobile devices, based on the energy consumption evaluation. Finally, after the device leaves, the corresponding Cloudlet resources are released for further usage.

Figure 2.8 summarizes the existing, recent MCC works towards reducing the energy consumption of SMDs. The solutions are usually focused on a particular aspect of the MCC environment, such as Availability, Communication Costs, and Computation Efficiency, at the same time as minimizing energy consumption of mobile devices. Orthogonal to these objectives, different approaches have been used as an attempt to achieve the targeted minimization.

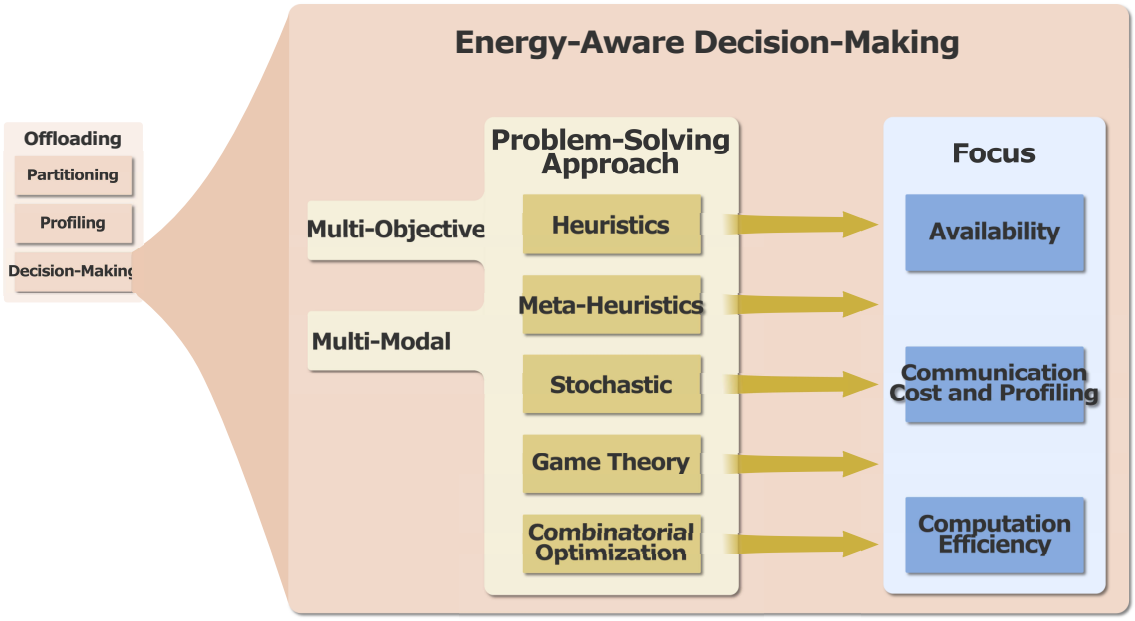



Figure 2.8: Offloading Objectives and Approaches for Reducing Energy in SMDs

### 2.2.1 Offloading Objectives

Enabling MCC-based energy-aware offloading procedure may not always be beneficial regarding energy reservation. Several previous works have evaluated the potential overhead that the energy-aware task offloading might introduce, in terms of computational waste and communicational overhead. On the one hand, regarding the communicational overhead, as a form of RPC, offloading introduces new communication, I/O, and synchronization process inevitably when it is compared to the device-only executions. On the other hand, the computational waste is generated due to the nature of the mobile applications, within which only a certain level of parallelism can only be achieved. As a result, enabling remote offloading implies on halting the execution process on the mobile device occasionally, which might significantly reduce resource utilization efficiency, especially for the high-end devices that are equipped with high-frequency computing units. As summarized in Tables 2.2 and 2.3, energy-aware MCC offloading approaches span over several goals. However, they, in general, aim at particular objectives when attempting to reduce or minimize, energy consumption of SMDs, considering the availability of cloud resources, communication costs involved with offloading, and computation gain.

Table 2.2: Summary of Energy-aware MCC Offloading Oriented Works (Device Level)

| **Reference** | **Granu** | **Comp** | **Comm** | **Task M** | **Issue** | **Solution** | **Avai** | **WiFi** | **Cell** |
|---|---|---|---|---|---|---|---|---|---|
| BECMC[15] | Device | N/A | Yes | Clone | Sync | Logger | Yes | Yes | Yes |
| DTMMC[140] | Device | Yes | Yes | VM | Scale | Migration | Yes | Yes | Yes |
| VMBC[220] | Device | Yes | Yes | AbiWord, GIMP | Preparation | Overlay | N/A | Yes | N/A |
| CSOSM[261] | Device | Yes | Yes | Math, Detection | Channel | TOPSIS | Yes | Yes | Yes |
| CDroid[14] | Device | N/A | Yes | Clone | Latency | Proxy | No | Yes | Yes |
| VD-ABC[115] | Device | N/A | Yes | Multi-task | Deployment | BCO | Yes | Yes | Yes |
| EnaCloud[172] | Device | N/A | Yes | Multi-task | Deployment | Opt. heuristics | Yes | Yes | Yes |
| CoopMCC[116] | Device | N/A | Yes | Multi-task | Cooperation | Power-splitting, contract | Yes | Yes | Yes |
| EA-HRM2[134] | Device | N/A | Yes | Stream | Heterogeneity | OHTA | Yes | Yes | Yes |
| DECM[136] | Device | Yes | Yes | Multi-task | Energy | Dyn. Prog. | N/A | Yes | Yes |
| TIM/EDA[159] | Device | Yes | N/A | Multi-task | Fairness | Auction | N/A | Yes | N/A |
| SynMCC[167] | Device | N/A | Yes | Stream | Consistency | Sync | Yes | Yes | Yes |
| MAYA[168] | Device | Yes | N/A | Multi-task | Cost reduct | ILP | N/A | N/A | N/A |
| LODCO[182] | Device | Yes | Yes | Multi-task | Exec Cost | CPU cycle | N/A | N/A | Yes |
| E2G[217] | Device | N/A | Yes | Stream | Proxy Select | SPNE | N/A | N/A | Yes |
| EENA[218] | Device | N/A | Yes | App | Brokering | Decision Matrix | Yes | N/A | Yes |
| EMOP[229] | Device | N/A | Yes | Stream | Multi-channel | DTMC, VIA | Yes | Yes | Yes |
| Off-MinED[237] | Device | N/A | Yes | Stream | Tradeoff | 0-1 ILP | Yes | Yes | Yes |
| MCCWET[250] | Device | Yes | Yes | Multi-task | Offloading/MPT | Convex Opt. | N/A | N/A | Yes |
| OF-MECO[251] | Device | N/A | Yes | Stream | OFDMA | Mix Int | N/A | Yes | Yes |
| EECO[254] | Device | Yes | Yes | Multi-task | Latency, 5G | Priorities | N/A | N/A | Yes |

### Availability

MCC heavily relies on the availability of Cloud services and resources due to its close dependency on mobile networks, such as WiFi and cellular communication. Besides being closely related to the existence of connectivity, availability might concern the selection of the most suitable communication channel, and Cloud resources for minimizing offloading costs.

In the sense of identifying most suitable Cloud resources, CloneCloud [99] has tackled the challenge in determining the most appropriate Cloud resource when migrating, offloading, applications and tasks to the Cloud. It also considered the "distance" of the resources from the position of SMDs, defining acceptable offloading decisions based on communication latencies. Thus, CloneCloud conducted a study that emphasized the

Table 2.3: Summary of Energy-aware MCC Offloading Oriented Works (Task Level)

| **Reference** | **Granu** | **Comp** | **Comm** | **Task M** | **Issue** | **Solution** | **Avai** | **WiFi** | **Cell** |
|---|---|---|---|---|---|---|---|---|---|
| CloneCloud[99] | Process | Yes | Yes | Search, Scan | Flexibility | Translator | N/A | Yes | Yes |
| ThinkAir[166] | Process | Yes | Yes | Scan, Puzzle | Execution | Parallelism | Yes | Yes | N/A |
| UMCC[186] | Process | Yes | Yes | App | MAC | Shannon form | Yes | Yes | Yes |
| PEDSA[248] | Process | Yes | Yes | Stream | Throughput | GA | N/A | Yes | Yes |
| COMET[143] | Thread | Yes | Yes | Read-Write | Multi-thread | Memory Sharing | Yes | Yes | Yes |
| CIMCA[127] | Task | N/A | Yes | IMCA | Delay | Routing | Yes | Yes | Yes |
| MAUI[108] | Task | Yes | Yes | Video, Game | Overhead | Adaptation | N/A | Yes | Yes |
| EACMA[3] | Task | Yes | Yes | Math function | Delay | Local Process | N/A | Yes | Yes |
| JACCR[13] | Task | Yes | Yes | Multi-task | Execution | Joint Optimize | N/A | N/A | Yes |
| MTCCO[180] | Task | Yes | Yes | DIA | Latency | Proxy, Encoding | N/A | N/A | Yes |
| EaCMD[114] | Task | Yes | Yes | Stream | Availability | Collaborative | Yes | Yes | Yes |
| ACOMC[128] | Task | Yes | Yes | GCM | Updating | Fuzzy Logic | Yes | Yes | N/A |
| EeSPC[256] | Task | Yes | Yes | Linear | Collaborate | LARAC | Yes | N/A | N/A |
| Odessa[207] | Task | Yes | Yes | Recognition | Adaptation | Greedy Profiling | Yes | Yes | N/A |
| MCC-GNEP[90] | Task | Yes | N/A | Multi-task | Multi-user | Game Theory | N/A | N/A | N/A |
| MECC[96] | Task | Yes | Yes | Multi-user | Multi-channel | Game Theory | Yes | Yes | Yes |
| DNN-MCC[123] | Task | Yes | Yes | DNN apps | Cooperation | Convolution partitioning | N/A | N/A | Yes |
| EA-HCM[135] | Task | Yes | Yes | Multi-task | Heterogeneity | Assignment | N/A | Yes | Yes |
| eDors[147] | Task | Yes | Yes | Multi-task | Energy | Min Max | N/A | Yes | Yes |
| EDTS[174] | Task | Yes | N/A | Multi-task | Dyn Sched | CPA | N/A | N/A | N/A |
| IDA[177] | Task | Yes | Yes | Multi-task | Decoupling | N-LIP | Yes | Yes | Yes |
| BCSA[178] | Task | Yes | Yes | Multi-task | Exec Cost | Selection | Yes | Yes | Yes |
| TaCO[146] | Task | Yes | Yes | Multi-task | Energy costs | Dependency | Yes | Yes | Yes |

availability regarding the new MCC offloading process.

In CloneCloud, similar to MAUI [108], which utilizes .Net framework [230], process level virtualization is also deployed to establish an intermediate layer which can translate tasks to suit different instruction sets. Thus, mobile devices and static servers can share the same source code. Then, application tasks are suggested to be partitioned automatically based on rules, to keep application developers from partitioning details.

VD-ABC [115] deals with a similar problem as CloneCloud. It determines the optimal deployment of the cloned virtual machine in the Cloud to reduce the communication distance between SMDs and cloned VMS to minimize energy consumption. The approach employs artificial bee colony optimization metaheuristic with Boltzmann selection polity to tackle the complexity of the problem when determining the best VM placement while acknowledging the offloading of several other SMDs. Consequently, VD-ABC must contemplate the feasibility of VM deployment in case the involved energy costs are substantially high and ultimately do not grant any benefit.

In a more restrictive perspective, EnaCloud [172] focused on the Cloud resources when performing offloading; it implemented a dynamic VM placement enhance energy efficiency in the Cloud side. By abstracting the VM placement as a bin packing problem, EnaCloud made use of heuristics to achieve a sub-optimal placement solution where it reduced energy consumption and overhead by adjusting resource demands. Focused on the placement of Cloud VMs, EnaCloud disregarded communication costs and latencies when determining the SLA-based placement strategy.

Even though there might be suitable resources for fully implementing MCC, the high volume of offloading requests may lead to a fast lack Cloud endpoints for receiving tasks and data. Within this context, DTMMC [140] has realized a cost analysis that concerned availability as the number of local users increases. This work first illustrates the lifespan of task offloading in the Cloud and then argues the inevitable task migration overhead brought by the unpredictable workload in such an offloading process. However, due to the lack of global-level information, hotspot servers may easily be over-committed, which further downgrades the task offloading performance.

TIM/EDA [159] has designed an auction-based offloading within a Cloud-based MCC architecture. Resources are valued differently in this approach, where it holistically consider computation gain, communication cost, link performance, and energy consumption. The auction model considers both Cloudlets and SMDs when offloading tasks. The model aims at promoting collaboration of SMDs and Cloudlets with incentive and better fairness when hosting offloaded tasks. TIM/EDA implements a strategy for evenly distributing load and increasing the availability of resources for further offloading of other SMDs.

Finally, availability is closely related to the connectivity between devices and offloading Cloud resources, which involves coping with the instability or suitability of communication channels and interfaces. When Cloudlet MCC model is contemplated, new communication opportunities might be available in the sense of additional communication channels, which are represented through different interfaces, and conditions,

which are reflected in the proximity of resources. EaCMD [114] concerned the availability of public WiFi and proposes a collaborative offloading strategy that introduces cellular networks as offloading channels. The optimal offloading channel can be selected via channel evaluation between WLAN and cellular network, which is performed by prefetching location-based caches. The maintenance and updates of the caches are guided by the mobility prediction. Based on the comparison and prediction, the collaborative offloading strategy can achieve 80 percent energy saving than the naive ones. In EeSPC [256], MCC collaboration is abstracted as a Markovian channel based DAG, which is further handled by the LARAC protocol.

MECC [96] explores the opportunities in multi-channel wireless communication. The approach considers the interference among multiple peers when attempting to access the communication medium for offloading. MECC makes use of game theory for implementing a distributed computation offloading algorithm, coping with the multi-channel wireless contention. During the execution, the algorithm analyzes and identifies the benefit of offloading tasks in light of the communication conditions where transmission rates lower than a threshold do not benefit any SMDs.

EECO [254] is similar to MECC [96]; it also leads with the multi-channel communication conditions and multi-access characteristics of 5G when offloading to the remote Cloud in a Mobile Edge Computing scenario. EECO defines an optimization model to minimize the whole system energy consumption and to satisfy the latency constraints of mobile applications. Thus, the model accounts for the costs of local computation and data transmission in terms of energy consumption. The model follows a three-stage scheme for energy-efficient computation offloading where it classifies communication channels and prioritize assignments to handle multi-user task offloading in 5G.

UMCC [186] places MCC in a much broader sense, Smart Cities, where it considers several Cloud topologies: Central Cloud, Cloudlet, and Distributed Mobile Cloud. The first objective of this approach consists of dealing with the heterogeneity of communication technologies. As a result, the model contains a utility function that considers application factors, such as latency, energy consumption, and bandwidth consumption. UMCC makes use of the Shannon Formula to solve this optimization problem where a weighted sum of offloading entities is implemented following different policies: greedy, cluster-based, and biased randomization.

### Communication Cost and Profiling

Due to the nature of the distributed MCC environment, energy-aware offloading cannot ignore communication characteristics involved with data transmissions. Availability closely relates to communication in terms of connectivity of SMDs over networks. Assuming that availability is already achieved, the cost of communication is a fundamental, delimiting factor for quantifying the actual benefit of MCC offloading.

The dynamic communication changes that might happen in mobile environments have served as a strong motivator to determine precise networking conditions as a decision-making parameter to promote MCC offloading. As a result, EACMA [3] analyzed communication overhead by evaluating the impacts of the communication distance and the number of intermediate hops. Similarly, this subject has been tackled in VMBC [220] and CIMCA [127], which suggested that WAN-level communication may compromise the usability of MCC offloading. The feasibility evaluation is also performed in terms of energy consumption and execution efficiency.

Bound to a narrower class of applications than in EACMA [3], BECMC [15] focused on determining communication overhead for dealing with data synchronization. It tested the task offloading performance of Android smartphones upon Amazon public EC2 Cloud platform [9]. In the proposed scenario, system level information and application code are tracked and monitored passively. Passive information is extracted through the *Intents* class while *Active* information is driven by the instantiation of the *Alarm* class, which periodically triggers data collection.

Odessa [207] also analyzed the communication costs for MCC offloading, so it concerned dynamic profiling and its benefits towards adaptation during real-time offloading. This approach proposed a greedy profiler that can periodically extract piggyback environmental stage data, which is further utilized to estimate the execution bottleneck and scheduling adaptation.

Similar to BECMC [15], MAUI [108] has evaluated the energy consumption on both the data transmission and the profiling overhead. For the communication overhead, it tests a typical code uploading scenario under which a small piece of the task is sent from mobile devices to remote resources via 3G or WiFi. The results presented two positive correlations – RTT (Round Trip Time) vs energy consumption and throughput vs energy consumption. Under a certain throughput value, 3G uploading may consume three to five times more energy than WiFi uploading. Specifically, the paper emphasizes the impacts of WiFi's Power Save Mode (PSM). The PSM is intended to reserve energy

by turning the devices into passive mode when no data is sent. However, the device state transmission cost from passive mode to active mode also introduces latency, which causes higher energy consumption for small transfers.

The restrictive communication conditions that constrained offloading in MCC models have been relaxed with Cloudlet MCC architectures. The shorter distance between SMDs and Cloudlet have allowed better offloading gains and energy reservation. CIMCA [127] has performed a transmission cost analysis study on the decision-making network, which has tested the offloading performance between mobile devices and Cloudlets in the scope of interactive offloading tasks. CIMCA achieves task offloading by one or multiple reachable Cloudlets that are peers to each other. The work also evaluates the task offloading performance based on three types of tasks with decentralized routing and centralized routing. CIMCA suggests that Cloudlet-based offloading can provide larger throughput and lower delay when compared to remote Cloud.

As a way to counter the costly communication latencies, VMBC [220] has explored the reduction of the amount of data involved with MCC offloading. VMBC targeted the long-distance transmission overhead with the VM preparation process. The output of this work suggests that the WAN level latency between mobile devices and Cloud resources may significantly restrict the usage of offloading applications even if sufficient bandwidth is provided. Concerning the reduction of VM state transmission overhead, the work proposed a VM overlay model to enable reduction of the size of the offloading image.

Correctly assessing the dynamic communication status of an MCC environment allows for a more successful offloading decision-making. Even though communication costs are not the primary objective in many offloading models, they have been recognized as critical for allowing overall execution and energy gains. EA-HRM2 [134] implemented a model that contemplates the communication costs for offloading tasks to the Cloud. In heterogeneous embedded environments, the access to the wireless communication interfaces may pose a high cost in terms of energy. Thus, the task assignment model of EA-HRM2, even though focused on computation gains, heavily relies on the energy costs involved with data transmissions.

Similar to EA-HRM2 [134], EMOP [229] developed a strategy to optimize offloading in terms of energy consumption considering the changing communication conditions within mobile environments. EMOP introduces an energy-efficient multisite offloading policy that uses Discrete Time Markov Chain (DTMC) to model fading wireless mobile channels and dynamicity of SMDs. The Markov Decision Process (MDP) based frame-

work formulates the multisite partitioning problem, which considers communication cost against the offloading gains concerning energy.

In the same context as both EA-HRM2 [134] and EMOP [229], eDors [147] dealt with conciliating two major dynamic offloading objectives: reducing execution time and energy consumption. The model focuses on the computation aspects of the tasks. However, to reduce energy consumption, it recognizes the communication costs for each offloaded task. Thus, eDors identifies the suboptimal tradeoff between the two objectives through Max Min heuristics where transmission power is controlled to favor offloading depending on dynamic communication conditions.

Comparable to eDors [147], MinED [237] studied the tradeoff between energy reservation and communication delay. In this case, energy reservation is understood as primarily promote through computation offloading. Thus, the model acknowledges that offloading promotes energy reservation by relieving local processing in SMDs. However, offloading incurs costly application delays due to inherent communication latencies. MinED then copes with this duality by formulating a joint optimization problem through 0-1 Integer Linear Programming (ILP).

Communication costs not only involve the surrounding networking environment of SMDs but also the characteristics of applications that are being offloaded. The frequency at which synchronization needs to be conducted for keeping offloaded tasks consistent dictates the communication overhead throughout the application lifespan. Consequently, SynMCC [167] introduced a model that acknowledges the data synchronization for maintaining the consistency in the duality between SMDs and their respective cloned Cloud VMs. Synchronization frequency is dictated by the amount of offloaded data and changes on the data. Thus, SynMCC implements application and tasks profiling in SMDs for the selective offloading decision-making. The decision-making attempts to reduce energy consumption through minimal data synchronization between SMDs and the Cloud.

Opportunism is another exploited factor in Smart City scenarios where there exist diverse network interfaces in SMDs that can extend connectivity and promote better conditions for MCC offloading. UMCC [186] considers heterogeneous urban environments for MCC. The framework focuses on the availability through the connectivity of SMDs over several Cloud topologies. However, besides maximizing connectivity, UMCC attempts to reduce the communication costs when there are many communication technologies. The framework achieves this cost minimization by using Shannon Formula to solve the optimization problem.

Sharing the same objective of reducing energy costs of communication as in UMCC [186],

E2G [217] designed a model for efficiently transmitting data to the Cloud in an MCC system. The policy behind this approach assumes that an SMD bridging the costly communication with the remote Cloud is the most effective strategy to reduce data transmission costs of neighboring SMDs. In order to promote fairness among SMDs and avoid any selfishness, the model is based on Sub-game Perfect Nash Equilibrium (SPNE). SPNE allows E2G to select an SMD responsible for bridging the communication and minimize the whole energy cost of transmissions.

The new trends in cellular networks have also enabled to explore the beneficial characteristics of 5G for enabling multi-channel, multi-access data transmissions with the purpose of boosting offloading. For instance, EECO [254] accounts for the costs of local computation and data transmission while acknowledging the multiaccess characteristics of 5G, observing Orthogonal FDMA. Besides concerning accessibility in 5G, the model minimizes the whole system's energy while satisfying latency constraints by considering the energy consumed in the macro and small base stations when offloading. The scheme behind the model presented a three-stage energy-efficient computation offloading through classification and priority assignment.

In the same 5G context of EECO [254], OF-MECO [251] introduced a model to maximize local communication resource allocation for Mobile Edge Computing scenarios. In a multiuser environment, the model explicitly deals with two Medium Access Control (MAC) protocols: TDMA and OFDMA. OF-MECO makes use of a threshold based convex optimization through priorities to cope with infinite and finite Cloud capacities. On the other hand, the model formulates a Mixed-integer problem to maximize channel gains against local computing energy consumption through Lagrange and Karush–Kuhn–Tucker (KKT) conditions.

Very close to EECO [254], EENA [218] implemented an energy-efficient and network-aware offloading algorithm where communication between SMDs and the Cloud is realized over 5G technology. The suggested model makes use of the 5G characteristics and contemplates the use of Cloudlets, or brokers, for connecting SMDS with the Cloud servers. EENA employs decision matrices with dynamic parameters for regulating offloading to minimize energy consumption and involved communication costs.

With technologies towards the practical implementation of Microwave Power Transfer (MPT), new possibilities can be explored to extend the battery lifespan of SMDs together with MCC strategies. MCCWET [250] focused on energy reservation through several strategies: controlling SMD's CPU cycles, harvesting energy through MPT, and selecting offloading. By assuming the full availability of MPT, the framework employs

Convex Optimization Theory to minimize mobile energy consumption by selectively opting for offloading or controlling local power. MCCWET makes use of the threshold to interleave MPT with offloading and guarantees transmissions with optimal data allocation for dynamic channels.

CoopMCC [116] expands MCCWET [250] by collaboratively allowing SMDs to help each other in an ad hoc fashion. CoopMCC explored the cooperation among SMDs to extend the lifespan of their batteries. The approach implemented a cooperative contract where it attempts to achieve the mutual benefit of two types of mobile devices: energy-poor and data-poor. CoopMCC relies on power-splitting policy to generate an optimization closed-form model where communication can be exchanged with energy. Energy-poor helps data-poor by relaying data, and data-poor helps and allows energy-poor to harvest its energy through power-transfers. The model identifies the optimal PS factor and transmission power as a form to stimulate cooperation and exchange and achieve an optimal cooperation contract.

**Computation Efficiency**

Extending the computation capabilities of SMDs consists of a primary MCC functionality where communication costs cannot be denied due to their inherent presence in any mobile environments. With the objective of enhancing the processing of applications and preserving SMDs' energy, offloading schemes have employed sub-partitioning of tasks, scheduling based on task dependencies, and Cloud resource "limitations".

Unlike the previous objectives that mainly concentrate on reducing overhead or coping with the availability of Cloud resources, some approaches sought to maximize the computing efficiency that MCC offloading can support. An offloading architecture has been introduced in ThinkAir [166], exploring the parallelism in the MCC-based offloading. In this proposed architecture, the application subtasks are partitioned with the ability to execute simultaneously. The offloading performance proves to be effective for computing-intensive tasks which involve little data communication during execution but large computational load.

Similar to ThinkAir [166], JACCR [13] evaluated offloading under which computation-oriented tasks are scheduled and mapped to many Cloud computing units. This pairing enables multiple virtual instances to serve different application tasks simultaneously, increasing the level of parallelism of offloaded tasks in Cloud virtual servers. The results show that the proposed MIMO (Multi Input Multi Output) offloading model can extensively reduce the task queueing compared to the device-only execution with the same

task sets.

Following the task sub-partitioning strategy and when it is possible, finer grained tasks can lead to more flexible offloading. BCSA [178] introduced an optimal selection algorithm that makes use of adaptive task partitioning and dynamic selective offloading strategies. The primary target of BCSA consists of minimizing execution costs where local energy consumption, execution time, and execution costs are the major parameters of its search. These parameters consider communication delay and costs, in terms of energy, involved with the offloading process.

In contrast to BCSA [178], DNN-MCC [123] explored the particular characteristics of Deep Neural Networks (DNN) for optimizing the performance of applications when offloading them to the Cloud. Though a mobile-Cloud collaborative approach, rich DNN-based applications, such as speech recognition and intelligent personal assistant (Siri), can be deployed more effectively in SMDs. The model exploits the partitioning of heavy DNN convolutions for effectively boosting execution and reducing energy consumption, relying on data exchanges.

Due to the popularity of streaming and gaming applications in the mobile devices, some works tries to augment the performance of these applications, such as MTCCO [180]. MTCCO evaluates the offloading performance in CDIA (Cloud-based Distributed Interactive Application) to traditional DIA (Distributed Interactive Application). Due to the limited CPU and GPU capabilities, the proposed scheme treats mobile devices as light-weight terminals only responsible for presenting streaming data. On the Cloud side, a mobile client is deployed to communicate with the application server on behalf of the corresponding mobile user, which forwards user commands to target proxies and routes streaming data back to the user's screen. In this case, mobile devices can leverage the gaming processing entirely to the Cloud, only in charge of uploading instructions and showing results.

A relevant work is also shown in PEDSA [248] for data stream tasks. Concerning the maximization of the application makespan and throughput, a GA-based algorithm is proposed for the stream task partitioning, which begins with random selection and ends with maximal computing efficiency or largest throughput. The evaluation presents that using the proposed task partitioning model can always outperform the all-mobile or all-cloud naive approaches on the graph generation application. However, the computational cost of the mobile device is not fully discussed. Based on the DVFS mobile device computing model, shorter task execution time may not necessarily lead to smaller energy consumption, due to the high computation load introduced by partitioning and

channeling.

Even though task partitioning is necessary for allowing offloading to Cloud servers and achieving a certain level of parallelism of applications, scheduling decision-making is fundamental for guaranteeing proper task execution to reach substantial computation gains. In that sense, CSOSM [261] introduced a client-server based MCC system, tackling the entire task partitioning, profiling, and the decision-making process. In CSOSM, the offloading scheduling process is abstracted as two aspects – offloading channel selection and resource mapping. The MCC system uses TOPSIS offloading decision-making algorithm, which is deployed to the multi-criteria wireless interface election while MIN-MIN heuristic is implemented on the Cloud components to process tasks. As the experiment shows, the proposed system can outperform ThinkAir [166], reducing execution time to two-thirds and preventing half of energy consumption.

In order to cope with the decision-making complexity, ACOMC [128] employed a fuzzy logic based code offloading engine on top of COMET virtualization platform [175]. In ACOMC, environment profiling information is translated into crisp sets. Based on the fuzzy threshold rules, these crisp sets further generate offloading decisions. Experimental results showed offloading improvement that led to an average 10 second execution time for message delivery.

COMET [143] enabled distributed shared memory between mobile devices and the Cloud resources via Dalvik virtualization [32]. Instead of PRC, applying DSM can support flexible thread level migration, which is presented to reach 15 times higher execution efficiency.

TaCO[146] introduced a DAG-based analysis over the offloading costs and benefits, to select a task to run in either a Cloudlet or a remote Cloud server. It implements a task-centric strategy so that a finer granularity offloading model is enabled. Also, lighter application cloned VMs are used instead of the traditional SMD cloned VMs. This scheme imposes higher overhead towards low density of SMDs but compensates in saturated offloading requests.

In smart city or urban computing, environments, the scale of the system adds multi-user scenarios where access introduces stress to resource sharing and application offloading. The challenge in such conditions consists of dealing with the selfishness of users and promoting fairness when allocating MCC resources for improving execution and reducing energy consumption. MCC-GNEP [90] explores "open offloading" in an unmanaged, multiuser scenario. The approach makes use of a game theory model using queuing theory to deal with the selfishness of users where the MCC system follows a Cloudlet ar-

chitecture. MCC-GNEP employs a distributed algorithm for the model's architecture to achieve an equilibrium as a way to address the multiuser offloading scenario. The model assumes that applications have been already partitioned, statically or dynamically, and methodologies have determined which tasks to offload.

Similar to MCC-GNEP [90], MECC [96] considers a MEC scenario where it acknowledges multi-user computation offloading in multi-channel wireless interference. Besides attempting to solve the availability of Cloud resources, the approach also uses game theory for distributed computation offloading algorithm to cope with the selfishness of users when offloading. Also, it extends the offloading model into a multi-channel wireless contention.

The distributed MCC environment inherently requires to deal with communication issues even when energy-area offloading approaches emphasize on computation efficiency. Even disregarding the interdependencies of tasks, offloading depends on the networking infrastructure for sending tasks to run and retrieve their output. In this regard, EA-HCM [135] focused on task assignment into heterogeneous Cloud cores. This approach extends on a previous work, EA-HRM2 [134], where communication costs are heavily considered for the offloading decision-making. EA-HCM implements a heterogeneous task assignment algorithm to reduce energy consumption involved with offloading but still considering wireless communication costs. Thus, this model solves an energy minimization problem observing heterogeneous computing and data preprocessing.

eDors [147] was listed in the previous section as an offloading strategy relying on controlling the transmission power of wireless channels to tackle high communication costs. eDors has the reduction of energy consumption and application execution time as its primary offloading objectives. Its model solves an energy-efficiency cost minimization problem in light of the task dependencies through a dynamic partition scheme over a DAG of task dependency. In the end, eDors implements a distributed algorithm based on computation offloading selection, clock frequency control, and transmission power allocation.

SynMCC [167] employs selective offloading decision-making to minimize the frequency in which data is synchronized between SMDs and their respective cloned Cloud VM. Even though this approach has been listed as oriented to communication costs, its core offloading strategy relies on controlling the computation gain through different scheduling policies: periodic, programmatic, event-based, and resource-intensive update.

As described previously, EMOP [229], although closely considering communication costs in its model, its major focus formulates on multisite partitioning. The partitioning

policy is built on a value iteration algorithm based on a state transition graph. A Markov Decision Process framework allows to estimate mobile wireless channels for offloading tasks to multiple Cloud servers.

Most of the offloading techniques assume that applications allow partitioning in subtasks and certain level dependency on the local SMD resources. When we observe practical examples of mobile applications, we notice that they present time constraints for completing operations and returning outputs. Thereof, IDA [177] has dealt with applications' time constraints in a multi-device scenario when offloading tasks. The model attempts to reduce the overall energy consumption of several devices in a Mobile Cloud where tasks are offloaded considering the involved cellular communication costs. The task scheduling of heterogeneous applications is mapped as a nonlinear integer programming problem where application time deadlines and transmission error rates are the major problem constraints. IDA, interactive decoupling algorithm is then used to combine two methods, linear relaxation and convex optimization, for reaching a scheduling solution.

In a similar context as IDA [177], EDTS [174] designed a dynamic task scheduling algorithm to minimize energy consumption of mobile applications in light of constraints related to time and application probability. The algorithm makes use of schemes to reduce energy costs related to local processing and offloading by identifying critical paths in dependency task graphs. A sub-optimal scheme solves the scheduling problem by using Data Flow Graph Critical Path (DFGCP) and an optimal solution that addresses the Critical Path Assignment (CPA) with dynamic programming.

All described offloading techniques deal with a significant level of search complexity to achieve plausible suboptimal energy and execution gains. For the sake of simplicity, local offloading decision-making is sought as an alternative to solve the energy and execution issues of SMDs in their local scope by assuming that Cloud resources are optimally available and suitable. LODCO [182] made use of a low-complexity on-line algorithm that implemented decision-making depending only on the local mobile device's information and execution status, such as consumed CPU-cycles and needed transmission power for offloading. The model minimizes the execution cost of mobile applications in the context of Mobile Edge Computing environments by using Lyapunov optimization-based dynamic computation offloading.

MAYA [168] dealt with a minimization of energy consumption and offloading costs in MCC environments. The model assumed that energy consumption was reduced through the execution of tasks in remote Cloud servers while the execution of the same offloaded tasks is posed as monetary service costs. The models had as a major parameter the

classification of SMDs according to their capacity to afford homogeneous Cloud services. The energy and cost minimization is mapped as an Integer Linear Programming optimization problem to determine the best suitable task scheduling in a three-tier Cloud model.

### 2.2.2 Problem-Solving Approaches

From the enlisting and description of offloading techniques in previous sections, we can clearly notice that, orthogonally to them, many approaches have been used to solve the optimization issues involved with minimizing energy consumption, maximizing execution efficiency, and minimizing energy-related communication costs. These optimization constraints may be understood as disjoint and requiring tradeoff thresholds or policies to determine the precise "offloading level" of an application before it might start harming either computation efficiency or energy reservation. Heuristics, meta-heuristics, Stochastic Optimization, Game Theory, and Combinatorial Optimization have been used extensively in order for these offloading strategies to implement performance of such complex offloading scenarios.

#### Heuristics

As illustrated by the offloading approaches described in the previous sections, heuristics allows for solving scheduling and placement problems faster when compared to traditional, formal closed-form models. Usually, heuristics leads to approximate solutions that might be very close to the optimal solution of a problem, approximating the exact solution. For searching algorithms, the heuristic function allows to rank alternatives while the search is conducted, performing as a fundamental element when defining the convergence of the search.

- *Convex Programming* allows for minimizing convex functions, assuming that the local minimum is the global minimum in general case of the problem to be solved. In the context of MCC offloading, MCCWET [250] has used convex optimization to minimize the mobile energy consumption.

- *Integer Linear Programming* (ILP) "simplifies" the problem by assuming some or all variables to be integers when solving an optimization problem. Usually, the ILP objective function and constraints are linear. For solving an ILP problem, Linear Programming relaxation can be applied to relax an NP-hard problem into

a polynomial solution; however, the relaxation may involve violating constraints, not be possible, or not be optimal. In the context of offloading, MAYA [168] use integer linear programming to minimize energy and Cloud service cost when requesting Cloud servers to offload SMDs. MinED [237] also formulated a joint optimization problem by using 0-1 integer linear programming to minimize energy consumption and communication delay.

- *Nonlinear Programming* allows to optimize a nonlinear, or non-convex, problem by defining an objective function that works over a set of constraints, system equalities and inequalities, and a set of unknown real variables. The method either maximizes or minimizes the objective function, the optimized targeted objectives. IDA [177] has employed Nonlinear Programming to optimize offloading in light of the cellular communication costs.
- *Dynamic Programming* recursively breaks down a complex problem into simpler sub-problems. Finding the optimal solution for the original problem consists of identifying and/or joining the optimal solution in the subproblems. EDTS [174] has used Dynamic Programming to identify the critical path in task dependency graph and determine an optimal scheduling solution, considering time and application probability constraints.

#### Meta-Heuristics

Different from heuristics, which are designed to deal with the specific class of problems, meta-heuristics represent a set of "general-purpose" heuristics that can be adapted, or applied, to any class of scheduling optimization problem. Evolutionary algorithms have been extensively used to search for optimization solutions. VD-ABC [115], for instance, have used Artificial Bee Colony (ABC) algorithm with Boltzmann selection policy to determine VM deployments as a way to minimize the communication cost with SMDs in MCC.

#### Stochastic Optimization

Stochastic optimization is broadly used to represent the random behavior of mobility and transmission uncertainty in wireless communication channels. With a random probability distribution, a stochastic model allows estimating communication conditions when they cannot be predicted accurately. EMOP [229] has employed Discrete Time Markov Chain to model wireless mobile channels.

#### Game Theory

Game Theory allows for solving conflict and cooperation problems. In the context o MCC, the cooperation of selfless users may lead to fairer split and access to Cloud resources, achieving overall higher energy preservation through offloading. MCC-GNEP [90] uses Queueing Theory to address a multiuser, unmanaged scenario and achieve an equilibrium where open offloading is implemented. MECC [96] also assumes a multiuser scenario and employs Game Theory for the distributed computation offloading algorithm that is then extended into a multi-channel wireless contention environment. E2G [217] defines a model based on SPNE to select an SMD to communicate with the Cloud.

### 2.2.3 Critical Analysis of the Offloading Techniques

The key explorations of recent MCC offloading works are summarized in Table 2.4. The open issues and potentials can be categorized based on three main offloading functionalities – task partitioning, profiling, and decision-making – as follows:

Table 2.4: Open Issues

| **Process** | **Issue** | **Description** |
|---|---|---|
| Task Partitioning | Compatibility | CPU instruction sets, computing capacity |
| | Usability | How to migrate traditional applications to MCC |
| Profiling | Accuracy | Run-time environment may change drastically |
| | Overhead | Frequent and accurate profiling may exhaust battery |
| | Fast Response | Needs to be simple and quick |
| Decision Making and Execution | Complexity | Quick optimization with estimated inputs |

1. Task Partitioning

   (a) Compatibility. Hardware on mobile devices and Cloud resources differs in terms of architecture and instruction sets. A typical mobile ARM-based CPU [133] utilizes RISC instructions while the newest server side x86-based [86] Intel Skylake processor is based on MMX, AES-NI, CLMUL, and FMA3 instruction sets. This difference hinders servers to directly process applications that are designed for mobile devices.

   (b) Usability. In the layered MCC architecture, application developers are supposed to concentrate on the issues only related to applications. Unfortunately,

the standard MCC deployment model is not officially defined. As a result, application designers are typically required to estimate the task offloading performance and partition tasks for mobile devices and Cloud resources entirely via supportive APIs or partially through the automatic partitioning tools. A significant amount of effort on reworking might be inevitable to enable traditional applications to execute on the MCC execution flow.

2. Profiling

   (a) Accuracy. Unfortunately, the profiled data is not 100% correct even if the most precise instruments are equipped. This situation occurs due to three main reasons. First of all, the profiled data, such as mobile device current battery volume, CPU frequency, and available network bandwidth, are all continuous values. During each profiling interval, these values are detected and assumed to be the average value of this particular interval. A small range can help these discrete numbers converge to the actual ones, but continual profiling potentially increases the computing overhead and similar energy cost, which in return reduces the entire offloading profits. Second, the profiled data needs to be transferred to the processing unit, so a delay exists between processing time and profiling time.

   (b) Overhead. Unlike the task partitioning that can be processed prior to the real application execution, profiling is typically performed on the fly while the application is triggered to ensure the accuracy. In this case, the overhead of profiling exercises more influence on the offloading efficiency than the task partitioning, which requires the extra occupied computing and storage to be minimized.

   (c) Fast Response. Multiple objects are required to be profiled quickly because these results are the input values for the next stage although the profiling involves different detecting models and parameters. For the profiling of task load, a commonly used energy evaluation model consists of DVFS in [98] and [205].

3. Decision Making and Execution

   (a) Complexity. Decision making is the core and the most complex part for offloading. As discussed in the profiling step of the process, all profiled data are

"estimated" correctly, which cannot achieve full correctness. Based on only these inputs, the decision process is required to produce final global optimization to trigger the real execution. Furthermore, concerning the computing capability and execution overhead on Cloudlets or mobile devices, the optimization is supposed to consume negligible load and time.

**Task Partitioning**

Task partitioning needs to handle the diversity of hardware and software. Hardware Layer typically issue refers to the CPU architecture sets; this issue requires the source code to be translated into different machine executable bits accordingly. The Software Layer problem comprises enabling developers to migrate traditional mobile applications to MCC applications with as little effort as possible.

To tackle the usability issues, MAUI [108] seeks to minimize the effort of developers when designing offload-able applications by introducing annotations. The proposed architecture encourages developers to simply tag all classes as remote-executable if they are not related to mobile-specific components and interfaces. In terms of compatibility, MAUI employs a deployment strategy to coordinate ARM-based mobile CPU and x86-based server CPU [214]. In this case, the intermediate language can be automatically translated into different machine languages respectively, and the same source code can be executed on different platforms.

CloneCloud [99] utilizes Dalvik virtual machines [154] integrated with Java compiler front-end to handle the compatibility. In addition, the task partitioning components can smartly tag the tasks as remote-executable or local-executable on the class or method level, without developer's efforts. In this case, traditional mobile device based Java applications can operate seamlessly on the Mobile Cloud environment. The self-managed task partitioning limits the scope of remote tasks due to mobile-specific features and the device-VM one-on-one mapping.

Instead of implementing virtualization in application-level, ThinkAir [166] has suggested to deploy Android-based x86 virtual machine [153] directly on the Cloud. This remote deployment may introduce larger overhead while it provides higher authority that can further enable parallel task execution. Instead of automatic partitioning, ThinkAir also requires manual annotation via APIs which may further support user-defined parallel algorithms.

In other instance, VMBC [220] also argues to utilize hardware-level virtual machines to handle the compatibility. This approach advocates that virtual instances can support

a broader range of application. Also, they are not limited by the specific languages that applications are coded. In addition, the establishment and recycling of virtual instances are already well managed on the Cloud, which can directly support MCC applications in the same way as traditional software.

On the one hand, hardware-layer solutions outperform process-level ones in terms of flexibility and security that virtual isolation brings. On the other hand, the virtual instances generally involve larger computing power waste. Currently, standard sample base images are still not available from public Cloud service providers. It requires smartphone designers and mobile OS developers to coordinate, define and tailor base offloading images for different phones and tasks.

**Profiling**

The data profiling refers to computing capacity profiling and networking profiling respectively, the results of which are the main references of the final offloading decision.

- Computing Profiling. The computing capacity profiling involves mobile CPU profiling and server computing units profiling, both of which are performed on the DVFS energy management model. DVFS, as shown in the study described in [204], can reserve energy through delaying task execution. The model changes the voltage during run-time, and the CPU frequency can scale up and down based on the task requirements. In this case, achieving energy efficiency may not necessarily mean poor support for real-time applications. As proposed in [205], instead of calculating average execution time, the profiling model concerns the worst case boundary and using look-ahead to reduce energy cost for embedded devices further.

- Networking Profiling. Unlike the computing units, the network environment, especially the local wireless LAN, is not stable. Although network parameters, such as bandwidth and RTT, can be pre-detected before offloading, they may change drastically during execution.

In CloneCloud [99], the profiling data are simplified as CPU availability, screen display, and network availability. Since it assumes a one-on-one task VM mapping and ignores the energy consumption of the Cloud resources, the proposed profiling model can detect these parameters easily and minimize the overhead that the profiling process brings.

MAUI [108] classifies profiling behavior into device profiling, program profiling, and network profiling respectively. The device profiling concentrates on the CPU utilization and pre-defines a linear relationship between CPU usage and energy consumption for the specific mobile device based on multiple experiments. In terms of network profiling, MAUI requires basic network parameters including RTT, bandwidth, and package loss rate. Similar to the device profiling and program profiling, MAUI also deploys the test-collect method to fetch these values.

ThinkAir [166] also organizes the profiling process in three sub-domains – task, hard devices, and network. The hard device profiling is straightforward, which reads the CPU usage and screen and records WiFi and 3G sensor with two states. The task profiling in ThinkAir directly utilizes standard Android APIs to document the task related information such as execution time, required resources, uploading task size and others, specifically tailored for Android phones and pads. For the network characteristics, it requires data regarding RTT, bandwidth, and send/receive rates that are gathered by test packets.

As it can be noticed in these profiling methods, the core concern consists of the overhead that profiling itself may bring. The profiling process not only needs to occupy at least one thread to read the APIs or pre-stored data but also consumes energy. The solution usually comprehends to pre-fetch test data in advance and simplify the execution time data collection.

### Decision Making & Execution

Based on the profiled data, the decision-making process seeks to produce a global optimization for the entire task offloading. The problem is generally mapped to the network path forwarding issue. Given a task set that involves $N$ offload-able tasks and each task is linked to some others based on the reliant relationship, the task execution scenario can be further presented as from the first one to the $N$ task, via many possible paths. In this case, the profiled coefficient can be added into the links between tasks and the decision goal is to find the path with the lowest weighted values.

Since both MAUI [108] and CloneCloud [99] assume a centralized execution, they suggest using Shortest Path algorithms that only runs in $O(Nlog(N))$ where $N$ is the number of sub-tasks, in a typical mobile application that the number of edges is highly limited. Although the decision-making overhead is controlled to a certain level, it depreciates repeated scheduling and suggests to use previous results unless the profiling results changed drastically.

The issue on the decision-making drastically increases in complexity when distributed offloading is enabled. In ThinkAir [166], the program profiler tags the tasks that can be executed simultaneously and calls the execution controller to arrange extra VMs to parallelize the labeled tasks. Due to the complexity of profiling and decision-making, only recursive algorithms and large-scale data analysis are supported in this mode.

## 2.3 Sustainable Energy-aware Cloud Scheduling

In the scope of MCC offloading, the "Cloud" generally refers to all external utilities that are distributed outside the mobile devices. According to the mobility of the utility, the outside resources can be classified as static resources and mobile resources. In terms of the static utility, these raw resources are typically organized and clustered in the form of Grid Computing or Cloud Computing. The Grid and Cloud architecture encapsulate underlying resource management details and virtualize resource pooling services that can be easily accessed via the Internet. In terms of the mobile utility, the resources are organized based on the wireless ad-hoc network, which dynamically maintains the offloading services according to the local device connectivity; the studies of which are discussed in the previous mobile device based section. In this section, the state-of-the-art of Cloud service provisioning is investigated regarding energy-aware MCC offloading, based on the static utility.

The concept of Cloud computing, as well as the Grid computing, is proposed for layering traditional computing capability and resources into the raw components, middleware, and applications. The advancement of applications and distributed systems has leveraged the demand for computation and communication resources, as well as simple, flexible, and transparent access to services, which have been made available through a Web interface in most cases. This growing demand for resources has boosted the development of solutions that push towards paradigm shifts in the method that resources are provisioned and managed.

As shown in Figure 2.9, the general "Cloud" system typically involves three layers – the soft layer SaaS, the intermediate layer PaaS, and the hard-layer IaaS, which are responsible for task mapping, virtual machine management, and infrastructure deployment respectively.

The offloading approaches described in the previous section, such as MAUI [108], CloneCloud [99], PEDSA [248], EaCMD [114], and Odessa [207], presented practical MCC use cases that involved data streaming, pattern recognition, and gaming applica-

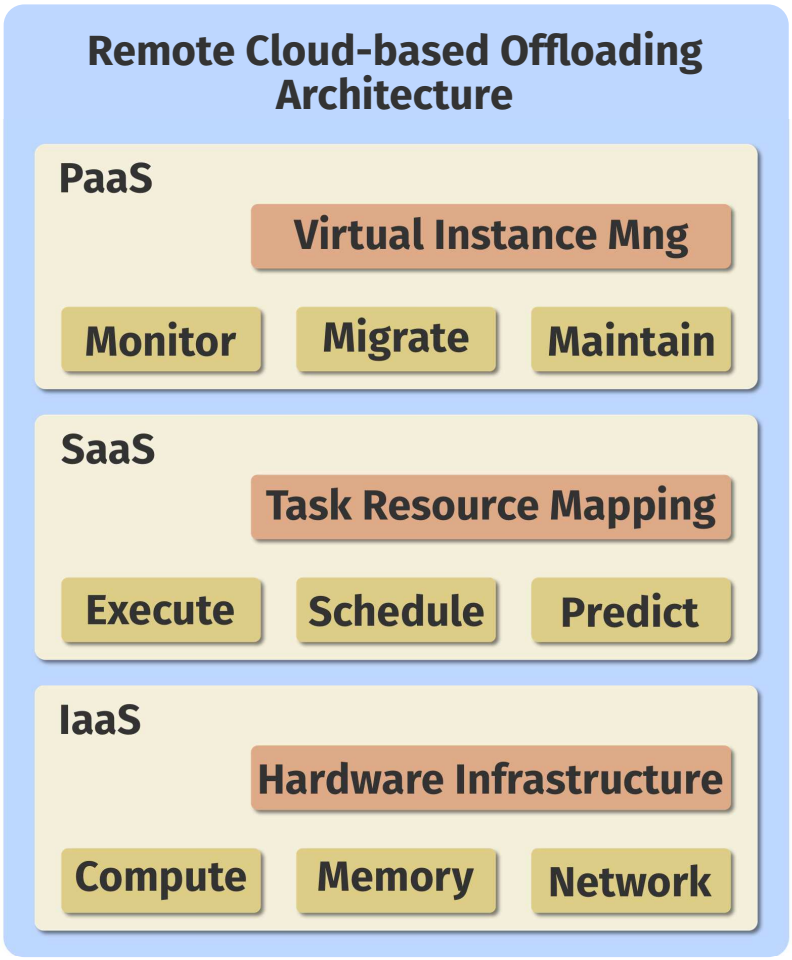



Figure 2.9: Cloud Offloading Supportive Model

tions in local-range and remote offloading. The elastic resource provisioning in Cloud-based architectures allows for more flexibly tailoring such relatively light-weight computing applications according to real-time load requirements.

According to the layered service provisioning model defined by NIST in [190], studies on energy-aware scheduling in Cloud Computing can be briefly divided into three directions: the handling of tasks and processes; the management of virtual resources; and the optimization of hardware resources. For each layered model, the energy-aware offloading studies are further categorized based on the Cloud resource functionality model – computing-centric model or communication-centric model, targeting the trade-off between execution efficiency and energy reservation.

The type of raw resources consists of a conditioning factor in classifying the hardware resource optimization issues, which are divided into computing resource optimization,

storage resource optimization, and networking optimization respectively.

### 2.3.1 Hardware Supportive Infrastructure

The hardware-layer studies focus on the virtualization technologies, which are led by Xen, VMWare, KVM, and AWS.

In both Xen_Comp [16] and Xen_Net [191], the development of Xen-based virtualization technologies are demonstrated, and its performance is measured. In the context of performance, Xen_Comp [16] presents the concept of paravirtualization, which alleviates memory, and CPU overheads by migrating partial tasks from virtual instances to hosts. Concerning the reduction of CPU overhead, Xen allows guest system calls accessed by processors precisely without bothering ring 0 Virtual Machine Monitor (VMM). In terms of networking, Xen_Net [191] introduces the high-level virtual network interface component which can migrate partial network processing from the guest domain to the physical NIC (Network Interface Card).

Similar to Xen, the KVM (Kernel-based Virtual Machine) presented by KVM_Sys [163] also caches virtual memory management units in the guests to improve computing efficiency. As for the storage devices and network I/O, programmed I/O and interrupts are utilized to enabling and enhancing the communication efficiency. As for VMware-based VMM, VM_I/O [227] and VM_Comp [235] explore the concept of paravirtualization further by attempting to minimize intermediate operations caused by the virtual monitor. In addition, MVP [17] explores the possibility to enable virtualization on mobile phones based on the general desktop virtualization concept.

Besides the aforementioned efforts in the scope of virtualization technologies, a series of evaluations are designed and implemented in the public Cloud with several execution scenarios, observing the real performances. Most notably in EC2_Dis [19], the performance of public Cloud instance EC2 from Amazon is evaluated based on large-scale, tightly coupled, distributed tasks.

EC2_HPC [124] is another study in EC2 towards HPC applications, in which the public Cloud instance is tested with standard HPC benchmarks in terms of memory bandwidth, computation and I/O with MPI implementations. The matter in this particular case consists of the Cloud virtual instances being managed by the Cloud platform but not the users, which means that it is still impossible for users to define the network topology for their instances. In addition, for every reboot, the location of the virtual instances may change due to the migration policy in the Cloud. As a result, the MPI

implementations that require instances in the same subnet perform poorly.

In the same context as evaluating the performance of EC2, EC2_NASA [189] has realized a performance analysis of Amazon EC2 by comparing it to the Pleiades, the NASA's supercomputer. The results show that in the single node mode where shared memories are used, the performance of EC2 is equivalent to the Pleiades due to the utilization of the same node architecture. However, the EC2 instances increase the delay when scaling up due to the underlying network architecture. This performance behavior has been observed in a study presented in [156], which described similar results and point out that the interconnection issue is one key bottleneck in the Cloud to support HPC-based applications.

Empirical performance analysis has been conducted and presented in [155], which evaluated the efficiency of public Cloud EC2 instances from the perspective of Many-Task Computing (MTC). Based on the experiment scenarios, the work suggests an order of magnitude improvement is required in the Cloud to achieve similar performance as Grid.

Table 2.5: Cloud-based Hardware-layer Studies

| **Work** | **VMM** | **Resource** | **Target** | **Evaluation** |
|---|---|---|---|---|
| Xen_Comp [16] | Xen | Static | Comp, I/O | SPEC, lmbench |
| Xen_Net [191] | Xen | Static | Net | Zero-copy netperf |
| KVM_Sys [163] | KVM | Static | Comp, I/O, Net | N/A |
| VM_I/O [227] | VMware | Static | I/O, Net | nettest |
| VM_Comp [235] | VMware | Static | Memory | SPEC, database |
| MVP [17] | VMware | Mobile | Comp | kstats |
| EC2_Dis [19] | Xen | Static | Comp, Net | cavity3D |
| EC2_HPC [124] | Xen | Static | Net | HPC standard |
| EC2_NASA [189] | Xen | Static | Comp | HPC standard |
| EC2_MTC [189] | Xen | Static | Comp, Net | MTC standard |

Table 2.5 briefly summarizes the works and studies above. From the table, we can conclude that the flexible Cloud solutions pose as a potential and promising feature to support light-weight multi-tenant MCC mobile applications. However, the virtualization technologies inevitably introduce non-negligible performance loss regarding computing and communicating, and the Cloud nature is not suitable for the high-performance computing-intensive or communication-intensive environment. As a result, scheduling, load balance, and communication protocols must be tailored appropriately under the current hard-layer infrastructure for mitigating performance losses and leveraging offloading

capabilities.

### 2.3.2 Virtual Instance Management

While the hard-layer paves the foundations of the Cloud architecture, the intermediate virtual resource management layer directly tackles the resource provisioning issue in the MCC-based offloading. As the bridge that connects the physical resources and application tasks, this layer encapsulates underlying implementation details and hosts application tasks in the virtual machines. The core functioning of this layer is to instantiate and maintain the virtual instances in proper physical hosts to process offloading tasks in a way that the energy consumption of the Cloud is minimized and the requirements of incoming tasks are acceptably satisfied. To achieve it, typically the studies are oriented by VM live migration and resizing which enable the VMs dynamically modify the computing capability and location, as shown in Table 2.6.

Table 2.6: Cloud-based VM-layer Studies

| **Work** | **Arch** | **Serv** | **VM** | **Energy** | **Scheduling** | **Prediction** | **Migration** |
|---|---|---|---|---|---|---|---|
| EaRAH [21] | Yes | N/A | N/A | DVFS | MBFD | N/A | MM,HPG,RC |
| PaPCR [162] | Yes | Yes | Yes | DVFS | Min-Price RT-VM | Look-ahead | N/A |
| PEGCA [118] | Yes | N/A | N/A | State | Predict-based | Neural Network | N/A |
| EaACB [126] | N/A | N/A | N/A | State | ACO | History-based | N/A |
| MoACS [137] | N/A | N/A | N/A | DVFS | ACO | History-based | N/A |
| EaFVM [117] | Yes | N/A | Yes | VM-based | Branching Heuristic | N/A | N/A |
| BbGbS [243] | Yes | Yes | Yes | N/A | Predict-based | History-based | Yes |
| LBVMR [151] | Yes | Yes | Yes | N/A | GA | N/A | Yes |

The VM level energy management is originated from VM-based migration – to achieve high energy efficiency by gathering VMs and releasing low load physicals. In BbGbS [243], a migration-based framework is designed and evaluated, targeting the Xen hypervisor. The Sandpiper architecture can automatically trigger load balancing behaviors in a self-organized manner without access to VMs. The data collection model is implemented to collect guests data outside the virtual guest, regarding CPU, network, and RAM. Domain-0 VM and hypervisor in Xen are utilized to obtain CPU task, network event and RAM swap list of each virtual client in the physical host. The detection algorithm is based on historical data and time-series prediction algorithm.

As mentioned in LBVMR [151], the overhead introduced by migration is inevitable the migration-based load balancing algorithms. In their research, a genetic-based algorithm

is proposed. By simulating and evaluating resource scheduling solutions of a certain number of generations, an acceptable scheduling strategy can be found concerning load balancing and the number of migrations. The presentation of the system employs the concept of spanning trees, which involves management root, physical machine and virtual machines. An individual tree represents a scheduling solution, which performs mutation and crossover to produce offspring.

Due to the significant computational and communication overhead introduced by the live migration, several works sought to handle the scheduling issue without adjusting the VM position frequently. This approach has been explored by EaRAH [22, 21, 23], which abstracted the virtual resource scheduling process as the bin packing problem where incoming VMs are presented as objects while physical hosts are bins. In the scheduling process, VMs are first instantiated in a set of physical hosts. Then, VMs are migrated if there exist hosts that violate utilization boundaries.

Concerning the improvement of the scheduling efficiency, the scheduling components involve prediction functions to tailor the VM-host mapping, such as the work described in PaPCR [162]. It also proposes a DVFS-based scheduling solution, focusing on the processor speed adjustment. Instead of utilizing live migration to alleviate the unbalanced load, this work investigates the use of a look-ahead prediction based on the current VM proportional share so that the processor can modify its frequency to achieve higher energy efficiency while meeting the task requirements.

Similarly, PEGCA [118] deploys a power-down prediction-based resource scheduling algorithm to solve the unknown load issue. In the described solution, neural network technology is utilized to learn the load variation behavior and predict further load change based on the current VM load. Based on the prediction, the Cloud management node can either reduce the number of VMs and suspend physical hosts if further system load decreases, or invoke additional physical servers and increase live VMs for the increasing of loads.

Besides the predictions, heuristics have also been used to tackle the performance of scheduling in the Cloud. EaACB [126] and MoACS [137] introduces Ant Colony Optimization (ACO) protocol to achieve self-manageable energy-aware scheduling. Unlike EaRAH [21] that first generates random scheduling solutions then reallocates the unbalanced load to achieve energy or execution efficiency optimization, EaACB [126] seeks to find the global optimization directly in the scheduling stage. In the proposed bin-packing ACO, the pheromone is defined as the minimum energy consumption scheduling solution generated by ants. In order to find the global optimization, a probabilistic rule is defined

to enable ants to bypass local optimal value stochastically while the pheromone trail updates gradually converge all ants to the final solution.

Similar to this ACO-based scheduling technique, MoACS [137] also deploys the ACO algorithm to solve the multi-dimension bin-packing issue. Unlike the previous work that only concerns energy consumption according to the on/off states, this study further evaluates the pheromone on the CPU utilization, which leads to a more robust result and a faster convergence rate.

In addition to the above nature-inspired algorithms, the techniques described in both EaFVM [117] and EVRP [144] deploy the Constraint Programming (CP) to search the best scheduling solution. In the proposed design, task requirements and current system load are translated into QoS and availability constraints while VM-based power objective is established for a heuristic search. In the execution, the branching heuristic is implemented to improve scheduling efficiency.

### 2.3.3 Task and Process Scheduling

The task and process related energy-aware issues are inherited from traditional distributed systems, such as clusters and Grid [131]. The energy-aware handling of such conventional systems is typically conducted through balancing and remapping tasks so that computing resources or networking resources can fully or partially powered-down for energy reservation. This layer is required to balance the trade-offs between the Cloud energy cost and the task execution quality as shown in 2.7.

EMMC [169] consists of an energy-aware cluster-based framework where describes two power-down based task scheduling frameworks – Covering Set and All-In Strategy – presenting a performance comparison and discussion. By utilizing the power-down strategies, a promising amount of energy consumption can be reduced. In this same context, a task replication reconfiguration method [171] has been proposed to develop an energy-aware cluster. By limiting the size of covering subsets and the replicas, energy can be reserved with a certain level of availability.

Inspired by bio-inspired optimization heuristics, many works have employed approaches based on genetic algorithms to handle the energy-aware task scheduling issue. In that sense, GATSC [138], JSMCC [176], EaGAT [94], and PBoHM [192], as well as variations of these strategies described in [157, 93], have made efforts to handle the task scheduling issue in the Cloud based on the Genetic Algorithm (GA). In these works, the scheduling problem is formalized as a self-managed mapping process that applies

Table 2.7: Cloud-based Task-layer Studies

| **Work** | **Resource Scope** | **Exec Eff** | **Energy Eff** | **Task Model** | **Network Model** | **Comp Model** | **Scheduling** | **Prediction** | **Scal** |
|---|---|---|---|---|---|---|---|---|---|
| EMMC [169] | Cluster | N/A | Yes | Meta | N/A | State | CS, AIS | N/A | N/A |
| EaGAT [94] | DataCenter | Yes | Yes | Meta | N/A | DVFS | GA | N/A | N/A |
| GATSC [138] | Cluster | Yes | N/A | Meta | N/A | MapReduce | GA | KCCA | N/A |
| JSMCC [176] | Inter-DC | Yes | N/A | Meta | N/A | State | GA | N/A | Yes |
| PBoHM [192] | DataCenter | Yes | Yes | DAG | N/A | DVFS | GA | N/A | N/A |
| MOAEA [249] | DataCenter | Yes | Yes | DAG | Delay, Size | DVFS | PSO | N/A | N/A |
| HBDSA [195] | DataCenter | Yes | N/A | Web-app | N/A | Service | Honey Bee | N/A | N/A |
| DENS [165] | Multi-DC | Yes | Yes | Video App | 3-tier Net | DPM | Weighted-tier | Buffer | Yes |
| EeDRC [33] | DataCenter | N/A | Yes | Database | Traffic-based | DVFS | Aggregate-Net | Traffic | N/A |
| QGMMH [150] | Grid | Yes | N/A | Meta | Bandwidth | FLOP | Min-Min | GHS | N/A |
| UPGMM [95] | DataCenter | Yes | N/A | Meta | N/A | FLOPS | Min-Min | N/A | N/A |
| LBTCC [236] | DataCenter | Yes | N/A | Meta | 3-tier Net | FLOPS | OLB, Min-Min | N/A | N/A |
| EMmTS [26] | DataCenter | Yes | N/A | Meta | Bandwidth | FLOPS | Max-Min | N/A | N/A |
| IMMTS [173] | Cluster | Yes | N/A | Meta | N/A | FLOPS | Max-Min | Load | N/A |
| DECM [136] | Cloudlet | Yes | Yes | Meta | Route-based | Meta | DP | N/A | Yes |

DVFS-based on resource models to application load models. The use of DVFS is guided by the energy-aware evaluation matrix, which typically targets the task execution time and the system energy cost as the multi-objective context.

MOAEA [249] used another bio-inspired optimization class of algorithms, which utilizes Particle Swarm Optimization (PSO) based population heuristic optimization to pair the tasks, processors with different voltage levels. In the proposed solution, the task mapping issue is defined as finding the optimal task-process-voltage triplet sets that contain the best fitness value. In the evolution process, the fitness value is evaluated as a combination of the task makespan, cost and energy while the descendants are generated by comparing the distance to the current optimal. To achieve the global optimal, offspring can randomly select another particle position with a certain probability.

HBDSA [195] also introduced a load balancing mechanism that is based on honeybee optimization algorithm. Idle computing resources are divided into foragers and scouts. Scouts serve a random task in the task queue and record the task information after completing globally (on the advertising board) with a probability. The foragers randomly read the globally published task information based on adjusted probability, and then choose and harvest the task. The probability is calculated based on the profit of the server and the whole system recorded by the global advertising board.

Besides the biological based algorithms, several works attempted to shorten the makespan and increase the utilization ratio of the resources by using Min-Min optimization approaches. QGMMH [150] employed the concept of Min-Min to support QoS-aware Grid computing. It calculates the minimum completion time of each task and prioritizes small tasks.

In the same trend, UPGMM [95] proposed an enhanced Min-Min algorithm with user-priority awareness and dynamic load balancing, which tackles the multi-tenant issue on the Cloud environment. In addition to the existing Min-Min functions, the proposed algorithm introduces an extra load comparison step, selecting small tasks from resources that are most overloaded. The selected tasks are then reassigned to low load resources to execute if the makespan allows. Although the proposed algorithm produces overhead due to the extra load comparison step, the experiment result shows an overall performance improvement in terms of task execution time, compared to Min-Min.

Compared to Min-Min, large tasks in Max-Min optimization strategies are considered to have higher priority and are scheduled first. In EMmTS [26], an enhanced Max-Min scheduling algorithm is proposed. In this approach, instead of selecting the largest tasks, the authors argue that choosing tasks with average or just above average load can lead to better makespan and load balance. The experiment result shows an improvement of the proposed algorithm compared to the improved Max-min algorithm when new large tasks are involved.

Similarly, IMMTS [173] employed Max-Min optimization strategy to tackle the elasticity issue in the Cloud. In traditional Min-Min and Max-Min studies, the handling of new tasks is mainly under the assumption that the previously scheduled tasks are completed, which is not always true especially in the multi-tenant Cloud environment. The proposed method designs and maintains a task table and a virtual resource table to trace the real-time task and resource change. The task status of the virtual machine is collected by the information agent inside the virtual machine. The collected data is then sent to load balancer, based on which relocation decisions are made.

Unlike the previous studies that focused on computing-related issues, DECM [136], DENS [165], and EeDRC [33] have devised a bandwidth-based network model especially to tackle the communication-based energy-aware problems. DECM [136] deploys the concept of dynamic programming to determine Cloudlet routing and task scheduling. According to the sequence of tasks and the position of Cloudlets, energy-aware recursive functions are designed to pair tasks and resources.

DENS [165] and EeDRC [33] evaluated the energy consumption of ports, line cards,

and switches, from the view of data center level, rack level, and server level. As suggested, the energy consumption of the ports is highly related to the scale of traffic created by the application tasks. Traffic prediction models have also evaluated both works. DENS [165] evaluates the potential network traffic according to the current uploading queue while EeDRC [33] defines traffic load prediction based on historical data. Based on the evaluation of task traffic, a trade-off can be made between communication delay and bandwidth especially for delay tolerant applications, which can reduce the system-level network energy cost.

## 2.4 Open Challenges and Research Directions

As discussed in the sections above, many research efforts have been presented regarding the energy-aware issue in MCC-based offloading. However, due to the novelty of the MCC concept model, enabling energy-efficient offloading remains a challenge. The main obstacles and potential research opportunities are summarized as follows:

1. Mobile-oriented

    (a) Availability. The availability issue in energy-aware MCC-based offloading is originated by the distinctness of computing architectures between mobile and static devices. However, the mobile devices cannot transparently translate the mobile-based task sets into runnable executions directly for the static offloading resources. This lack of interpreter standard in MCC necessitates the developers either to manually divides tasks into mobile execution tasks and offloading executions or to deploy an intermediate task partitioner that can translate the codes across the platforms. On the one hand, the manual approach requires a considerable amount of work when migrating traditional mobile tasks to the MCC platform. On the other hand, the auto-partitioning method inevitably introduces run-time computing and energy overhead.

    (b) Mobility. Unlike the static computers that are connected through wired networks, the mobile elements are moving under the wireless channels. The unstable offloading environment may introduce unpredictable communication overhead which further impedes the energy-efficient offloading as discussed in Section 2.2. Many profiling technologies are proposed to alleviate the influence of the fluctuated network that may lead to inappropriate offloading

decisions. However, the profiling process itself introduces energy overhead, and the profiling accuracy can still be improved. Also, the decision-making algorithms should concern the coverage of the offloadees and the mobility of the offloaders. Penalty functions can be used to penalize the potential offloading interruptions in the channels.

2. Cloud-oriented

   (a) Security. As a distributed paradigm, MCC requires data sharing fundamentally between personal mobile users and the public Cloud resources. However, considering the potential security threats, users should not always share the task-related data especially when the users first enter a new local area and are surrounded by unknown devices. Instead of stamping and sharing user-based task information directly in the local area, protection mechanisms should be implemented on both the mobile part and the local Cloudlet to filter user-sensitive data and detect malicious behaviors in an energy-efficient manner.

   (b) Execution Efficiency. Due to the deployment of RPC in MCC-based offloading, the task execution efficiency highly depends on the status of remote resources. However, from the perspective of Cloud operators, the Cloud system-level energy efficiency often conflicts with the task execution efficiency. According to the patterns of mobile tasks that are typically small and delay-sensitive, traditional energy-aware migration-based scheduling protocols heuristic-based solutions may easily lead to profiling failures due to intolerable communication latency.

## 2.5 Conclusion

In this chapter, the state-of-the-art of works related to energy-aware is extensively surveyed in the scope of MCC-based offloading,. The origin of MCC is illustrated based on the nature of Mobile Computing and the comparison between Grid Computing and Cloud Computing. The offloading process, which consists of partitioning of tasks, profiling, decision-making, and offloading components, has been extensively explored in several works, considering the local devices, Cloudlets, and the remote Cloud. The networking aspects, as a determinant factor in performance, have also been investigated. All these elements have been broadly described and discussed in the survey study, guided through

the functionality, issues, and solutions of the schemes, architectures, and techniques. This chapter gathered and examined the existing studies, systematically classifying according to the perspective of the mobile device and the Cloud, observing performance aspects in terms of the energy-aware offloading processing and the trade-off between energy reservation and execution efficiency.

# Chapter 3

# A Cloudlet-based Multi-layered Offloading Prototype

This chapter proposes a prototype for Cloudlet-based offloading from scratch. Concerning the evaluation of task load, in the proposed model, computing intensive distributed simulation applications are used as the offloading task. For the Cloudlet infrastructure, local Cloud and public Cloud resources are utilized for the deployment. This chapter presents a multi-layered cloud simulation scheme is proposed, concerning usability, elasticity, energy consumption, and fast deployment. Components are designed and implemented to provide unlimited computing resources to end users by coordinating public computing resources during runtime. Energy-aware elements are utilized to reduce unnecessary energy cost from computing resources. A deployment model is designed to accelerate the migration and deployment process of the cloud simulation platform. In addition, the scheme contains functions for job scheduling, monitoring, and a friendly web-based graphic interface to ease the configuration, operation, and maintenance of the underlying system.

The reminder of the chapter is organized as follows. In Section 1, the proposed scheme is described, including architecture, key components, and functioning. In Section 2, several experiments are presented and discussed in terms of energy consumption, management efficiency, and performance. In Section 3, the proposed solution design and experimental results are summarized, and directions for future works are provided.

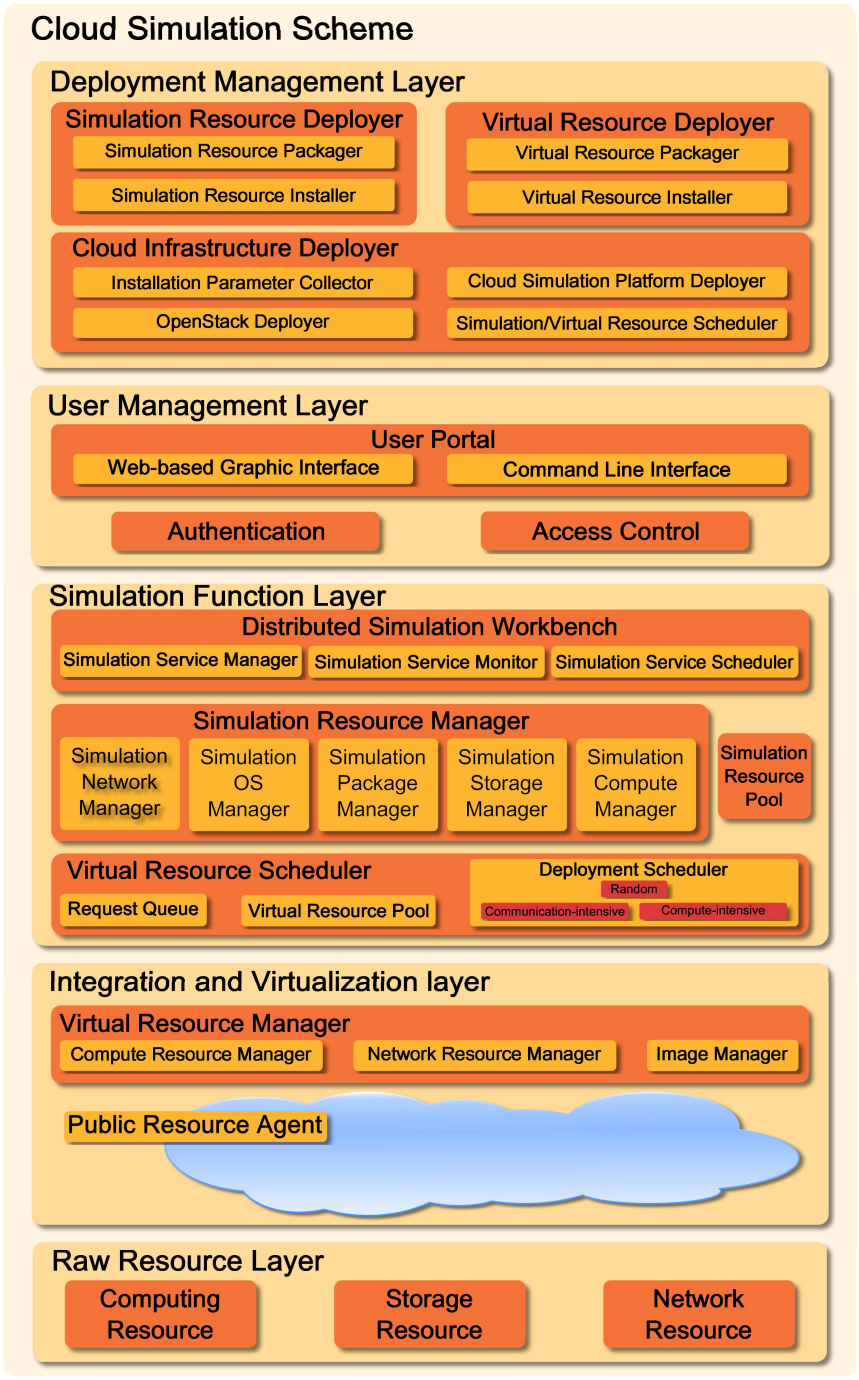



Figure 3.1: Cloud Simulation Framework

## 3.1 Cloud Simulation Platform

As shown in Figure 3.1, this Cloud Simulation Platform consists of five layers: the Raw Resource Layer, the Integration and Virtualization Layer, the Simulation Function Layer, the User Management Layer, and the Deployment Management Layer. Specifically, the deployment mechanism, security mechanism, migration mechanism, and resource scheduling mechanism will each be illustrated in detail with the description of related key features of each layer.

### 3.1.1 Deployment Management Layer

Fast and automatic deployment and migration of the proposed scheme in new environments is a priority. To achieve this, the Simulation Resource Deployer, the Virtual Resource Deployer, and the Cloud Infrastructure Deployer (as depicted in Deployment Management Layer of Figure 3.1) cooperate to backup current platform status, pack and transmit core resources to new environments, and deploy the platform based on its previous saved status.

#### Simulation Resource Deployer

The Simulation Resource Deployer focuses on simulation-related package handling. These simulation-related packages are managed by the Simulation Storage Manager (in Layer 3, Figure 3.1) and are stored in both cloud and block storage. When platform packaging starts, the Simulation Resource Packager is triggered by the packaging process, and it calls the Simulation Storage Manager to find all packages that require packing. After the search is completed, the Simulation Resource Packager compresses all necessary packages, waiting for transmission. When installation begins, the Simulation Resource Installer is invoked by the installation main program. It decompresses simulation-related packages and stores these packages in a temporary folder. After the Cloud Infrastructure is properly deployed, the Simulation Resource Installer calls Simulation Resource Manager to register the packages currently stored in the temporary folder. Finally, these packages are removed from a temporary folder and stored in the proper cloud storage and block storage again, ready for further usage.

**Algorithm 1:** Cloud Simulation Platform Packaging and Installation Algorithm

```
1  Require: SIGN_SR, SIGN_VR, SIGN_CP.
2  Note: CID − Cloud Infrastructure Deployer, IPC − Installation Parameter Collector,
     OD − OpenStack Deployer, CSDP − Cloud Simulation Platform Deployer,
     SVRS − Simulation/Virtual Resource Scheduler. UM − UserManager
3  if SIGN_CP == packaging_start
4  then
5      SIGN_SR ⇐ SIGN_CP
6      SIGN_VR ⇐ SIGN_CP
7      CID.stop(UM, SRM, VRM, SIGN_CP)
8      while SIGN_CP! = packaging_ready
9      do
10         SRpacks ⇐ SRP.search(SRM, SIGN_SR)
11         SRpacks.save()
12         VRpacks ⇐ VRP.search(VRM, IM, SIGN_VR)
13         VRpacks.save()
14         if SIGN_SR == ready && SIGN_VR == ready then
15             CID.add(SRpacks, VRpacks)
16             CID.save()
17             SIGN_CP ⇐ packaging_ready
18         else
19             SIGN_CP ⇐ recheck
20             SIGN_SR ⇐ SIGN_CP
21             SIGN_VR ⇐ SIGN_CP
22     SIGN_CP ⇐ transmitting
23 if SIGN_CP == installation_start
24 then
25     CID.auth()
26     IP ⇐ IPC.search(comp, st, nw)
27     OD.install(IP)
28     CSPD.install(IP)
29     SVRS.sche(SRI, VRI)
30     while SIGN_CP! = installation_end
31     do
32         SRI.unpack(SRpacks)
33         SRI.register(SRpacks)
34         VRI.unpack(VRpacks)
35         VRI.register(VRpacks)
36         if SIGN_SR == installation_end
37         && SIGN_VR == installation_end
38         then
39             CID.test()
40             CID.end()
41         else
42             SIGN_CP ⇐ recheck
43             SIGN_SR ⇐ SIGN_CP
44             SIGN_VR ⇐ SIGN_CP
```

### Virtual Resource Deployer

The Virtual Resource Deployer functions in a similar manner as the Simulation Resource Deployer. It is responsible for the handling of the simulation units – the virtual instances. When packaging begins, it calls the Virtual Resource Scheduler (in Layer 3, Figure 3.1) and the Virtual Resource Manager (in Layer 4, Figure 3.1) to record the real-time statuses of all computing instances in the entire system. The current statuses of virtual instances are saved by creating images. The Virtual Resource Packager classifies these images according to their owners. During the installation process, the Virtual Resource Installer first checks the Authentication status of the image owners, and then unpacks the images based on image users. Before the image can be used in the new environments, the Virtual Resource installer must trigger the Virtual Resource Scheduler to register the images and rebuild computing instances accordingly. These are created with the same instance type and network topology as in the original cloud simulation platform.

### Cloud Infrastructure Deployer

The Cloud Infrastructure Deployer disposes components related to cloud middleware and virtualization tools. The configuration of these components are highly based on the new targeting environments. The Installation Parameter Collector is designed to collect relevant parameters required during the installation for the new physical environment, such as compute capability, Hard Disk capacities, and network characteristics. These parameters are passed to the *OpenStack Deployer* and the Cloud Simulation Platform Deployer to install the core components and services as shown in Figure 3.1. After this step, the Simulation/Virtual Resource Scheduler triggers the Simulation Resource Installer and the Virtual Resource Installer to handle relevant softlayer and hardlayer deployment respectively.

### Cloud Simulation Platform Packaging and Installation Algorithm

The entire packaging and installation process is shown in Algorithm 1. The packaging program begins in line 3 and ends in line 22. After the packaging main process is initialized, the Cloud Infrastructure Deployer calls the Simulation Resource Manager, the Virtual Resource Manager, and the User Portal to stop accepting new tasks, as shown in line 7. Once the current simulation tasks and the virtual instance scheduling tasks are completed, two sub-processes are triggered by the main program to package simulation

resources and virtual resources at the same time. On one hand, the Simulation Resource Packager finds the required simulation-related packages in the Simulation Resource Pool with the help of the Simulation Resource Manager. These packages are then copied and compressed from the cloud storage. On the other hand, the virtual computing resources are handled by the Virtual Resource Packager. The Virtual Resource packager calls the Virtual Resource Manager to find the virtual instances, and invokes the Image Manager to build images to save the status of each instance. After both sub-programs are completed as shown in line 14, the packed simulation resources and virtual resources are registered to the main packaging process in the Cloud Infrastructure Deployer. Finally, the Cloud Infrastructure Deployer packs external tools such as the cloud middleware and virtualization components, ready to transmit.

The installation process contains three steps: the cloud infrastructure installation, the simulation resource installation, and the virtual resource installation. The main installation process first invokes the Cloud Infrastructure Deployer to build the basic cloud platform. Before the real installation process begins, the installation parameters that are required during the cloud infrastructure installation are collected by the Installation Parameter Collector. The collecting process may need the authorization of the system administrator before it can begin. Then, the *OpenStack Deployer* and the Cloud Simulation Platform Deployer build core components in the Integration and Virtualization Layer and main services in the Simulation Function Layer. After this, The Simulation/Virtualization Resource Scheduler calls the Simulation Resource Installer and the Virtual Resource Installer to install simulation resources and virtual resources. The Simulation Resource Installer and the Virtual Resource Installer work simultaneously, uncompressing, storing, and registering relevant packages in the new environments. After these two sub-processes are finished, in the end, the Cloud Infrastructure Deployer tests the functions of key services, as well as the availability of the simulation pool and the virtual resource pool, ensuring the correct installation.

### 3.1.2 User Management Layer

This layer focuses on security issues in the cloud environment. In this layer, an authentication and access control mechanism is proposed to protect users against threats from both inside the cloud and outside the cloud. In addition, this layer also provides a web-based graphic portal and a command-line interface, through which users can access the simulation resources and the computing capability of the proposed cloud simulation

platform. This user portal enables users to design, code, analyze and test complex distributed simulations via lightweight terminals such as laptops, tablets, or even smart phones.

**Authentication and Access Control Algorithm**

Algorithm 2 shows the authentication and access control process. After the user registers in the platform, as shown in line 2, the cloud firewall adds the new user to the acceptance list. In this stage, the user receives authorization to access the shared materials in the Simulation Resource Pool. Users can take advantage of softlayer resources in the pool which gathers and presents codes, templates, solutions, applications, and experiences shared by other cloud users. In order to secure the user-defined simulations in the Simulation Resource Pool, a unique private key is created and stored by the simulation owner, which is required when the user edits or updates its private simulations and application. An X-token, controlled by the Virtual Resource Manager, is provided to enhance the security of user-defined virtual instances. This access key allows the user to access private virtual instances in the Virtual Resource Pool, as shown in line 11. It is required when users create, access, modify, or delete instances. The X-tokens and the passwords function as a two-layered security mechanism, stored separately in the platform. On one hand, if one user is accidentally attacked and loses the control of its instances due to the leak of username/password, the attacker cannot tap into the data inside the user-defined instances due to the lack of the X-tokens; on the other hand, if the X-tokens are stolen, users can stop or suspend their instances immediately through the control console (login by username/password), and then reset the X-tokens. The default lifespan of the X-tokens is 24 hours, which is set the same as the identity components in the internal cloud middleware [23]. With encryption, X-token and virtualization technology, users on the cloud are better isolated and protected so that malicious operations from cloud users or system administrators inside the cloud can be more easily blocked, ensuring that at any time, one virtual instance can only belong to one cloud user for its exclusive usage.

### 3.1.3 Simulation Function Layer

This layer provides core functions and services that naturally enable and support different types of distributed simulation standards, such as High Level Architecture (HLA), Distributed Interactive Simulation (DIS), and Data Distributed Service (DDS), which

**Algorithm 2:** Authentication & Access Control Algorithm

```
1  Require: username/password, private_key, x_token
2  User.signup()
3  User.fw.ad(user)
4  if User.auth() == True
5   then
6      User.access.add(SRP.r)
7      if User.auth(key)
8       then
9          User.access.add(SRP.w)
10         if User.auth(x_token)is not expired then
11             User.access.add(VRP.w)
12         else
13             User.auth.update()
14 else
15     User.access.del()
```

include the Distributed Simulation Workbench, the Simulation Resource Manager, and the Virtual Resource Manager.

### Distributed Simulation Workbench

The Distributed Simulation Workbench offers users a self-defined distributed simulation foreground, enabling users to focus on design, analysis and testing without further concerns regarding the configuration and maintenance of the underlying physical system. Based on the user's privilege, the Simulation Service Manager furnishes and coordinates simulation services for users, which include an online modeling service, a simulation analysis service, a fault tolerance service, and a load relocation service. The on-line modeling service is designed for modelers to code, submit, execute, and debug through a web-based lightweight interface. Other services can be lightly applied by this interface. Simulation analysis service records and provides end users with the results of simulation execution, simulation-related system logs, and simulation performance statistics, such as CPU load, simulation execution time, and network bandwidth usage, helping to reveal potential issues in the user-designed programs and applications. Fault tolerance service protects

user-defined simulations from computation errors, storage errors and network errors, providing rollback and failover mechanisms. The VM migration is utilized as an approach for simulations to recover from failed physical resources. The load relocation service supports user's ability to reschedule virtual instances and modify instance types or network topology during run-time, simplifying the set of comparison experiments. The statuses of these services are tracked by the Simulation Service Monitor. In addition, the Simulation Service Monitor detects and updates the states of the physical machines that periodically compose these simulation services. If any physical error occurs, the Simulation Service Monitor is responsible for blocking relevant resources and informing the cloud simulation platform administrator. The Simulation Service Scheduler manages and maintains the background execution queue for the simulation service requests from multi-users, ensuring proper execution orders of simulations. In addition, the Simulation Service Scheduler calculates the energy consumption of the entire platform and implements energy efficient policies, reducing energy consumption by migrating and centralizing virtual instances and releasing idle physical machines.

**Simulation Resource Manager**

The Simulation Resource Manager concerns the network, OS, software dependency, storage, and computation aspects of the distributed simulations. It also manages the Simulation Resource Pool, which gathers, presents, and shares codes, templates, solutions, and applications of diverse distributed simulation standards among the cloud users. The Simulation network manager handles the network model for the given distributed simulations. It currently supports unicast, multicast, and broadcast approaches used by most distributed simulation standards for communication. In addition, it works with the Virtual Resource Scheduler to initialize the virtual instance's network topology for self-defined virtual instances. The simulation OS Manager controls and stores a list of images of OS for booting virtual instances. Users are able to modify system settings, such as firewall rules, partitioning of the hard disk, software repository, and other system information before the virtual instances are booted. The Simulation Package Manager is responsible for arranging the simulation of dependent packages. This component natively supports standards including HLA, DIS, and DDS, and it automatically configures and installs dependency packages that each corresponding implementation may further require. In order to enable the diversity of distributed simulations, the Simulation Package Manager also allows users to manage and define self-designed packages for specific scenarios. The Simulation Storage Manager provides two types of storage in the back-

ground for simulations: cloud storage and block storage. Cloud storage is mainly used as replicas to backup resources in the Simulation Resource Pool in the event that local resource storage servers fail. In addition, these cloud replicas can be easily shared among users inside and outside the platform. The block storage saves all necessary information that this cloud simulation platform requires during runtime and it utilizes the Network File System (NFS) to accelerate the scheduling process of simulation resources. The Simulation Compute Manager concerns the compute capability for the current distributed simulation. It communicates with the Simulation Service Monitor to check local compute resources. If local resources are not sufficient for current simulations, it calls the Virtual Resource Manager and the Public Resource Agent to enable additional public computing resources to fulfil the computation requirements.

**Virtual Resource Manager**

Due to the diversity of modeling and simulation applications, computing resource requirements differ. The Virtual Resource Scheduler is utilized to control the granularity of resources so that the need of diverse simulation applications can be better met. Based on the requirement information from users, the Virtual Resource Scheduler helps to arrange the simulation environment based on the parameters of the computing unit, such as the number of virtual processors, memory size, network topology, and the quantity of virtual machines. The Virtual Resource Scheduler invokes corresponding components in the Virtual Resource Manager and in the Public Resource Agent, discovering proper computing resources and instantiating virtual computing instances in the Virtual Resource Pool.

**Scheduling Algorithm and Dynamic VM Reallocation**

Several resource scheduling algorithms are provided for instance creation, as shown in Algorithm 3: random, computation-intensive, and communication-intensive. $N$ in line 1 defines the number of virtual resources that require deployment in simulations. The default value is 0, which means the current available resources are already adequate, and there is no need for additional virtual resources to be scheduled from the Virtual Resource Pool. In this context, $IT$ in line 1 is short for the instance type. This parameter expresses the type of virtual instances that may be further initiated and utilized. It contains computing and storage capability information such as processors, RAM and hard disk. $SA$, which is first shown in line 1, stands for the type of the scheduling algorithm. In Random, as shown from line 4 to line 14, the Virtual Resource Scheduler randomly

creates instances. First, from the Virtual Resource Pool, a queue of all current available resources are listed. Then, based on the number of available resources, a random integer number $n$ is selected according to a normal distribution. The resource whose index number is $n$ in the resource queue is chosen as the destination physical host. After this selection, the request to boot the virtual instance is queued in the background and then redirected to the destination's physical host. Finally, the virtual instance is instantiated on the targeting physical host, and the system records and updates the resource information in the Virtual Resource Pool. At this time, one round of scheduling ends and the next round is ready to begin. Differently from the random scheduling, the computation and communication-intensive approaches make use of a greedy technique for allocation of resources. In computation-intensive scheduling, shown from line 15 to line 24, a current available resource queue is first created in the same manner as the random scheduler. Then, the resource whose current computing capability is the largest in the queue is selected as the destination resource. In this mode, large virtual instance types are particularly chosen as $IT$ for simulations involving large computation tasks. With this type of scheduling and the large virtual instance type, the rounds of scheduling can be noticeably reduced. In the communication-intensive mode shown between line 25 and line 35, the first resource in the current available resource queue is selected as the destination host. Then, this selected physical host attempts to schedule and boot as many virtual instances as possible on itself until its compute capability is exhausted. In this way, communication-intensive simulation entities can be likely to exist in the same physical host, so that the communication distance of simulation entities is reduced. After each round of scheduling, a virtual instance migration may be triggered by the Simulation Service Scheduler, reallocating virtual instances and centralizing simulation tasks. In this case, idle physical resources can be released and the energy consumption of the system can be reduced.

### 3.1.4 Integration and Virtualization Layer

In order to utilize the raw computing capacity from diverse resources, the Integration and Virtualization Layer is designed to coordinate these resources and implement virtualization technologies. This layer contains two core components: the Virtual Resource Manager and the Public Resource Agent. Although these components are not visible to end users, they play an important role in automatic configuration, management and maintenance of the underlying raw resources while hiding complex details of

**Algorithm 3:** Scheduling Algorithm & Dynamic VM Reallocation

```
1  Require: N, IT, SA
2  N_update()
3  IT_update()
4  if SA is Random Scheduling
5   then
6      while N! = 0 do
7          ava_res_que ⇐ res_pool.search(IT)
8          n ⇐ random(0, ava_res_que.index)
9          des_res ⇐ ava_res_que.get(index = n)
10         boot_que.add(des_res, IT)
11         sys.rec()
12         sys.update()
13         N_update()
14         VM.realoc()
15 if SA is Compute − intensive Scheduling
16  then
17     while N! = 0 do
18         ava_res_que ⇐ res_pool.sort(IT)
19         des_res ⇐ ava_res_que.find_low_load()
20         boot_que.add(des_res, IT)
21         sys.rec()
22         sys.update()
23         N_update()
24         VM.realoc()
25 if SA is Communicate − intensive Scheduling
26  then
27     while N! = 0 do
28         ava_res_que ⇐ res_pool.search(IT)
29         des_res ⇐ ava_res_que.get(index = 1)
30         while des_res is not exhausted or N! = 0 do
31             boot_que.add(des_res, IT)
32         sys.rec()
33         sys.update()
34         N_update()
35         VM.realoc()
```

system management.

### Virtual Resource Manager

The Virtual Resource Manager plays a crucial role in integrating physical resources, which includes three sub-managers: the Compute Resource Manager, the Image Manager, and the Network Manager. The Virtual Resource Manager maintains and manages the cloud middleware and virtualization tools. Based on the user-defined requirements from the aforementioned Virtual Resource Scheduler, the Virtual Resource Manager calls its relevant sub-managers to process compute-related, storage-related, or network-related management programs. These programs work as based for building the experimental or execution environment. The Compute Resource Manager deals with the establishment of virtual instances based on the selected scheduling algorithm and the characteristics of the user-defined virtual instances. The Image Manager handles the operating system and simulation-related soft-layer issues. According to various implementations of different distributed simulation standards, the Image Manager pre-arranges the operation system and the simulation-related packages for the virtual instances before it is booted from the Compute Resource Manager. The Network Manager controls the network topology of the virtual instances. In this case, these virtual instances can be instantiated in the same physical host or among multiple adapters, based on simulation requirements. In addition, the Network Manager maintains the key bridge that is designed to support communications between local virtual instances and public instances. It virtualizes the public portal of the cloud simulation platform, binding the virtual portal and public tunnels.

### Public Resource Agent

In order to achieve rapid elasticity in the cloud simulation platform, the Public Resource Agent is designed to increase computational capabilities during runtime. This is achieved by introducing public physical resources such as instances in the public cloud (eg., Amazon EC2 instances). To coordinate the local resources and public resources, the diversity of the public network should be taken into careful consideration. In the public network, multicast and other approaches of package transmission protocols, which are widely used in many distributed simulation standards, may not be well supported by routers, switches, or hubs, which are in the communication path between cloud instances deployed over different remote networks. For instance, routers drop multicast

messages by default if the destination resource and the original resource are not set in the same subnet. The Public Resource Agent is introduced to manage the public networks to meet the demands of distributed simulation. First, the Public Resource Agent builds specific virtual tunnels for each public resource based on the physical network, penetrating the public Internet and encapsulating packages and messages that need to be sent and received. After the virtual tunnels are initiated, the relevant Firewall rules are modified to support the established communication among local instances and public instances, redirecting packages and messages to the proper destinations. The Public Resource Agent listens to the Virtual Resource Scheduler and analyzes the simulation requirements defined by users, so it only invokes public instances if local current computational capability are not sufficient and the Service Level Agreement of the user is not violated.

**Simulation Execution Algorithm**

In Algorithm 4, the underlying simulation execution process is described step-by-step. First, when the Virtual Resource Scheduler receives simulation tasks from the user, the identity and priority of a user is verified, as shown in line 3. This ensures that the user can reach full access to the Simulation Resource Pool and the Virtual Resource Pool. The detailed authentication and access control process is demonstrated in Algorithm 2. Then, based on the information collected from the Simulation Service Monitor, the Virtual Resource Scheduler decides whether it is necessary to invoke public instances. On one hand, if local computing capabilities are sufficient (as shown in lines 5 to 10), the Public Resource Agent is not contacted. In this situation, the Virtual Resource Manager calculates the number of virtual instances that must be instantiated and collects user-defined parameters such as SA, IT, and simulation related packages. The Virtual Resource Manager then boots the instances in line 9, as illustrated in Algorithm 3. On the other hand, if local computing resources do not meet the requirements, the Public Resource Agent is called to arrange additional computing capabilities, as shown in lines 12 to 17. In this case, after the local computing resources are exhausted, the Public Resource Agent configures, links and bridges the public instances to local user groups, enabling communications among local virtual instances and public instances in the same user-defined group. In this stage, the virtual resources are properly prepared, and are ready to execute softlayer commands. Before the distributed simulation applications begins to run, the simulation scripts are supposed to map to destination instances by the Simulation Resource Manager, as shown in lines 20 to 25. The Simulation Resource Man-

ager pairs the index of user-defined simulation packages in the Simulation Resource Pool and the index of virtual instances in the user's private Virtual Resource Pool, delegating simulation executing packages to destination instances. After each round of simulation, the executing packages can be remapped to different instances for comparison study. After the simulations are completed, as shown in lines 26 to 32, the simulation-related packages and logs are uploaded to Hard Disk storage and cloud storage. The status of computing resources is saved in the system as images and the computing capabilities can properly store and reused later for a future instantiation.

### 3.1.5 Raw Resource Layer

The Raw Resource Layer is a pool of untreated resources, such as computing resources, storage resources, and network resources. The whole cloud simulation platform is built on top of these resources. In this layer, physical hosts are not configured and coordinated, as they contain only an operating system and basic software that such an operating system brings.

## 3.2 Experiments and Result Analysis

The prototype of this platform has been implemented and deployed on a cluster, on one single machine, and on the Amazon EC2 instance respectively to test and compare the performance. The cluster was composed of 20 nodes, and each node of the cluster contained a Quad Core 2.40GHz Intel(R) Xeon(R) CPU and 8 gigabytes of DIMM DDR RAM memory. All nodes were interconnected through a Myrinet optical network that allowed data transmission up to 2 gigabytes per second. The single machine setup contained a Quad Core 2.40GHz Intel CPU (4700MQ) and 16 gigabytes of RAM. VMware Workstation [227] was used as the virtualization tool, which created one management node and two computing nodes. The details of the nodes are shown in Table 3.1. For public cloud instances in our experiments, T2.micro instance from AWS [9] was employed, which provided a basic configuration: one VCPU and 1 GB of memory. Based on different deployment environments, instance types, scheduling algorithms, and experiments were implemented to evaluate the energy consumption, resource provision efficiency, and simulation performance.

The first experiment concerned the preparation time of a simulation environment, including the computing resource scheduling time, raw resource providing time, and

**Algorithm 4:** Simulation Execution Algorithm

```
1  Require: VRP.w, N, IT, SA, P.
2  Note: VRP − Virtual Resource Pool, P − Simulation related Packages,
     VRS − Virtual Resource Scheduler, VRM − Virtual Resource Manager,
     SRM − Simulation Resource Manager, CRM − Compute Resource Manager,
     IM − Image Manager, NM − Network Manager, PRA − Public Resource Agent,
     SRP − Simulation Resource Pool.
3  while !VRS.end() && User.access(VRP.w) do
4      while VRS.cur(VRP) <= VRS.req() do
5          if VRS.req() <= VRS.cur(VRP.loc())
6           then
7              VRM.N.set() ⇐ VRS.req() − VRS.cur(VRP))
8              VRM.IM.set(P) ⇐ SRM.map(SRP)
9              VRM.boot(VRM.CRM(N, IT), VRM.NM(SA), VRM.IM(P))
10             VRS.update()
11         else
12             VRM.N.set() ⇐ VRS.req() − VRS.cur(VRP.loc())
13             VRM.IM.set(P) ⇐ SRM.map(SRP)
14             VRM.boot(VRM.CRM(VRS.cur(VRP.loc()), IT),
                VRM.NM(SA), VRM.IM(P))
15             PRA.set(VRM.N, PRA.getpair(PL), VRM.IM(P))
16             VRM.NM.bridge.setgroup(VRM.cur(user), PRA.cur(user))
17             VRS.update()
18     VRS.end() ⇐ User.def()
19 while !SRM(end)&&User.access(VRP.w) do
20     VRM.insList.set(user) ⇐ VRM.cur(VRP.user.simX)
21     SRM.simList.set(user) ⇐ SRM.upload(SRP.user.simX)
22     SRM.mapSimPair(VRM.insList(user), SRM.simList(user))
23     SRM.execute()
24     SRM(end) ⇐ User.def()
25 if SRM(end)
26  then
27     SRM.upload()
28     SRM.SRP.save()
29     VRM.IM.save()
30     VRM.recycle()
31     User.access ⇐ NONE
```

Table 3.1: Single Machine Environment

| **Host** | **VCPU** | **RAM** | **Bandwidth** | **Hard Disk** |
|---|---|---|---|---|
| Management | 2 | 2 G | 100 Mbps | 30 G |
| Computing 1 | 4 | 4 G | 100 Mbps | 40 G |
| Computing 2 | 1 | 2 G | 100 Mbps | 20 G |

softlayer resource packaging time. The computing resource scheduling time involves the components of the whole platform; the scheduling time starts when the User Portal receives the user's resource requests and ends after these requests are properly stored, recorded, and processed in the Request Queue managed by the Virtual Resource Manager. The rare resource provision process includes the selection of computing resources in the raw computing pool, the mapping of OS and computing resources, and the establishing of instances. The softlayer resource packaging procedure focuses on the building of a simulation-run-ready environment, which contains the simulation-related package dependency analysis, package searching, package configuration, and package installation. In this experiment, the Linux system was used as the operating system for the virtual instances; CentOS 6 and Fedora 17 were used. The simulation applications used one testing restaurant simulation application implemented in Portico java-based RTI [231] as the HLA simulation; one simple Sender/Receiver application implemented in OpenDIS [187] as the DIS simulation; and one HelloWorld application of OpenSplice [20] as the DDS simulation. The virtual instance type for the cluster and the single machine employed in this experiment is shown in Table 3.2. In this experiment, random scheduling was selected as the scheduling algorithm for the computing instances.

Table 3.2: Virtual Resource List

| Virtual Instance Type | VCPUs | RAM(MB) | Hard Disk(GB) |
|---|---|---|---|
| Mini | 1 | 512 | 0 |
| Small | 1 | 1024 | 10G |
| Medium | 2 | 2048 | 20G |
| Large | 4 | 4096 | 40G |
| Self-defined | 1 to 4 | 512 to 4096 | 0 to 40G |

The results of the first experiment are shown in Table 3.3, 3.4, and 3.5. *Time_sche*

refers to the computing resource scheduling time, and $Time\_f17$ and $Time\_cen6$ indicate the raw resource providing time with a Fedora 17 OS and a CentOS 6 OS respectively in Tables 3.3 and 3.4. As the results show, on one hand, the cluster mode achieves a better performance in resource scheduling due to the better computing capability of the management node. On the other hand, the raw resource provision time of the single machine mode is much shorter because its scale of computing resource pool is much smaller, containing only two computing nodes. In this case, the raw resources can be recycled, regenerated, and reused efficiently after they are released for each round. The results in Table 3.5 show the softlayer resource packaging time. The softlayer packing process occurs on the management node for both single machine mode and cluster mode. The process time depends on the size of the packages and the compute capability of the management node. Because the CPU in the cluster is better, the process time of the cluster is shorter. The package size of DIS, DDS and HLA differs, which leads to different process time in each mode. This indicates that based on the proposed cloud simulation scheme a simulation-run-ready environment can be completely brought up and ready for execution within 250 seconds in the worst case among the three simulation standards. Different types of distributed simulation standards, operating systems, and computing instances are automatically configured, instantiated and managed based on the simulation requirements.

Table 3.3: Computing Resource Preparation in the Single Machine Mode

| Virtual Resource Type | Time_sche(s) | Time_f17(s) | Time_cen6(s) |
|---|---|---|---|
| Mini | 2.064 | 3.445 | 3.945 |
| Small | 2.278 | 3.705 | 3.998 |
| Medium | 2.162 | 3.753 | 4.011 |

Table 3.4: Computing Resource Preparation in the Cluster Mode

| Virtual Resource Type | Time_sche(s) | Time_f17(s) | Time_cen6(s) |
|---|---|---|---|
| Mini | 1.389 | 16.871 | 17.539 |
| Small | 1.421 | 17.278 | 17.932 |
| Medium | 1.322 | 17.502 | 18.244 |
| Large | 1.433 | 17.892 | 18.577 |

Table 3.5: Packages Handling Process

| Platform Mode | Time_hla(s) | Time_dis(s) | Time_dds(s) |
|---|---|---|---|
| Single Node | 189.845 | 141.728 | 234.013 |
| Cluster | 144.672 | 97.412 | 189.844 |

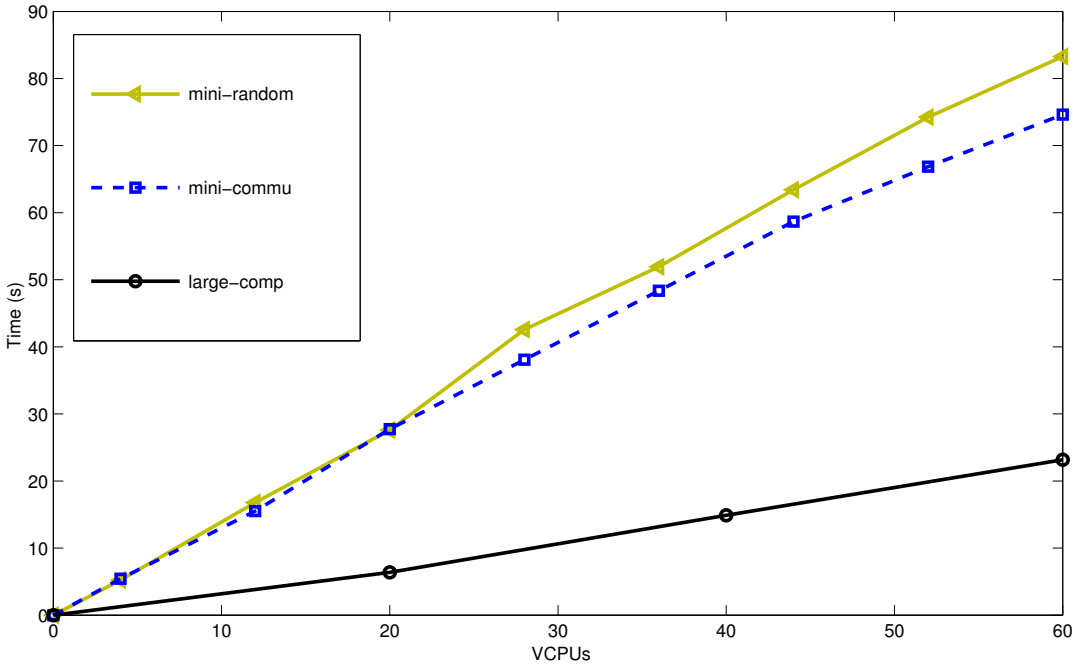



Figure 3.2: Scheduling Algorithm Performance Evaluation

The second experiment evaluated the performance of scheduling algorithms, shown in Algorithm 3, which were natively supported in this cloud scheme when simulations scaled up in the cluster mode. In this experiment, mini virtual instances and large virtual instances were introduced as computing instance examples. The scheduling time recorded in this experiment was the same as $T\_sche$, as explained in the first experiment.

The second experiment consisted in evaluating the delay on using different scheduling techniques. As depicted in Figure 3.2, when simulations scaled up, the time of scheduling increased in an almost linear trend, except for some slight fluctuations caused by background load. The background load originated from the cloud system, which had to continue running its services for periodically detecting and updating the status of the physical machines that composed the environment. Such services coordinated the resources by observing and recording their power status, network availability, current free RAM, and other characteristics. As the results show, Computation-intensive scheduling (from line 25 to line 35 in Algorithm 3) achieved the best performance, as it minimized the rounds of resource scheduling. As for random scheduling (from line 4 to line 14 in Algorithm 3) and communication-intensive scheduling (from line 15 to line 24 in Algorithm 3), the latter presented overall performance with an improvement of 10 percent. This difference was generated because in the communication-intensive scheduling, the search queue for available resources in the resource queue was shorter – the first available physical host in the resource queue was selected as the destination host. For random scheduling, the average scheduling search length was $que.len/2$, which was much longer than the communication-intensive. Another reason for obtaining these results was due to the built-in patten in recording scheduling information. The Communication-intensive scheduling recorded only when one physical host was exhausted while random scheduling needed to record immediately after the conclusion of each scheduling round. For different types of scheduling algorithms in this platform, the scheduling time for one instance was less than 1.5 seconds. As described in [228], the proposed script-based solution can save modelers substantial amount of time to deploy large scale complex simulation environment compared to a manual deployment.

The third experiment addressed the energy consumption of the proposed scheme. Due to the lack of temperature sensors in the hardware, CloudSim [89] was employed to estimate and observe the performance of the energy-saving approach in the scheme. The parameters in the experiment, such as the VM migration time and VM scheduling time, were based on the real test results of the machines in the cluster, shown in Table 3.6. The simulation used a long-term computing-intensive simulation that consumed almost

Table 3.6: CloudSim Simulation Parameters

| Scale | 20 computing nodes and 1 management node |
|---|---|
| Number of VM Required | 10 (Small, Medium, and Large) |
| VM Migration Time | 23 Seconds |
| VM Scheduling Time | 1.4 Seconds |
| Rated Power | 355 W |
| Scheduling Time Interval | 60 Minutes |
| Scheduling Algorithm | Random Scheduling |

Table 3.7: Scheduling with Migration

| Round | 1 and 2 | 3 and 4 | 5 and 6 | 7 and 8 | 9 and 10 |
|---|---|---|---|---|---|
| Current VM Type | S, M | S, S | S, M | L, S | L, M |
| Migration | 1 | 1 | 2 | 0 | 0 |
| Current Power | 712.538 | 1058.662 | 1429.614 | 2485.944 | 3551.349 |
| Current Hosts in Use | 1 | 2 | 2 | 4 | 4 |

100% CPU and ran for approximately 24 hours. The type of the computing instances used in this experiment are the same as the ones listed in Table 3.2.

Tables 3.7 and 3.8 show the results of the simulation task scheduling and executing processes with and without energy-aware components. As the results indicate, the proposed energy-saving scheme can reduce large amount of energy by centralizing simulation tasks and releasing idle computing resources. In the experiment scenario, 47.45% energy is saved in total 10 hours on average. With the proposed distributed simulation based energy-aware components, the migration consumed approximately only 300 seconds due to the migration for all the sample simulation applications during the period of 10 hours. As a result, this approach is viable and beneficial for distributed simulations that run for periods of time in the scale of hours.

The fourth experiment concerned the performance loss in simulation execution that this multi-layered cloud platform may cause when compared with the native Grid platform. In the experiment, comparisons are made with the proposed simulation Cloud deployed on our different established scenarios: on the cluster mode, on the single machine mode, and on the public cloud computing instances from AWS [9]. In the cluster

Table 3.8: Scheduling without Migration

| Round | 1 and 2 | 3 and 4 | 5 and 6 | 7 and 8 | 9 and 10 |
|---|---|---|---|---|---|
| Current VM Type | S, M | S, S | S, M | L, S | L, M |
| Current Power | 1065.54 | 2487.13 | 3909.884 | 5333.634 | 6758.49 |
| Current Hosts in Use | 2 | 4 | 6 | 8 | 10 |

mode, the large virtual resource type (as shown in Table 3.2), was used to boot fifteen virtual instances on the cloud while the grid used fifteen physical hosts directly. In the single machine mode, a small computing instance type, shown in Table 3.2, was used for the cloud while the grid used the Host Computing 2 as shown in Table 3.1. The simulation application implemented in the experiment was based on the Portico RTI restaurant example [231], the same as the first experiment. In the application, the sample federate produced a controlled synthetic load, which initially involves creating a pseudo-random number $x$ following a normal distribution and then introducing a recursive procedure for $x$ turns, to exhaust the computational capability for the instance in which it ran. After the computation, the sample federates communicated with each other about their computation results. Different iterations were used to measure the simulation execution performance.

The results for this fourth experiment are depicted in Figure 3.3. Regarding the experiments in the cluster mode, which involved both communication tasks and computing tasks, the Grid outperforms the Cloud in terms of the simulation execution time: when the experiment reached 200 iteration rounds, the performance of the Grid was 3.94% better than the Cloud. The reason for the performance loss of the Cloud is the utilization of Virtualization Technology (VT) in the Integration and Virtualization Layer, according to [16]. As for the experiments in the single machine mode that only contained computing tasks, the performance of the Grid was much better: when the iteration number reached 140, the Grid performed approximately 15 times better than the Cloud. This result was obtained due to the nested virtualization for the cloud scheme in the single machine mode. As a result, an additional layer of abstraction and encapsulation in a pre-existing virtualized environment was created. This may influence the performance of executing simulations, according to the discussion and results from [24]. It is worth mentioning that for the public instance mode that contained the same computing task as the Grid in the single machine mode, the performance of the Amazon public cloud

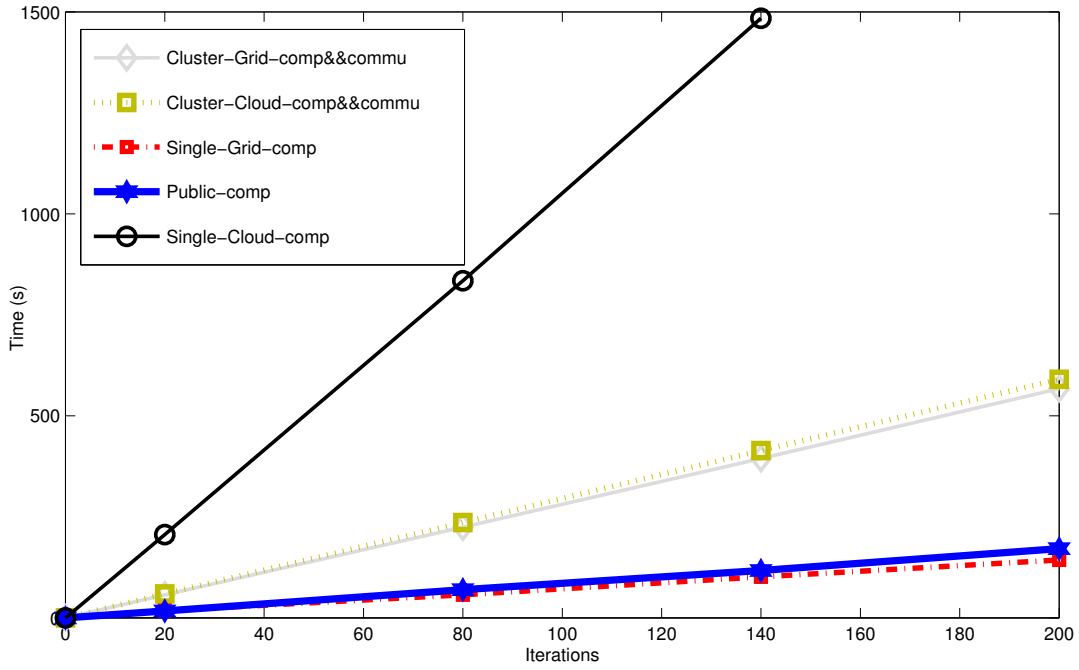



Figure 3.3: Simulation Performance between Cloud Platform and Grid Platform

instance was only 19.8% slower than the latter when the iteration round reached 200, which indicates that the public cloud is very promising for holding computation-intensive distributed simulations.

Based on the results of the aforementioned four experiments, we can observe that the proposed multi-layered cloud simulation platform enables distributed simulations in the cloud environments, providing elasticity, automatic management of diverse resources, fast deployment, security, and reduction of energy costs. There is little performance loss in terms of the simulation execution time when compared with the native Grid platform, mainly due to the delays caused by the virtualization of the Cloud. However, this overhead is acceptable due to the benefits of Cloud Computing listed in the early sections of the chapter.

## 3.3 Conclusion

In this chapter, a layered cloud platform was proposed for the distributed simulations, which performs as a prototype for the Cloudlet-based offloading platform. Concerning usability, energy consumption, security, reliability, and elasticity, relevant components

are implemented to support fast deployment, to ease the management of underlying resources, to reduce energy usage, and to enable fine-grained resource handling during runtime. The design of a multi-layered scheme in different scenarios was described. In the experiments, the performance of this cloud simulation platform was evaluated and discussed in detail. Based on the experimental results, we conclude that the use of cloud technologies is a promising method for facilitating distributed simulations and such computation-intensive offloading tasks, especially when the network environment presents greater efforts towards optimization and performance.

# Chapter 4

# A Task-centric Mobile Cloud-based System to Enable Energy-aware Fine-grained Offloading

To cope with the challenge–on enabling energy-efficient offloading, in this chapter, a task-centric Cloudlet-based MCC system is proposed. The traditional device-centric Cloudlet schemes are restricted to passively tracing the device-level mobility, performing the device-clone local offloading execution and a one-on-one device-instance mapping on the remote Cloud. Thus, in the large-scale multi-user offloading scenario, only a small portion of users can get access to the local offloading resources while a considerable number of offloading requests are rejected or forwarded to the remote Cloud DC. These offloading requests may severely suffer from the long distance communication overhead due to the lack of the local computing capability. Also, the traditional device-clone based resource management schemes produce unnecessary resource preparation overhead. The overhead is introduced by preparing the task clones that are out of the local user interests and are never triggered. Moreover, the device-instance-based Cloud resource management is not energy-friendly as the Cloud cannot shutdown scheduling policies due to the running device. Compared to the conventional device-centric solutions, the task-centric Cloudlet actively traces and analyses the local task execution and perform fine-grain task-level scheduling. The offloading resources can be scheduled based on the priority of local task interest. This fine-grain task-level scheduling can efficiently improve the throughput of the resource-restricted Cloudlet and reduce resource preparation delay. This chapter concentrates on the fine-grain Cloudlet-based resource management, which

targets on the resource limitation and preparation overhead issues on the Cloudlet and considers the task-level energy-aware offloading collaborative scheduling on the Cloud.

The organization of the remaining sections is as follows. In Section 1, the energy-aware resource scheduling model is formulated with the resource scheduling algorithms on the Cloudlet and remote Cloud. In Section 2, the proposed collaborative model is validated, and experiments are implemented. In Section 3, the chapter is summarized in the conclusion, contribution, and future direction discussion.

## 4.1 Task-centric Cloudlet-based MCC

In this section, the proposed system is presented with discussions of the primary system element functioning and behavior in the offloading process. Besides, the offloading execution workflow is further demonstrated in the layered Cloudlet-based system.

### 4.1.1 System Components

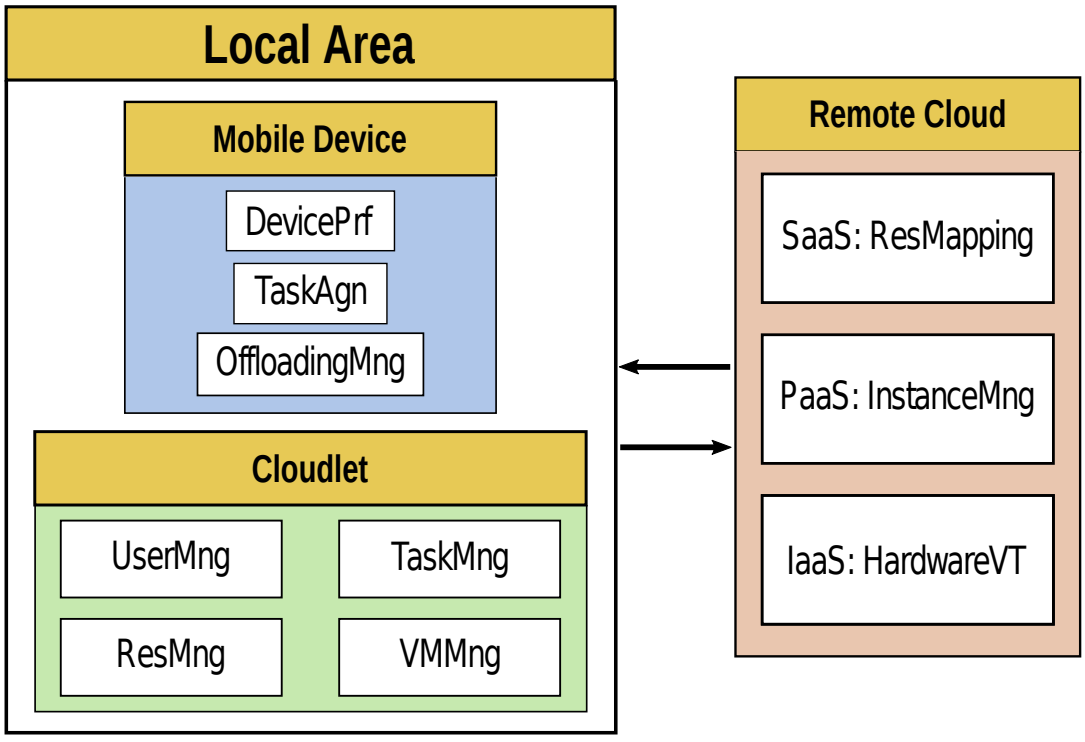



Figure 4.1: Task-centric Cloudlet-based Offloading System

The proposed alternative Cloudlet-based system consists of mainly four functioning components based on two layers – the mobile devices and the Cloudlets that constitutes the local layer; the remote Cloud resources that makeup of the remote layer, as shown in Figure 4.1.

### Mobile Devices

The mobile devices are defined as the computing capacities which can trigger and execute offloading tasks in the coverage of local Cloudlets. To enable the service-oriented Cloudlet-based offloading process, the mobile devices are required to perform registration, partial execution and profiling functions in the proposed scheme. The registration occurs once the mobile devices enter the local zone discovered by the Cloudlet or other mobile devices, through which necessary device information such as task pattern, computing capacity, battery lifespan and network-related metadata is synchronized with the central Cloudlet manager.

### Cloudlets

The Cloudlet comprises the core functioning components in the task-centric offloading system. These elements directly handle the discovery, coordination, and management of local mobile devices; monitor, arrange and prepare local offloading resources. As an intermediate bridge, the Cloudlet also communicates with remote Cloud resources and bridges the offloading channels in the case that Cloudlet resources are overloaded.

### Remote Cloud

The remote Cloud components are designed based on the definition from the standard [190]. These resources are referring to the public computing services provided by companies such as Amazon AWS, Google AppEngine, and Microsoft Azure.

In the proposed system we adopt the private Infrastructure as a Service (IaaS) architecture as the remote Cloud resources and implemented with the help of OpenStack virtualization toolkit [222] on a cluster of machines. We assume that the private Cloud can support literary "unlimited" computing utility as supported in the public Cloud datacenters. In terms of its function, the remote Cloud focuses on the offloading execution process that is redirected from the Cloudlets. Based on the task request type, the offloading tasks from Cloudlets are filtered and mapped to corresponding computing units by the Cloud Task Manager, and the results are returned accordingly.

### Networking

The network configuration in the proposed Cloudlet-based task-centric system is mainly divided into two aspects: the local wireless network and the remote wired network. We assume the local range mobile devices are connected via WLAN to the Cloudlet in the scope of the large-scale multi-user offloading scenario, where the limited cellular bandwidth cannot correctly scale up due to the high pre-investment infrastructure cost. The WAN level communication between Cloudlet and the remote Cloud are through wired network since both of the resources are stationary.

## 4.1.2 Offloading Workflow

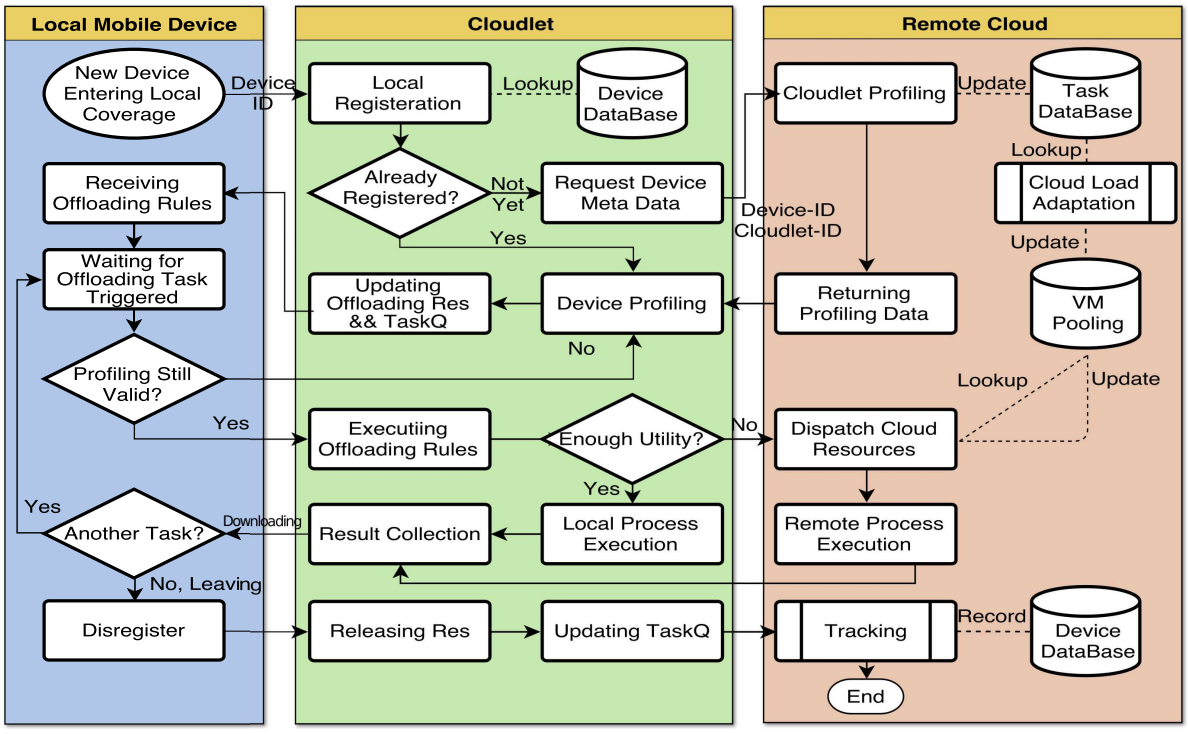



Figure 4.2: Cloudlet-based Offloading Execution

The proposed Cloudlet-based offloading workflow is shown in Figure 4.2, which includes the local mobile devices, Cloudlets, and the remote Cloud resources. The process starts when the mobile device enters the local Cloudlet coverage. It registers the device metadata and sends the test profiling message to the local Cloudlet. If no registration information can be found locally in the database, the Cloudlet will search the device-

corresponding remote Cloud resource via which to obtain the device remote Cloud-related profiling data. At this stage, the Cloudlet returns the local and remote profiling data to the mobile device and updates the computing utilities, ready for executing offloading tasks. During the offloading execution, based on the profiling data, the mobile device schedules the task execution path, triggers and shifts the remote execution components to the Cloudlet. The Cloudlet checks the availability of the computing utility, processing the offloading task locally or dispatching the requests to the remote Cloud. The Cloudlet collections the local or remote processed results and returns the results to the mobile device until all sub-offloading tasks are accomplished. In the end, the Cloudlet releases the task computing units when necessary after the device leaves.

## 4.2 System-level Offloading Formulation

Based on the aforementioned MCC components and offloading scenarios, the proposed task-centric Cloudlet-based energy framework is formulated in this section, which presents the task module, the computing module, and the communication module in details.

### 4.2.1 Task Evaluation and Execution

In order to enable MCC-based offloading, application tasks are first required to be partitioned and profiled so that decision-making can be performed to trigger proper energy-efficiency execution, based on these processed data. According to the offloading execution flowchart shown in Figure 4.2, the system task model involves the task energy evaluation model on the mobile aspect and the task execution model on the offloading resources, in which the former concerns the specific task execution path of the corresponding mobile device and the latter caters the multi-user task execution scenarios in the multi-device scope.

The device-based task energy evaluation model establishes the offloading execution path on the single task level. In order to evaluate the task energy efficiency, the task execution process is represented by the execution DAG, where the vertex stands for the processing of an atom sub-task while the edge direction indicates the sub-task dependencies. Given the first node that initiates the task application and the last node that returns task results, each topological ordering between them suggests a possible execution path. Each vertex is assigned the value that equals to the energy cost of accomplishing that

single sub-task $t \in T$ from the perspective of mobile devices, in the case of mobile device processing and remote processing respectively as shown in Equation 4.1.

$$DAG(ver, t) = \begin{cases} mComp(t) & Mobile\_proc, \\ m\_idle * rComp(t) & Remote\_proc \end{cases} \tag{4.1}$$

In the mobile process case, the sub-task is executed by the mobile computing utility, utilizing mobile computing functioning model $mComp()$(the details will be illustrated in the following Computing Energy Module section). As to the remote processing situation, the mobile device maintains in the idle status, waiting for the remote processing, which leads to the mobile-based energy cost that equals to the state holding overhead.

Besides the task process energy cost presented in the vertex, the task communication is not necessarily to be free. The sub-task communication energy consumption may sometimes surpass its process counterpart in the data-intensive operations such as scanning, encoding, and decoding. To manage the communication overhead introduced by the sub-task dependencies, weighted values are calculated and appended to the edges between vertexes, indicating the total energy cost of transmission the task-dependent data, as shown in Equation 4.2.

$$DAG(edge, t) = \begin{cases} mComm\_up(t) & M2C, \\ mComm\_down(t) & C2M \end{cases} \tag{4.2}$$

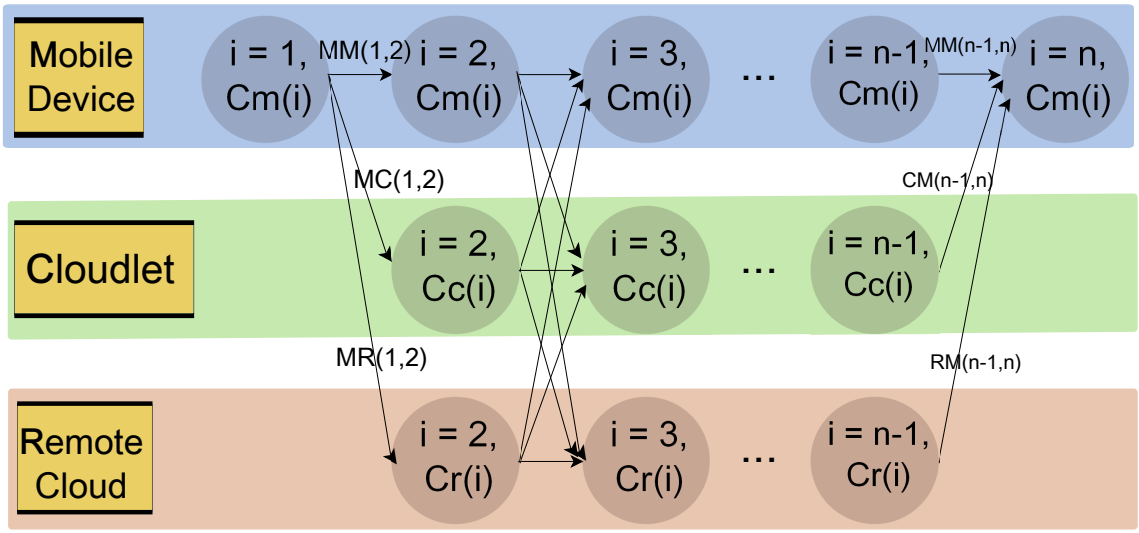



Figure 4.3: DAG Example with Linear Dependency-based Tasks

Figure 4.3 illustrates a DAG example where a task is divided into $n$ atomic tasks with linear dependency. As shown in the figure, the vertex value $Cx()$ represents the computing cost of that atom task $i$ on the mobile device $Cm(i)$, Cloudlet $Cc(i)$, and remote Cloud $Cr(i)$, respectively. The edge values describe the communication cost between two dependent tasks, where $XY(i, i+1)$ represents the communication cost between the task $i$ and the task $i+1$ from the resource $X$ to the resource $Y$. In this case, each path from the node $N_{i=1}$ to the node $N_{i=n}$ denotes a possible offloading solution and the path with minimal cost is the target solution.

Concerning the energy-aware offloading execution on the Cloudlet and the remote Cloud resources, we introduce a threshold-based task queueing model to cater the offloading resource scheduling, aiming at maintaining a list of run-ready hotspot application servers to meet the requirements of time-constraint offloading tasks and reducing the energy cost of the external resources during processing offloading requests. A Cloudlet queue and a Cloud queue describe the local Cloudlet processing and the remote Cloud group processing respectively. We use the general noun *cQue* to represent the resource scheduling behavior on the Cloudlet and the remote Cloud together. In the system queue *cQue*, the number of currently available resources is represented by the queue length *cQue.len*. Each slot in the queue stands for a computing unit with the status either occupied or empty. The occupied slot means the corresponding computing unit is busy and the empty slot means that unit is waiting for processing offloading tasks. In this case, the system load at time $t$ can be described in Formula 4.3 with the system load ratio in Formula 4.4.

$$O(t) = \sum_{slot}^{slot=occupied} slots\ in\ cQue(t) \tag{4.3}$$

$$R(t) = O(t)/cQue(t).len \tag{4.4}$$

To achieve the adaptivity, a hot threshold and a cold threshold are defined as $H, C$, as the initial scaling boundaries. The initial values of the threshold are deployed according to the work [247], in which the setup parameters are proved to be effective in the VM resource scheduling for web applications. The adaptive scaling principles are defined in the following Formulas 4.5 and 4.6, and the influence is shown in Formula 4.7.

$$h(t+1) = \begin{cases} h * 2 & O(t) = cQue(t).len \\ h(0) = 1 + C & O(t) < cQue(t).len \end{cases} \tag{4.5}$$

$$c(t) = c(0) = H \tag{4.6}$$

$$cQue.len(t+1) = \begin{cases} cQue.len(t) * h(t) & R(t) > H \\ cQue.len(t) & C <= R(t) <= H \\ cQue.len(t) * c(t) & R(t) < C \end{cases} \tag{4.7}$$

As shown in Formula 4.5, the scaling up parameter $h$ is initiated according to $C$. The value of the parameter keeps static if the number of current resources is larger than the incoming requests. Once the current resource queue is full, the scaling parameter doubles its value to increase its influence on the system until the number of current resources is larger than the incoming requests again. For the scaling down parameter $c$, we conservatively maintain it as the constant initial value, considering the time-constraint requirements of the offloading tasks. The system adaptivity is reflected by the changes of the currently available queue size as shown in Formula 4.7.

### 4.2.2 Computing Energy

To properly profile and estimate the task processing cost that is further utilized in offloading scheduling, the computing energy model is designed and implemented on the mobile devices, local and remote offloading resources.

In terms of the mobile computing energy profiling, The $mComp(t)$ function is calculated based on the device-specific profile settings and the task meta-data, according to the DVFS model, shown in Formula 4.8. The profiling setting values are defined by the device manufacturers which are accessible via the system info file, on which different CPU frequency level is bound to one distinct electric current value. As for the task meta-data, the information can be obtained during the application installation, which includes the load $t.load$, CPU usage $t.scale(level)$ of each atom sub-task.

$$DVFS(t) = P_{based} + P_{scaling}(t.Power\_level) \tag{4.8}$$

In this case, for a given sub-task $\bar{t} \in T$, the energy cost of processing on the mobile device can be defined as

$$mComp(\bar{t}) = \int_0^{\Delta t} (DVFS(\bar{t}.scale()) + Illu(\bar{t}))\Delta td(\bar{t}) \tag{4.9}$$

$$\Delta t = \frac{\bar{\mathrm{t}}.load}{\bar{\mathrm{t}}.scale(level)} \tag{4.10}$$

where the $Illu(\bar{\mathrm{t}})$ calculates the energy consumption on the screen. Given the fact that sub-tasks are only responsible for the atom function with stable CPU utilization and short processing time, the sub-task execution integral formula can be simplified as the following linear expression and as:

$$mComp(\bar{\mathrm{t}}) = \Delta t(DVFS(\bar{\mathrm{t}}.scale(avg)) + E_{Illu()} \tag{4.11}$$

where $\Delta t$ equals to the sub-task execution time and $\bar{\mathrm{t}}.scale(avg)$ is the average CPU level that can be directly obtained from the meta-task information. In this case, depending on the device energy settings and task meta information, the task execution energy cost can be statically estimated without real-time profiling, thereby reducing the overall offloading overhead. Concerning the accuracy of the simplified estimation, the estimated energy model is evaluated in the real-time execution, discussed in the performance evaluation section. The result presents an overall 95% similarity which is considered to be solid as the proof of the next offloading scheduling process.

In the case of the computing energy consumption on Cloudlets and the remote Cloud, the power estimation is depending on the three-layered Cloud model defined in [190] and [199] where the Cloud-based services and functions are layered as IaaS , Platform as a Service (PaaS) and Software as a Service (SaaS) upon the raw resources. In this case, given a raw physical computing server $pH$ with the number of VMs as $nofV$, the computing energy cost during the time interval $\Delta t$ can be defined as:

$$pHComp(\Delta t) = \Delta t(DVFS(curLoad/(MaxC * nofC)) \tag{4.12}$$

$$curLoad = \sum_{n=1}^{nofV} VM.scale(avg) \tag{4.13}$$

where the physical host $curLoad$ value sums up the computing loads in all the managed virtual instances and the $MaxC * nofC$ equals to the maximum CPU clock of the server with all available cores. In this case, the energy gain of the proposed task-centric platform can be defined as follows when comparing to the traditional device-based solutions:

$$\Delta E = \sum_{n=1}^{nofH} (\Delta pHComp(\Delta t)) \tag{4.14}$$

given the fact that the DVFS CPU frequency/energy ratio on the based component remains considerably smaller than the scaling part, as shown in the spec models used from the work [89], assembling tasks in the proposed offloading framework can achieve higher frequency/energy ratio which leads to a global energy cost reduction, compared to the device-based solution under the same server setting, task load, and VM allocation policy.

### 4.2.3 Network Energy

As discussed in the previous task models, the extra network energy cost is introduced by the offloading sub-task dependency. Unfortunately, the energy cost of the communication overhead is hard to be evaluated statically, due to the dynamically changing network environment. On the one hand, the organization of the network complicates the energy consumption estimation, which involves wireless LAN between mobile devices and Cloudlets, WAN level network between Cloudlet and remote Cloud, cellular network between mobile devices and base stations. On the other hand, the network condition may be influenced drastically by the mobility of the devices, the current load and signal interference, which will lead to wrong offloading decisions if the profiled network information cannot correctly reflect the currently load changed.

To estimate the offloading communication energy cost in the dynamic changing environments, we proposed a test message based solution to adjust to the network variations. In the registration phase, a test message is processed between the mobile device and the Cloudlet to obtain the RTT on the local level. The RTT is utilized to estimate the energy cost variation that is introduced by the network environment, as shown in Formula 4.15. The $d.Ecost$ is the base WiFi active energy cost recorded in the device metafile.

$$\Delta E = (RTT/2) * d.Ecost - d.cur * d.Ecost \tag{4.15}$$

According to the device meta-data, the Cloudlet connects to the corresponding remote-Cloud resource and records the time cost on the remote level. In the task offloading phase, the device $d$ arranges profiling updates when the current change passes the limit $\delta$, as shown in Formula 4.16:

$$d.update(\bar{t}) = \begin{cases} d.cur & |\Delta E| < \delta \\ d.new & |\Delta E| > \delta \ or \ \bar{t} > interval \end{cases} \tag{4.16}$$

where $\delta$ is defined as the minimum value that leads to a different offloading execution path. In this case, the energy consumption of the atom task dependency processing $mComm(\bar{t})$ can be calculated as:

$$mComm(\bar{t}) = \Delta t * mComp(\bar{t}.proc(\bar{t})) + d.update(\bar{t}) * d.Ecost \tag{4.17}$$

where $mComp(\bar{t}.proc(\bar{t}))$ is the average computing level to process the data from internal storage or RAM to the NIC and $d.update(\bar{t}) * d.Ecost$ is the energy cost on the network channel.

### 4.2.4 Cloudlet – Offloading Scheduling Execution

The Cloudlet performs the crucial role in the mobile task offloading. During the task offloading processing, the Cloudlet is responsible for organizing local devices, handling incoming task requests, preparing local offloading resources, communicating with remote Cloud resources and delivering offloading results.

As shown in Algorithm 5, the Cloudlet first registers the devices and incoming tasks in the task queue in line 6. To accommodate to the network condition variation and current Cloudlet load, the communication channel between the devices that trigger offloading requests and the Cloudlet is calculated by sending test messages while the connection status between Cloudlet and remote Cloud is either already cached or re-evaluated if the data is out-of-date, as shown in line 7 and line 8. After the network and computing resource data collection process completes, the Cloudlet replies to the devices with the profiling results, ready to trigger the task scheduling process. The scheduling is handled as shown in the loop between line 9 and line 23, based on the DAG task model discussed in the previous sections. Firstly, the offloading task is divided into atom tasks where the vertex represents the sub-tasks, and the edges imply the execution logic. Then, the processing load of each atom task is calculated on the Cloudlet, and remote Cloud resources and the weight of the edges is measured accordingly based on the profiling data.

At this stage, an execution path can be generated with the minimum energy cost according to the vertex processing value and edge weight. Given $n$ as the number of atomic tasks, according to the DAG definition, the number of vertexes is $3n - 4$, as

**Algorithm 5:** Cloudlet-based Offloading Scheduling Algorithm

```
1  Require: Available Tasksets : t_set,
2  Device : d,
3  Cloudlet : C_l,
4  Remote Cloud : C_r
5  # Caching && Resource Preparation Procedure:
6  t_que = select(tList : t in t_set)
7  net_l = net.test(d, C_l)
8  net_r = C_l.inQue(d) ? C_l.lookup(d) : C_l.forward(d, net, C_r)
9  for each t in t_que do
10     t.DAG = C_l.get(t.partition())
11     for each ver in t.DAG do
12         ver.comL = C_l.comp(res.l, d)
13         ver.comR = C_l.comp(res.re, d)
14         for each edge in ver do
15             edge.outL = C_l.comm(net_l, verList : ver in t.ver.next())
               edge.outR = C_l.comm(net_r, verList : ver in t.ver.next())
16     C_l.resReq = t.DAG(C_l)
17     C_l.cacheUpdate()
18     if C_l.resNotFull() then
19         C_l.resUpdate(t.cache, res.load)
20         Return: t.DAG().getExecPath(ver, C_l)
21     else
22         C_l.forward(resReq, C_r
23         Return: t.DAG().getExecPath(ver, C_r)
24 # Execution Procedure:
25 while (t = t_que.next()) != null do
26     if t.path() == C_l then
27         vm = C_l.mapping(t, res)
28         vm.lock(); t.exec(); vm.release()
29     else
30         vm = C_l.getRes(C_r, t)
31         vm.lock(); t.exec(); vm.release()
32     t_que.add(record)
33     t_que.update()
34     C_l.cacheUpdate()
```

shown in Figure 4.3. Accordingly, the number of edges for the linearly coupled situation can be calculated as $9n - 21$. Assuming that these tasks are loosely coupled, we define a constant value $c$ to record the number of extra task dependencies compared to the linearly coupled situation. In this case, the number of total edges can be counted as $9n + 9c - 21$, and the computing complexity is $O((9n + 9c - 21) * log(3n - 4))$, which converges to $O(n * log(n))$. In the best cases that the network parameters are stable, the rescheduling of the linearly coupled tasks can reuse the previous execution path, which can lead to a linear $O(n)$ execution efficiency. Given that the processing occurs on the mobile terminal side, the scheduling of offloading tasks does not decrease the computing throughput of Cloudlet resources. After the schedule, the offloading is ready to be triggered. The Cloudlet either maps the sub-tasks to locally cached resources or forwards to the remote resources, based on the available utility. When the offloading results are returned, relevant resources are released for further tasks. In addition to the offloading task execution process, the Cloudlet also updates the cached instances when the list of hotspot applications changes.

### 4.2.5 Remote Cloud – Resource Scheduling Execution

Based on the MCC execution flowchart shown in Figure 4.2, the remote Cloud is obligatory to handle offloading tasks forwarded by the intermediate Cloudlets when local resources cannot support the current offloading demands. As shown in Algorithm 6, the Cloud Manager first queues all incoming task sets and periodically invokes the Task Managers to collect and the system-level corresponding task loads based on the cluster level monitoring results, which is further used to estimate the scale of required resources in the future, as shown in line 5 to line 8. In the next step, the Cloud Manager delegates incoming tasks to the homologous Task Managers that further arrange available virtual instances to process the offloading requests. In addition, the Task Managers evaluate the tasks loads of the controlled resources, performing scale-up and scale-down operations to avoid SLA violations and to maintain system-level energy efficiency, shown in line 9 to line 16.

## 4.3 Performance Evaluation and Discussion

In this section, the performance of the proposed system model is evaluated, compared and discussed on real executions and simulations. The environment configuration is first

**Algorithm 6:** Task-centric Cloud-Level Offloading Scheduling Algorithm

```
1  Require: Incomputing Tasksets : t_que,
2  Cloud Manager : C_mgr,
3  Task Manager : t_mgr,
4  while true do
5      if C_mgr.clock() == Interval then
6          for each t_mgr in C_mgr do
7              t_mgr.calculate(res)
8              t_mgr.estimate(load)
9      while (t = t_que.next())! = null do
10         C_mgr.dispatch(t)
11         t_mgr.map(t, pm, vm)
12         t_mgr.record(load)
13         if t_mgr.estimate(load) > threshhold.hot then
14             t_mgr.init(vm, res)
15         if t_mgr.estimate(load) < threshhold.cold then
16             t_mgr.release(vm, res)
```

discussed, then the experiment details and results are presented and discussed.

### 4.3.1 Environment Setting

The prototype of the system is implemented based on the client-server paradigm to validate the proposed task, computing, networking, and energy model, and the performance is further evaluated on the Cloudsim simulator [89] in the large-scale execution scenarios with comparisons with the previous device-based solutions. The resources and parameters used in the experiments are shown in Table 4.1. The values in $Simulation\_Host$ and $Simulation\_VM$ are used in the Cloudsim simulator in the large-scale execution evaluation scenarios. The settings in the $Validation\_Server$ and $Validation\_Client$ are utilized to observe the energy prediction model, offloading execution efficiency and energy efficiency in the real executions.

Table 4.1: General Cloudlet and Cloud Environment Set

| **Resource** | **Parameter** | **Value** |
|---|---|---|
| Simulation_Host | Type | HpProLiantMl110G4 |
| | | HpProLiantMl110G5 |
| | Schedule_Interval | 300 sec |
| | MIPS | 1860, 2660 |
| | PES | 2 |
| | RAM | 4 GB |
| | Bandwidth | 1 Gbit/s |
| Simulation_VM | Type | Micro, Small |
| | MIPS | 500, 1000 |
| | PES | 1 |
| | RAM | 0.63 GB, 1.7 GB |
| | Bandwidth | 100 Mbit/s |
| Validation_Server | Type | Laptop |
| | CPU | Intel 4700MQ |
| | RAM | 16 GB |
| | WLAN | 802.11n |
| Validation_Client | Type | Moto G4 |
| | CPU | Quad-core Cortex-A53 |
| | RAM | 2 GB |
| | WLAN | 802.11n |
| | Network | LTE Cat4 150/50 Mbit/s |

### 4.3.2 Performance Matrix

In the experiment scenarios, the energy cost offloading model is first validated by comparing to the real execution results. Based on the energy model, the offloading efficiency is concerned and tested for the mobile devices in the second experiment. The third experiment set concentrates on the service provisioning throughput on the Cloudlets, and the last experiment set evaluates the execution efficiency and energy cost of the remote Cloud resources.

#### System Model Validation

The first experiment is implemented to test the validation of the proposed system model. We adopted the task benchmark in the work [101] defined by the Binary-trees

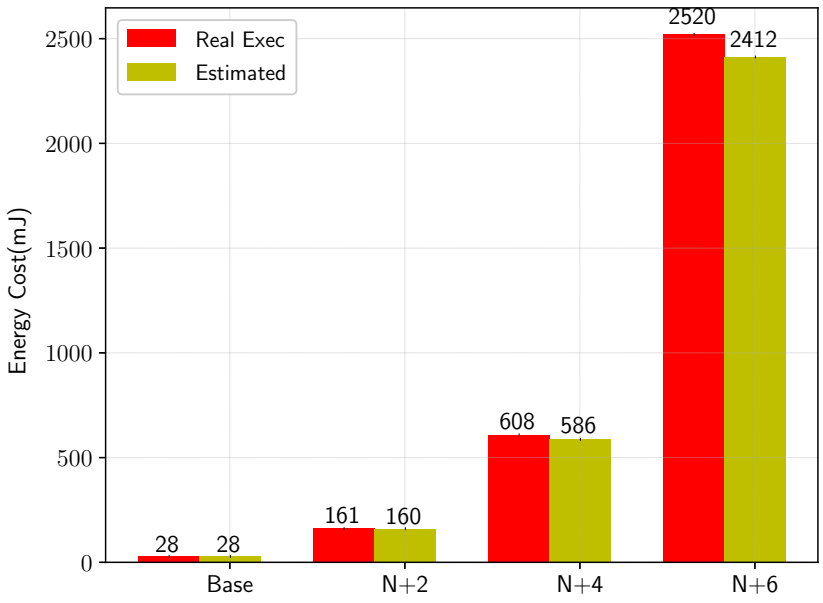
Table 4.2: Offloading Validation Task Set

| **Type** | **Load Level** | **Exec Time (ms)** |
|---|---|---|
| Computing_Server | Base | 1.3 |
| | N+2 | 7.4 |
| | N+4 | 37 |
| | N+6 | 132.4 |
| Computing_Client | Base | 52.7 |
| | N+2 | 265.5 |
| | N+4 | 1206.1 |
| | N+6 | 5590.1 |
| Communication | Base | 262.3 |
| | 256Kb | 654 |
| | 512Kb | 1733 |
| | 1Mb | 7096.3 |

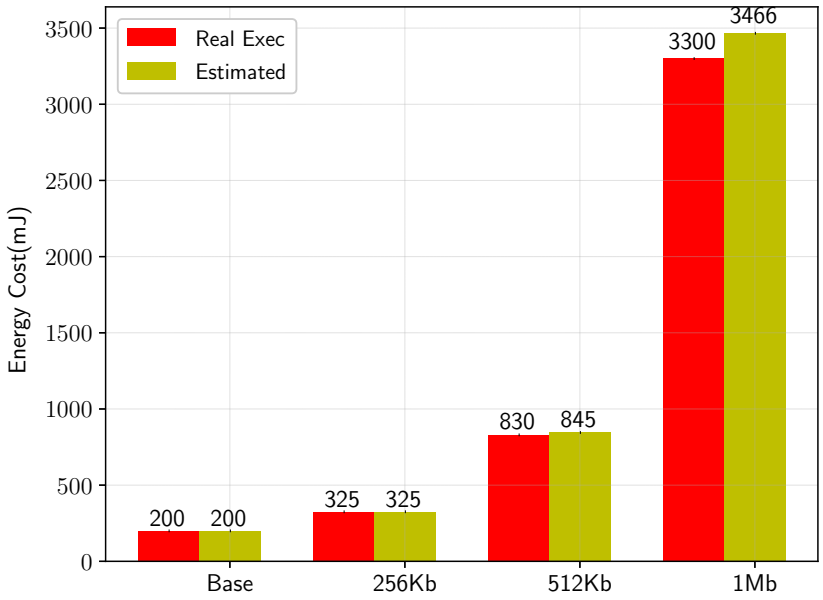
routines, [1] where different height binary trees are allocated and de-allocated to mimic the common search and sort computing behaviors in the mobile applications. To evaluate the communication behavior, a TCP-based text copying process is attached to different heights of the created binary tree with different file size. The task load and execution time are recorded in Table 4.2 while the comparison between real-execution energy cost and proposed model-based energy prediction is depicted in Figure 4.4 respectively. The highest load set standard is defined to the value under which JVM will begin to perform garbage collection routine. In this case, the external background load influence from the compiler level is maximally avoided.

The results of the first experiment are shown in Figure 4.4. In terms of the computing task set in Figure 4.4a, our proposed energy estimation can achieve reasonable accuracy when comparing to the real execution. For the baseline scenario and N+2 scenario, 99 percent accuracy can be achieved due to the stable usage of computing dynamic voltage level. As to the high load scenario with complexity N+4 and N+6, the energy estimation model can still achieve 95 percent accuracy under the influence of computing load fluctuation. Similar behavior is observed in the communication task set Figure 4.4b. The baseline and low load scenario involve low variation which leads to more accurate prediction, while during the high load scenario the energy cost is impacted by the unpredictable signal interference, package loses and others. In the highest load scenario, the

[1] Source code from: Gouy, Isaac. The Computer Language Benchmarks Game. http://benchmarksgame.alioth.debian.org (28/06/2017).




(a) Computing Task Validation




(b) Communication Task Validation

Figure 4.4: Task and Energy Model Validation

proposed model can still achieve 95 percent accuracy. The results indicate that the proposed energy cost model can properly reflect the energy cost behaviors in computational tasks and communicational tasks for the mobile elements.

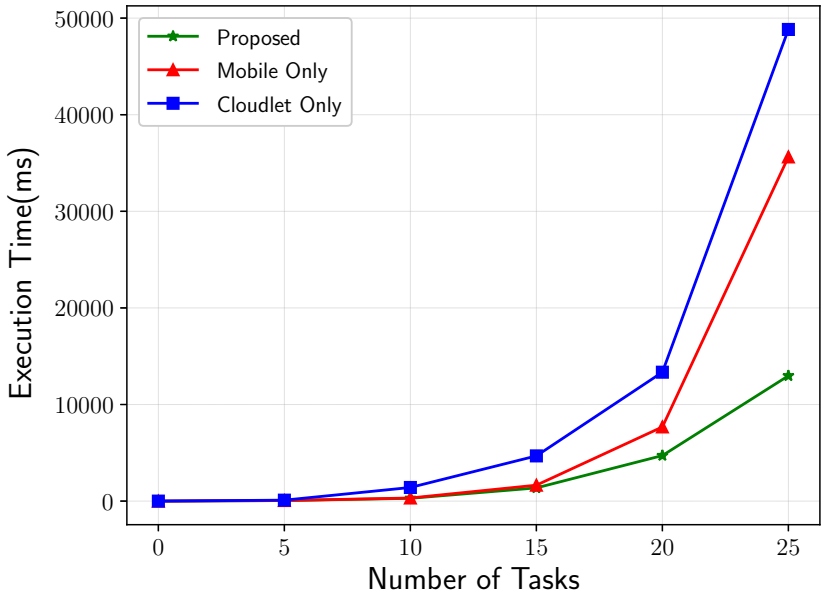
**Offloading Efficiency on Mobile Elements**

The second experiment concerns the offloading efficiency from the perspective of the mobile devices, in which the execution efficiency gain and energy cost reduction is evaluated. The task sets used in the execution scenario are from the real execution data the same as the first experiment. The offloading performance of the proposed task-centric Cloudlet is compared to the mobile-only scenario that all tasks are solely executed on the mobile devices and to the remote-only scenario in which all tasks are shifted to the Cloudlet and executed. The parameters used are defined and presented in Table 4.1.

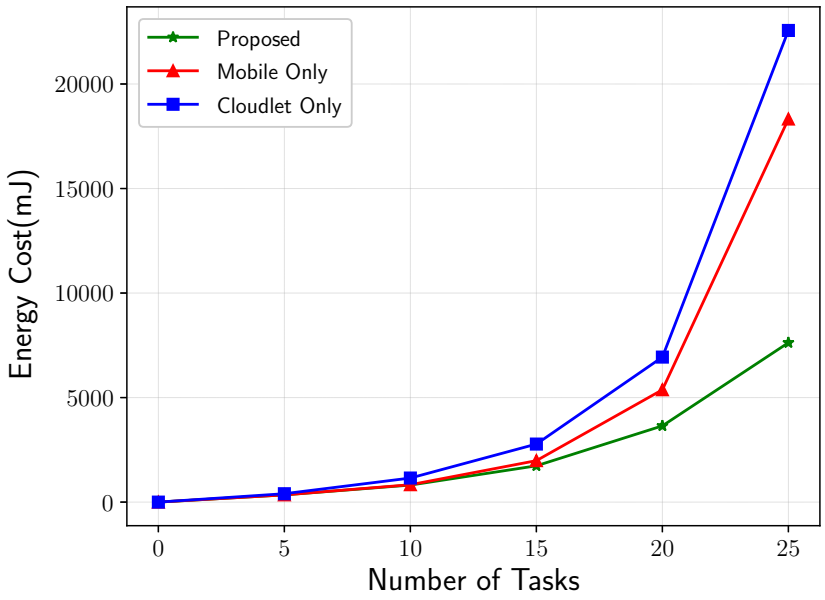
Figure 4.5 shows the offloading efficiency results in terms of the task execution time and the mobile device energy consumption. In terms of the task execution time (shown in Figure 4.5a), the proposed offloading scheme can achieve 65.35% and 73.43% shorter overall completion compared to mobile-only and Cloudlet-only execution cases. Similarly, as to the energy cost of the mobile devices (shown in Figure 4.5b), the proposed task-centric model can reduce 58.39% and 66.21% energy cost compared to the mobile execution and Cloudlet execution respectively. The performance gain is attained through maximizing computational gain from the computing server that performs 20 times faster than the mobile terminal, as shown in Table 4.2. However, migrating all tasks to the Cloudlet may not lead to a better performance than the original mobile-based sole processing due to the communication overhead introduced by the task dependency. The communication process involves high computing cost operations that read data from internal storage to the random access memory, copy task output to the network interface, data transmission, and communication channel maintenance. This observation is also proved and presented in our proposed evaluation scenario – the Cloudlet-based solution consumes 20 percent more energy and requires 35% extra processing time.

**Cloudlet Throughput Evaluation**

In the third experiment set, the Cloudlet-level processing performance is evaluated. The proposed task-centric offloading scheme is firstly compared to the traditional device-based protocol class on the non-time-constraint tasks, to observe the queueing delay and task preparation overhead based on the total task execution time, shown in 3.*a*. Secondly,




(a) Execution Efficiency




(b) Energy Efficiency

Figure 4.5: Offloading Execution time and Energy cost on Mobile Devices

Table 4.3: Cloudlet and Task Set

| **Parameter** | **Definition and Value** |
|---|---|
| Env Preparation Time | 30 to 100 sec |
| Scheduling Interval | Gaussian (30,$\sigma = 1$) seconds |
| Sample Task Set | AbiWord, GIMP, Gnumeric, Kpresenter, PathFind, SnapFind |
| Number of Task Type | n = 120 |
| Number of Device | n = 120 |
| Cloudlet Resource Queue Length | l = 24 to 80 |
| High Priority Tasks (Devices) | 10 % Trigger Offloading Request |
| Number of High Priority Tasks | len * 0.2 |
| Number of High Priority Devices | len * 0.2 |
| Overload Ratio | n / len |
| Processed Tasks | p = Tasks Processed by Cloudlet |
| Processed Ratio | p / n |
| Device-centric | Device-based Offloading Class |
| Device-RS | Random Service Allocation |
| Device-Optimized | Heuristic Optimization |
| Device-Mobile-aware | Mobility-aware |

the proposed scheme is evaluated in the large-scale time-constraint offloading scenarios, concerning the throughput performance in 3.*b*. Thirdly, we test the performance of the proposed scheme under mobility situations in 3.*c*. To compare and test the performance, we adopted the experiment task set recorded from the device-based work [220] summarized in Table 4.3.

In the experiment set 3.*a*, the Cloudlet processing efficiency is evaluated based on the non-time-constraint tasks, focusing on the queueing delay and preparation overhead. During the offloading execution, the proposed task-centric scheme and the device-based offloading class adopt the first-come-first-serve policy to process the tasks, and all unprocessed requests are cached in the waiting list.

The experiment result of 3.*a* is depicted in Figure 4.6 and it presents an overall approximately 50% execution time reduction in the proposed solution, in a 90% trust interval. This performance is achieved by diminishing the task queuing time. By predicting and caching the high-priority task overlay in the task repository, the Cloudlet resources get a higher chance to reuse the same execution environment for the new arrival ones. In this case, compared to the device-based solutions, a more significant number of tasks can be designated and processed without the environment context preparation overhead especially in the high load scenario. Observing the value intervals of each point, it shows that the proposed algorithm can also effectively reduce the fluctuation range by merely

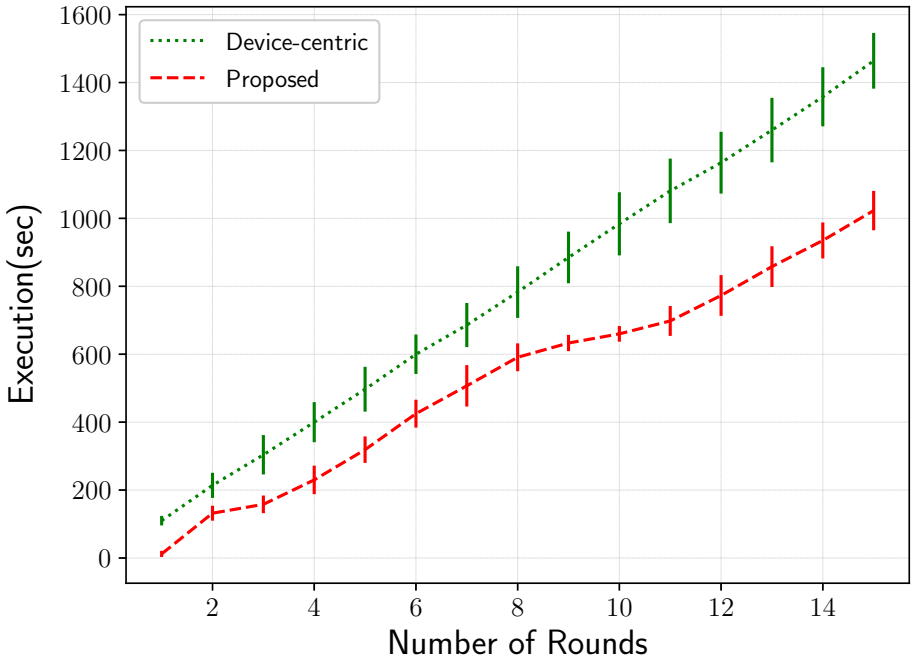



Figure 4.6: Cloudlet Caching Performance based on Different Number of Tasks

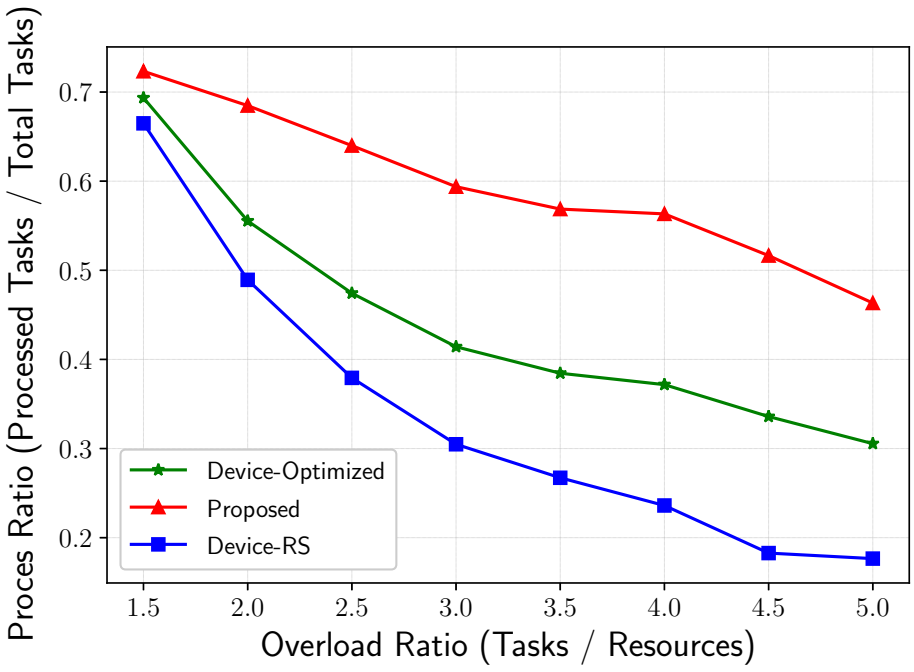
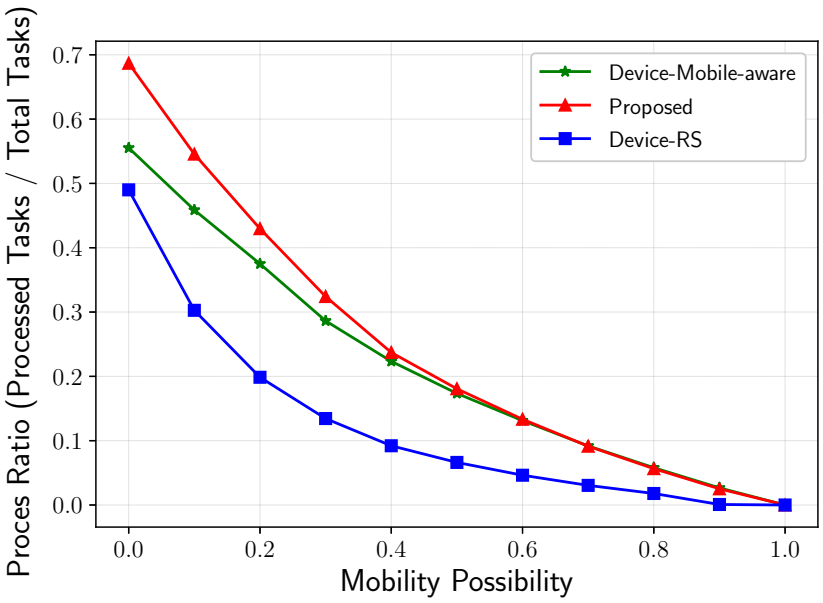
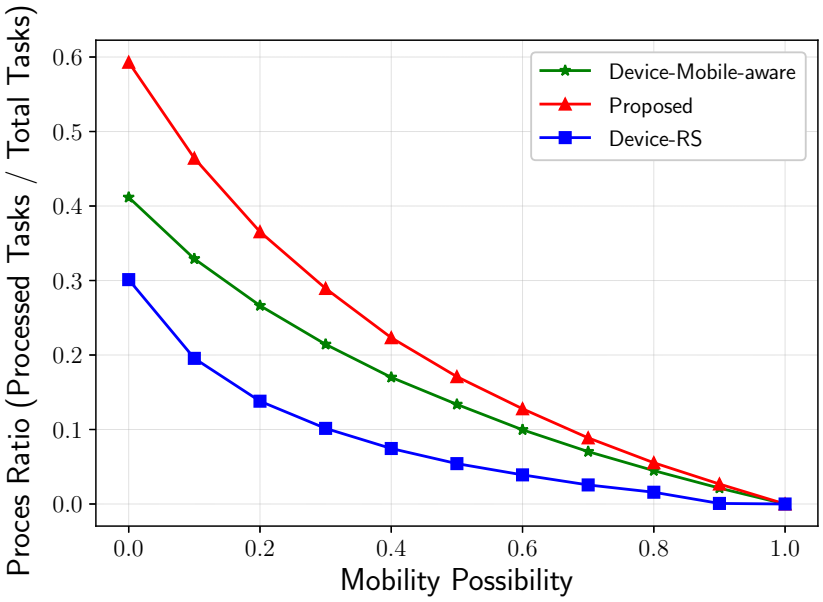


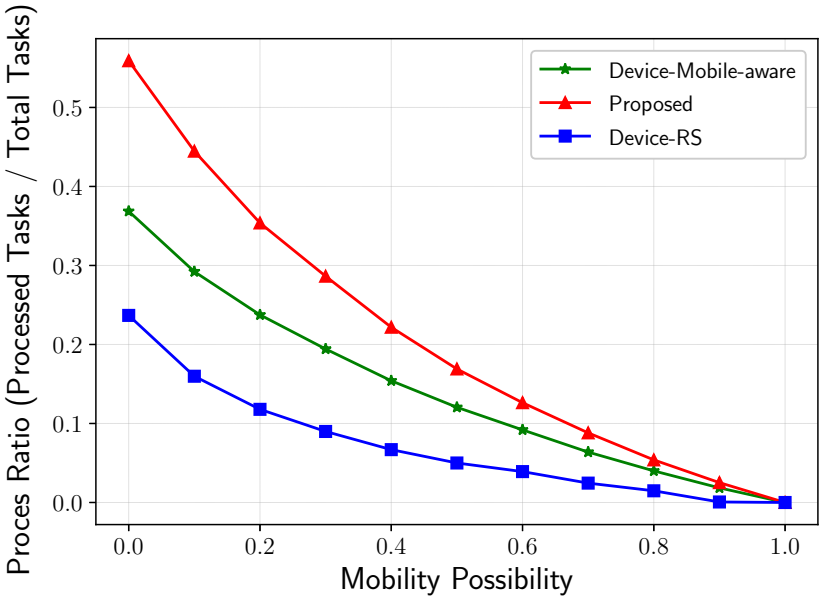

Figure 4.7: Cloudlet Processing Performance based on Different Number of Resources



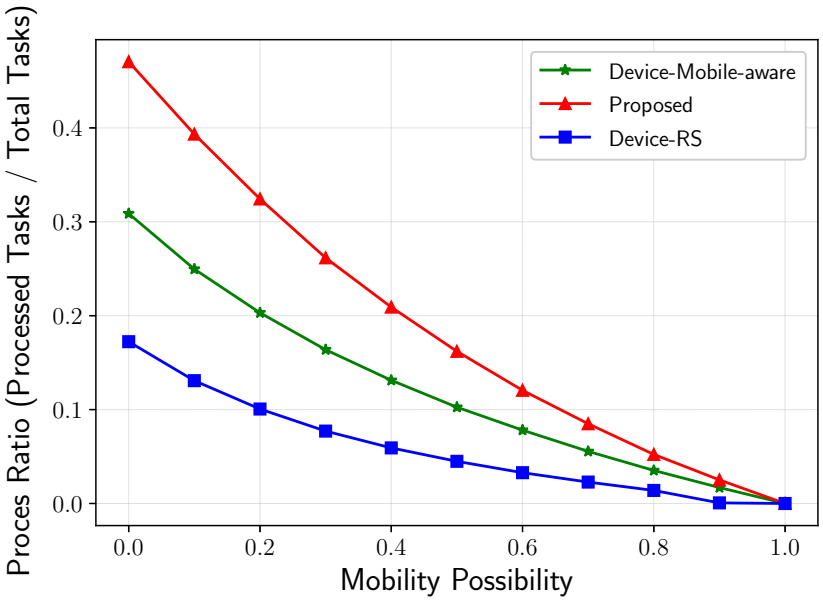

(a) Overload Ratio = 2




(b) Overload Ratio = 3




(c) Overload Ratio = 4




(d) Overload Ratio = 5

Figure 4.8: Mobility-based Throughput Evaluation on Different Mobility Possibility and Overload Ratio

adopting the execution records, comparing to the device-centric counterpart that did not maintain the global task execution knowledge.

In the experiment 3.*b*, the proposed Cloudlet offloading processing performance is examined on the time-constraint offloading tasks. We assume that all the real-time offloading tasks are time-restricted and require immediate responses from the Cloudlet (process local offloading or not). We introduce the device-based offloading scheme Random Service selection (Device-RS) as the baseline, which does not hold device information and randomly selects the devices in coverage to schedule the resources. We also introduce the device-aware heuristic offloading scheme (Device-Optimized) that keeps tracking and analyzing the local device information and properly prioritizes resources to devices with higher offloading requests. To fairly compare the performance of the proposed scheme with the Device-Optimized scheme, we utilize the same static number of total tasks and devices with the same percent of high priority tasks and devices, as shown in Table 4.3.

The result of the experiment 3.*b* is shown in Figure 4.7. The three curves all show that as with the increasing of the overload ratio, the Cloudlet processing capability decreases. Due to the lack of device and task information, the Device-RS is restricted by the length of currently available resources. Compared to the baseline, the proposed scheme and the Device-Optimized scheme offers a much higher processing throughput under the high load scenario because of the awareness of high priority devices and tasks. In addition, the proposed scheme can also achieve 9% *to* 162% times better performance, compared to the Device-Optimized scheme. This is owing to the benefits introduced by the fine-grain task-level scheduling, which enables resources to be prepared according to the priority of tasks.

The experiment 3.*c* concerns the offloading scheme performance under mobility. The run-time mobility of devices may lead to the failure of the current offloading and trigger extra resource preparation overhead. In the experiment scenario, we introduce the Device-Mobile-aware offloading scheme which can predict the mobility of devices and reserve the resources for other devices in coverage during the current offloading execution. We use the same task set as experiment 3.*c* and add the Mobility Ratio parameter in the execution.

The results of the experiment 3.*c* is shown in Figure 4.8 based on different overload situations. The initial values with 0 mobility are the same as in Figure 4.7 and all the offloading schemes converge at zero processing ratio when the mobility possibility increases to 100%. As shown in the results, the proposed task-centric offloading scheme outperforms the device-based random selection and mobile-aware counterparts due to the

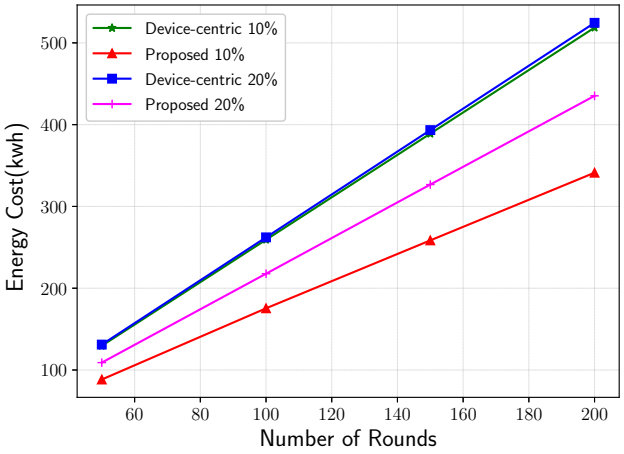
reduction of resource preparation overhead. However, the mobility of devices shows a more considerable influence on the proposed system – the Process Ratio decreases faster than the device-based solutions. This is because the proposed system lacks the tracking of device-based location data. In the future work, we will investigate the mobility-related failure detection and fault tolerance issues.

**Cloud Resource Energy and Execution Evaluation**

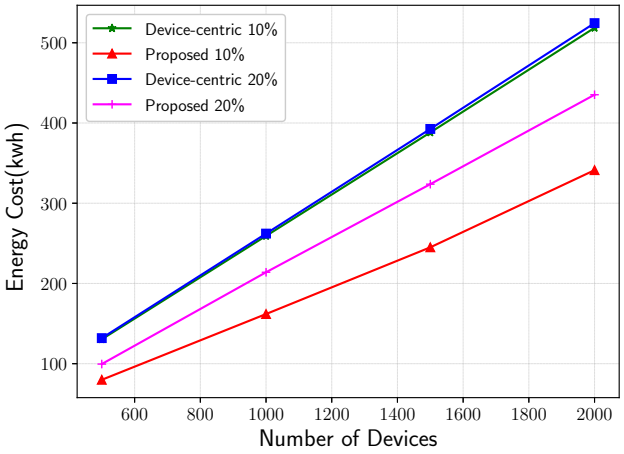
The fourth experiment set evaluates the proposed task-centric model on the remote Cloud level, observing the energy cost of executing offloading tasks. To fairly compare the proposed solution to the device-based solution, we adopted the same embedded "dvfs" and "random" VM selection and allocation policy in the CloudSim simulator to handle the mapping between physical machines and virtual instances in the large-scale execution cases. In the proposed test cases, the task execution energy cost is observed from the views of the execution rounds, the number of devices and the load density of tasks respectively. The density of tasks represents the average load in the system. The values of each CPU is randomly generated in the CPU load range 0 to 100 percent to satisfy the multi-user multi-task unpredictable execution scenarios.

The results are presented in Figure 4.9 which includes the comparison on rounds Figure 4.9a, on number of devices Figure 4.9b and task density Figure 4.9c respectively. As shown in Figure 4.9a, the proposed scheme achieves around 35% less energy cost on density level 10% and 15% less energy cost on density level 20%. A similar result can be observed on the number of devices in Figure 4.9b, where the proposed offloading system can reduce 36 percent and 16 percent energy cost separately, comparing to device-based solutions. In Figure 4.9c, the influence of task density is further shown. Due to the one to one mapping utilized in the device-based resources, idle virtual instances are required to be maintained in case of task violation while the proposed task-centric method can elastically modify the live instances according to the task load level. On the density level 30%, the proposed solution gets the same performance as device-based counterpart because no further resources can be scaled down according to the resource scaling lower boundary.

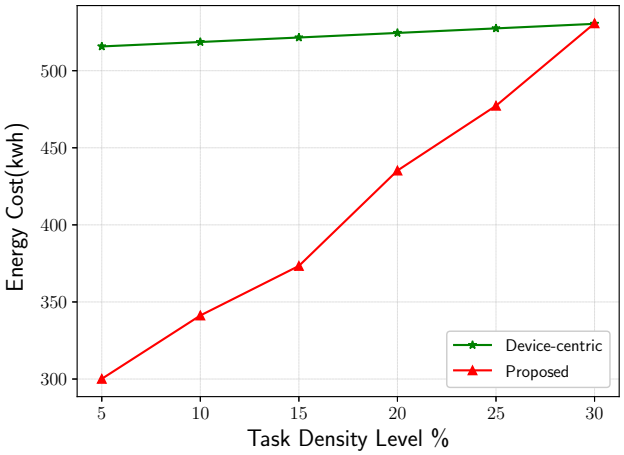
The fifth experiment set concerns the SLA violation situations on the proposed task-centric Cloud resource scheduling process. Concerning the time-constraint requirements of the offloading tasks, we assume that the offloading service SLA requires the Cloud executes the offloading task immediately once it receives the task request. Otherwise, the SLA is violated. We compare our proposed adaptive scheme to the Single Threshold




(a) Evaluation based on Task Rounds




(b) Evaluation based on Number of Devices




(c) Evaluation based on Task Density

Figure 4.9: Cloud Offloading Performance based on Number of Tasks, Number of Devices and Task Density

Table 4.4: Cloud SLA set

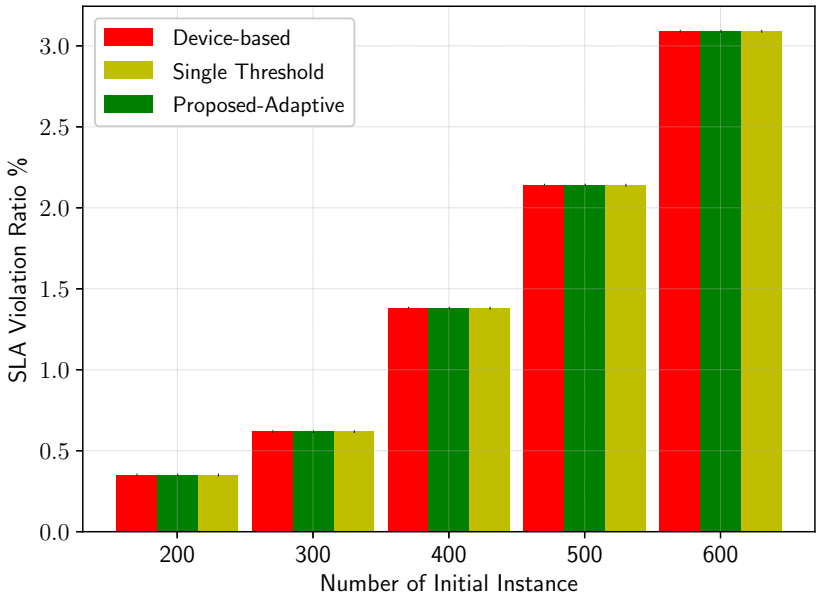
| **Parameter** | **Definition and Value** |
|---|---|
| Number of Host | h = 500 |
| Number of VM | v = 2000 |
| Initial Number of VM | n = 200, 300, 400, 500, 600 |
| Initial Number of Task | t = n/2 |
| Task Variation Rate | r = 20%, 40%, 60%, 80% |
| Device-based | Always on |
| Single Threshold | Static Threshold |

and Device-based. The Single Threshold uses static thresholds without adaptivity during the execution. The device-based solution utilizes "always-on" solutions so that the SLA violation only happens when the number of incoming tasks exceeds the capacity of that Cloud. We establish the experiment based on different initial instance ranging from 200 to 600. The task variation rate – representing the system load changing ratio – is set from 20% to 80%, according to the work [246], shown in Table 4.4.

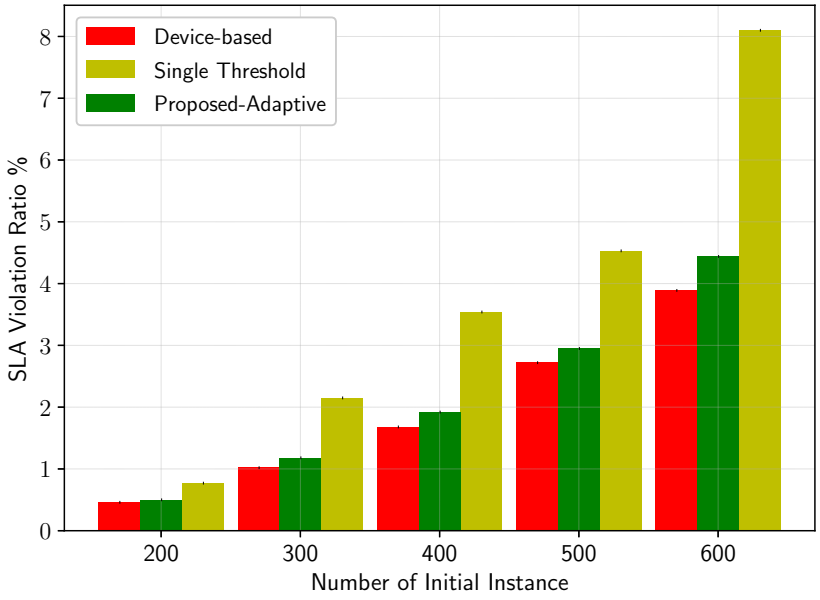
The results are presented in Figure 4.10. In the low task variation scenarios Figure 4.10a, SLA violation only occurs when the task size just passes the total Cloud capacity. In the medium and high task variation scenarios, the proposed adaptive scheme can achieve 45%, 84% and 81.8% fewer SLA violations, compared to the static single threshold counterpart in the 40%, 60%, and 80% task variation ratio respectively. Compared to the device-based "always on" solutions, the proposed task-centric scheme only introduces extra 4% SLA violations due to inappropriate scaling, in the worst case scenario shown in Figure 4.10d.

### 4.3.3 Discussion

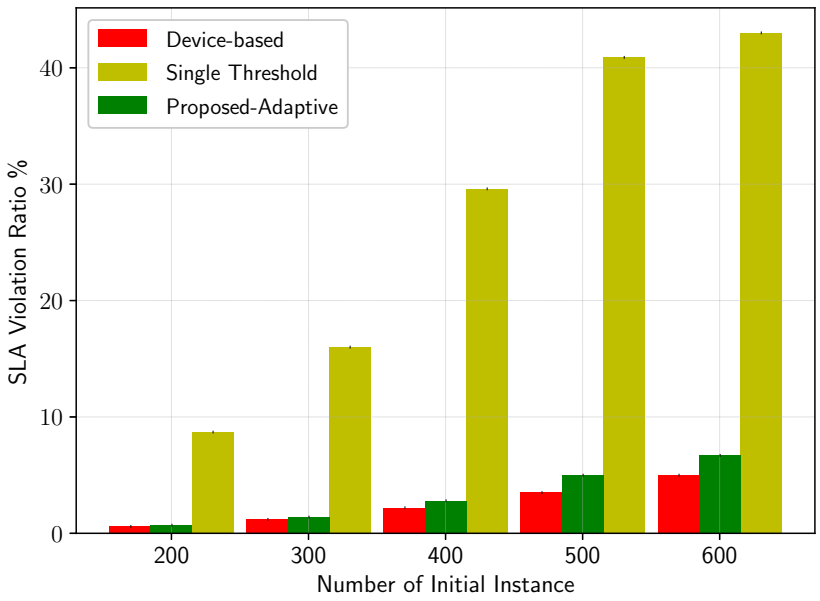
In this section, The system energy model is first tested and validated, and the performance of the proposed system is then evaluated on the mobile terminals, intermediate Cloudlets and remote Cloud resources respectively. By adopting the proposed fine-grain task energy prediction model, the mobile element level offloading energy efficiency and execution efficiency is compared with mobile-only and Cloudlet-only execution scenarios with an overall approximately 60% improvements in the high load cases. By taking advantage of the global level task execution knowledge, the proposed task-centric Cloudlet can achieve around 50% higher processing throughput compared to the device-based execution solutions. By manipulating the resource provisioning on the task level, the



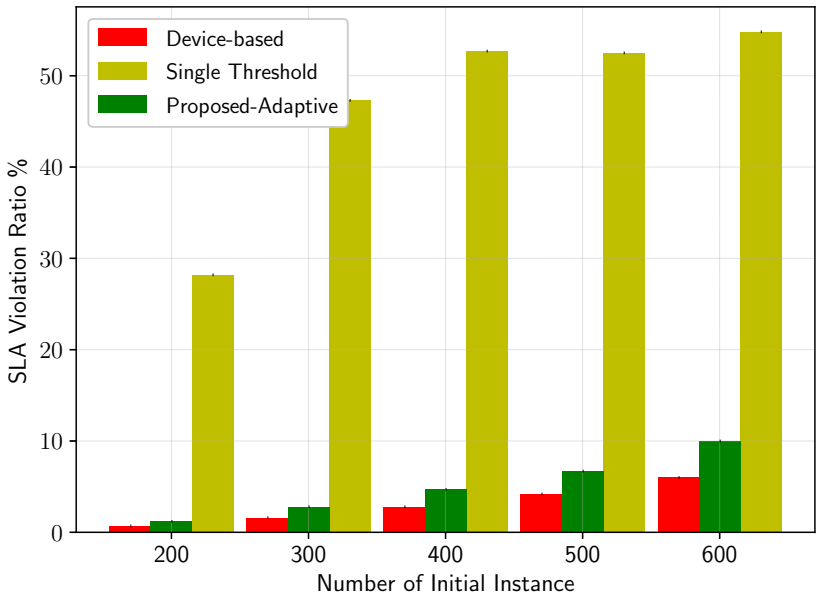

(a) 20 % Task Variation




(b) 40 % Task Variation




(c) 60 % Task Variation




(d) 80 % Task Variation

Figure 4.10: SLA Violation based on Task Variation Rate and Initial Number of Resource

proposed remote Cloud remote scheduling scheme can respond to the load changes more flexible than the device-based service models, which in return enables more than 30% energy reservation when comparing to the traditional device-based ones.

## 4.4 Conclusion

In this chapter, a Cloudlet-based task-centric model is proposed. The system is first presented with analysis and discussion of the local mobile devices, Cloudlets, remote Cloud and networking, regarding the roles and core functions. In the proposed system which incorporates [145], the offloading scenario is formulated with the task module, computing module, and network module respectively and a collaborative offloading scheduling algorithm is proposed. Compared to the device-based solutions, the proposed task-centric Cloudlet-based system can achieve fine-grain dynamic resource control in the Cloudlet and remote Cloud. In addition, the leverage of global execution knowledge also benefits the offloading decision-making. Based on the experiments, the proposed system formulation is proved to be valid in the real data; the scheduling protocol is evaluated in terms of execution efficiency, energy efficiency on the task level and the Cloudlet level.

The energy efficiency, execution efficiency, scalability and availability of the system is tested and observed in the experiments. As shown in the experiment results, it can be concluded that the proposed system in this chapter outperforms the device-based ones in terms of the task processing efficiency and system-level energy efficiency in the large-scale static and mobility scenarios. The user security is also maintained by adopting the virtualization-based hardware isolation on the Cloudlet and remote Cloud.

# Chapter 5

# Proactive Offloading and Sustainable Resource Allocation in Heterogeneous Cloudlets

In this chapter, a proactive Cloudlet-based hybrid offloading model is proposed to handle the energy and QoS-aware heterogeneous offloading and resource allocation issue. Compared to the conventional Cloudlet-based solutions, the proposed offloading model considers the heterogeneous offloading scenarios with different types of tasks, resources and mixed offloading methods. It can coordinate with Cloud resources and perform partition-based offloading and migration-based offloading dynamically and simultaneously according to the task load and QoS. In addition, the system is integrated with fine-grain load prediction components to achieve proactive resource allocation, concerning the trend and seasonality of the task load. The chapter focuses on the Cloudlet-based mixed offloading issue, which targets the mixed offloading coordination and fine-grain load prediction.

The organization of the remaining sections is as follows. In Section 1, the system architecture and component are briefly introduced, an energy-aware deadline-constraint heterogeneous scheduling and resource allocation problem is formulated, a PSO-based optimization solution is introduced, and a time series prediction model is demonstrated. In Section 2, the proposed system is tested and compared to traditional Cloudlet-based frameworks. In section 3, the chapter is summarized, a conclusion is drawn, and future research directions are further discussed.

## 5.1 System Formulation

In this section, the proposed Cloudlet-based hybrid offloading system is first briefly introduced with core component and function introductions. Then, the formulating the offloading decision-making and resource allocation process is presented based on the queueing model. According to the queueing model, the time-constraint energy and execution efficient heterogeneous offloading problem is proposed. In order to solve the problem promptly, a PSO-based heterogeneous offloading algorithm is designed to obtain the near optimal solution. Concerning the fluctuation of system loads, a SARIMA-based load prediction model is integrated to balance offloading resources proactively.

### 5.1.1 System Description

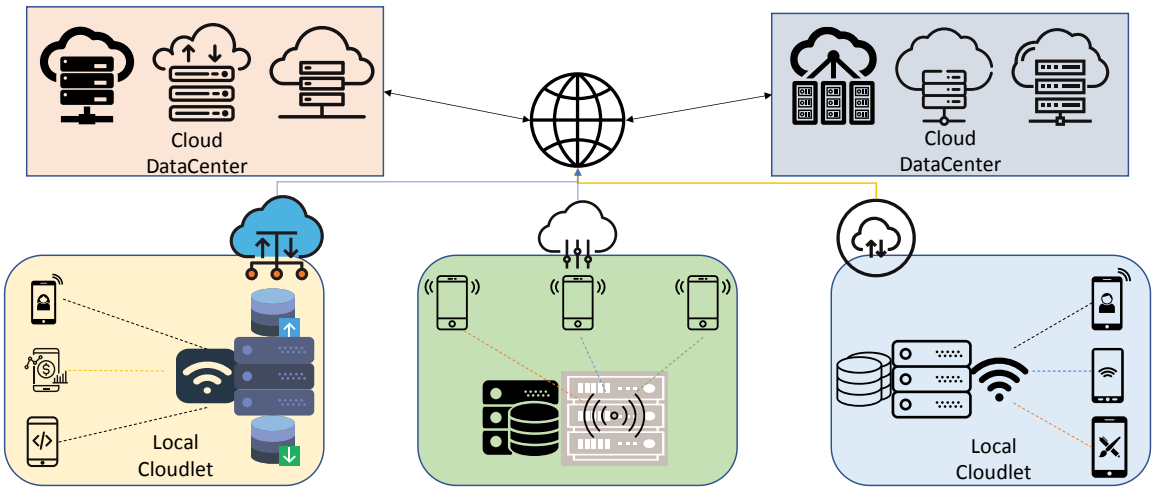



Figure 5.1: Cloudlet-based Hybrid Offloading System Architecture and Networking

This chapter considers the two-layered offloading architecture and networking, as shown in Figure 5.1. According to the vicinity, the offloading resources are divided into local Cloudlets (Rounded Rectangles) and remote Cloud datacenters (Rectangles), which are connected through wired Wide Area Network (WAN) as shown in the solid lines. The dash lines in the local region depict the wireless connections between mobile devices and the Cloudlet. In the offloading processing, different devices may trigger diverse offloading requests. These demands are sent to non-identical Cloudlets through varying network conditions, being executed locally or forwarded to heterogeneous remote Clouds.

Table 5.1: Parameter Definition

| **Parameter** | **Definition** |
|---|---|
| $\lambda$ | arriving rate of total offloading tasks |
| $\lambda_{cl}$ | arriving rate of Cloudlet-based tasks |
| $\lambda_c$ | arriving rate of remote Cloud-based tasks |
| $\lambda_{t_i}$ | arriving rate for task $i$-based request |
| $\lambda_d$ | arriving rate for migration-based request |
| $\lambda_{cl_b}$ | blocking rate of Cloudlet res allocation queue |
| $\lambda_{i_b}$ | blocking rate of task $i$-based proc queue |
| $\lambda_{d_b}$ | blocking rate of device-based proc queue |
| $\lambda_{drop}$ | dropping rate of offloading |
| $\mu_l$ | proc rate of Location Mapping Engine |
| $\mu_{t_i}$ | proc rate of task $i$-based res |
| $\mu_{cl}$ | proc rate of device-based res |
| $\mu_c$ | proc rate of Cloud res |
| $\mu_{cl}$ | proc rate of Resource Allocation Engine |
| $\mu_d$ | proc rate of migration-based res |
| $P_{cl}$ | probability of sche a task to the Cloudlet |
| $P_c$ | probability of sche a task to the remote Cloud |
| $P_{t_i}$ | probability of sche a task to task $i$-based res |
| $P_d$ | probability of sche a task to device-based res |
| $c_i$ | num of current available res for task type $i$ |
| $d$ | num of current available device-based res |
| $C$ | num of total res units in the Cloudlet |
| $N_t$ | num of total offloading task types |
| $d_t$ | num of total dropped task types |
| $N_{net}$ | num of total communication units on Cloudlet |
| $\mathrm{E}[t_{i\_q}]$ | avg number of tasks in task $i$-based que |
| $\mathrm{E}[t_{i\_wait}]$ | avg waiting time of tasks in task-$i$-based que |
| $\mathrm{E}[t_{i\_proc}]$ | avg processing time in task $i$-based res |
| $\mathrm{E}[t_{i\_sys}]$ | total time spent on task $i$-based res |
| $\mathrm{E}[_{d\_sys}]$ | total time spent on device-based res |
| $t_{id}$ | task processing deadline for type $i$ |
| $T_i$ | total offloading time for task $i$ |
| $T_{i\_comm}$ | communication time for task $i$ |
| $T_{i\_proc}$ | proc time for task $i$ |
| $S_i$ | package size of task $i$ |
| $S_b$ | package size of base image |
| $R_L$ | data rate of WLAN |
| $R_W$ | data rate of WAN |
| $D_L$ | delay of WLAN |
| $D_W$ | delay of WAN |
| $E_{md\_i}$ | avg energy cost on device $md$ for task $i$ |
| $E_{md\_w}$ | avg transmission energy cost on device $md$ |
| $E_{md\_idle}$ | avg idle status energy cost on device $md$ |

### 5.1.2 System Components and Functioning

In our proposed system, the core functioning units in the Cloudlet include Profiler, Res Monitor, Decision Engine, Task Allocator and Load Balancer, as shown in Figure 5.2. The Profiler and Res Monitor collect and save task-related offloading data and resource-based offloading environment information, respectively. Based on the above logs, the Decision Engine adopts the Proc, Energy and Prediction module to schedule offloading tasks and to balance the load. The Task Allocator and Load Balancer trigger the offloading execution process and update the task and resource records.

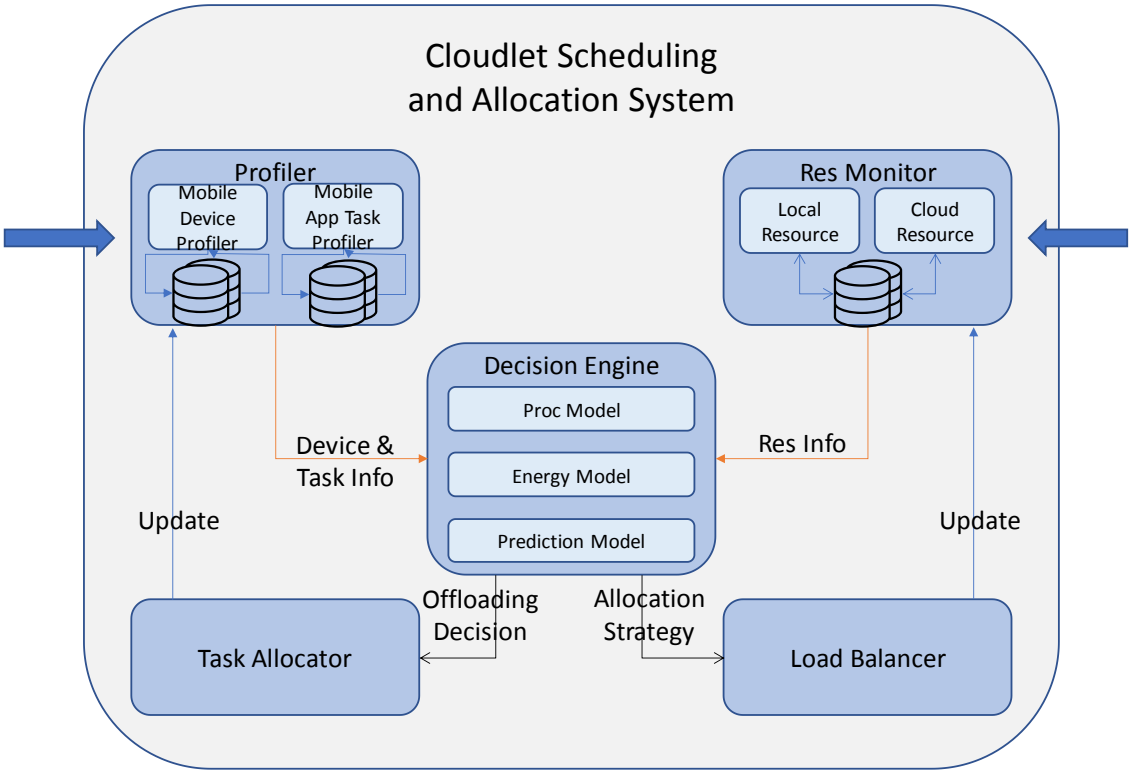



Figure 5.2: Cloudlet-based Functioning Components

### 5.1.3 Queue-based Offloading Processing Model

The proposed Cloudlet-based offloading processing model is shown in Figure 5.3. Once the mobile devices enter the coverage of the local Cloudlet, they can register themselves with the Cloudlet agent and begin sending offloading requests. The total offloading arrival rate for all offloading tasks from all mobile devices is denoted by $\lambda$.

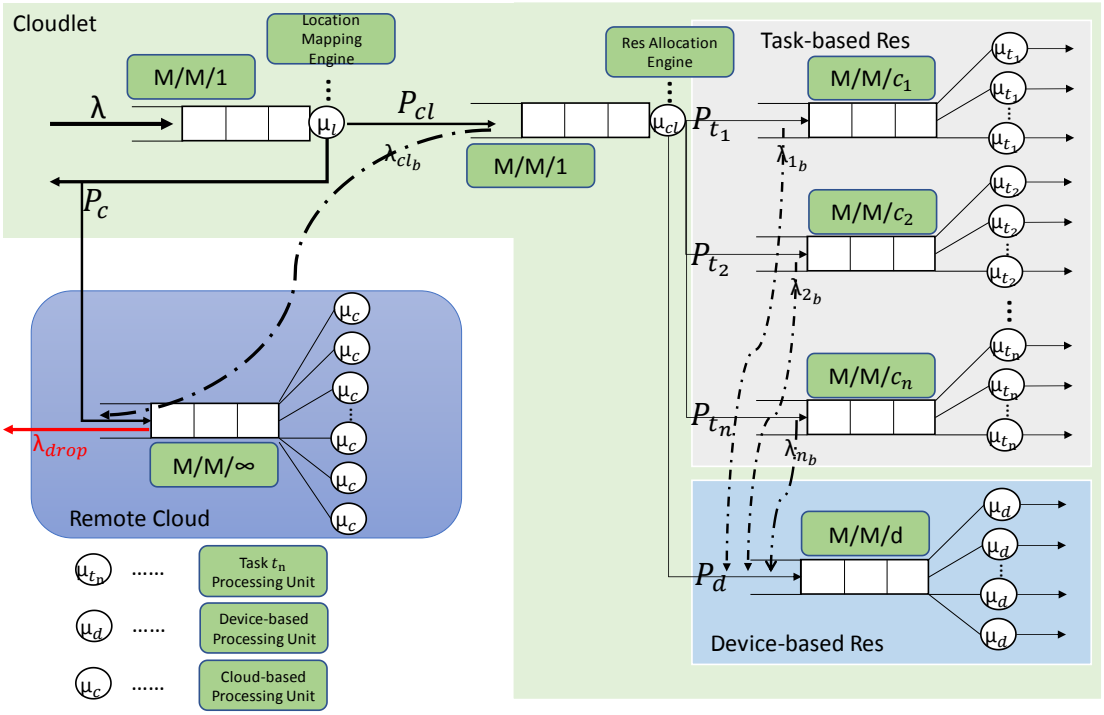



Figure 5.3: Queue-based Offloading Task Processing Model

When the Cloudlet receives new offloading requests, the Location Mapping Engine with service rate $\mu_l$ filters these requests according to the user's requirements. The private security-sensitive offloading tasks are forwarded to the Cloud corresponding individual computing units with possibility $P_c$ while the general computing components are sent to the Cloudlet Resource Allocation Engine with possibility $P_{cl}$ for further Cloudlet-based resource scheduling. In this work, the model Public cloud is modeled as a $M/M/\infty/N_{net}$ queue. It is assumed that the buffer and waiting time of the queue is negligible, because the remote Cloud is assumed to maintain theoretically 'unlimited' computing capability. However, the capacity of the queue is constrained by the limited communication units on the Cloudlet $N_{net}$.

For the requests sent to the Cloudlet Resource Allocation Engine, the engine further dispatches the tasks to the local Cloudlet computing utilities. The Cloudlet resources are divided into task-based resources and device-based resources, which are prepared for task-level partition-based offloading and VM-level migration-based offloading, respectively. According to the work [259], this part of the work summarizes the characteristics of both offloading methods in the Table 5.2. The partition-task method requires the offloading execution environment for that type of task to be fully prepared on the Cloudlet

Table 5.2: Partition-based Offloading and VM Migration-based Offloading

| Method | Partition-task | VM-Migration |
|---|---|---|
| Prep Delay | High | Low |
| Comm Overhead | Low | High |
| Energy Cost | Low | High |
| Usability | Limited | Good |
| Compatibility | N/A | Yes |

resource clone before the offloading begins. In this case, only task-related soft-layer information is synchronized during offloading run-time, which leads to low communication overhead and less data transmission energy cost. The VM-migration method can achieve better usability because it does not expect pre-configuration and installation of specific task-related dependent packages. However, it needs to migrate the execution environment from the mobile devices to the Cloudlet during run-time, which leads to higher communication overhead and data transmission power consumption on the mobile elements. It is worth mentioning that the migration-based solution is compatible with the task-partitioning offloading method if all soft-layer information and hard-layer environments for that partition-based offloading are synchronized during run-time.

Finally, the offloading requests dispatched by the Resource Allocation Engine are processed either on the task-based resources for partition-task offloading or on the device-based resources for VM-migration. Regarding the partition-based tasks, we define the vector set $< P_{t_1}, P_{t_2}, P_{t_3}, ..., P_{t_n} >$ to indicate the probabilities that the Resource Allocation Engine send the task with type $t_i$ to its corresponding resource set. The processing of task $t_i$ is modeled as $M/M/c_i$ queue where $c_i$ is the number of computing units to serve the offloading tasks with type $t_i$. The dispatching possibility for migration-based tasks is set as $P_d$ and the processing is based on the $M/M/d$ queueing model where $d$ is the number of servers. We assume that the task-based offloading requests and the general device-based offloading requests arrive according the Poisson process, and the service times are exponentially distributed.

Concentrating on the Cloudlet task-based processing queue for the task type $t_n$, the transient rate diagram for the incoming tasks can be modeled based on the birth-and-death process under the constraint of formula 5.1 and 5.2, according to [29]. The constraints are employed to ensure the system is able to achieve the stable states.

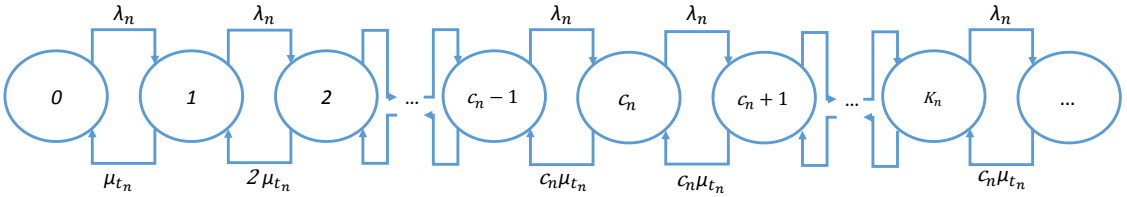



Figure 5.4: Transient Rate Diagram for Processing Task Type $t_i$

$$\lambda_{t_n} < \mu_{tn} * c_n \tag{5.1}$$

$$\lambda_{t_n} = \mu_{cl} * P_{t_n} \tag{5.2}$$

The diagram is presented in Figure 5.4. The stable probability of the number of incoming tasks with type $t_n$ in the $M/M/c_n$ queue system can be calculated as follows, where $\rho_0$ is the possibility of zero tasks in the system:

$$\rho_j = \begin{cases} \frac{\lambda_{t_n}^j}{\mu_{t_n}^j * j!}\rho_0 & \forall j < c_n, \\ \frac{\lambda_{t_n}^j}{\mu_{t_n}^j * c_n^{j-c} * c_n!}\rho_0 & \forall j >= c_n \end{cases} \tag{5.3}$$

According to the probability normalization condition in 5.4, the value of $\rho_0$ can be calculated as in 5.5 and 5.6.

$$\sum_{j=0}^{\infty} \rho_j = 1 \tag{5.4}$$

$$\rho_0 = (\sum_{j=0}^{c_n-1} \frac{\lambda_{t_n}^j}{\mu_{t_n}^j j!} + \rho_{wait})^{-1} \tag{5.5}$$

$$\rho_{wait} = \sum_{j=c_n}^{\infty} \rho_j = \frac{\lambda_{t_n}^{c_n}}{\mu_{t_n}^{c_n} c_n!} \sum_{j=c_n}^{\infty} \frac{\lambda_{t_n}^{j-c_n}}{c_n^{j-c_n} \mu_{t_n}^{j-c_n}} \tag{5.6}$$

As constrained by formula 5.1, the $\rho_j$ series converges and the sum of the series can be estimated as in 5.7, according to the Erlang C formula.

$$\rho_{wait} \approx \frac{\lambda_{t_n}^{c_n}}{\mu_{t_n}^{c_n} c_n!} \frac{c_n \mu_{t_n}}{c_n \mu_{t_n} - \lambda_{t_n}} \tag{5.7}$$

Then, the value of $\rho_0$ can be computed in 5.8:

$$\rho_0 \approx (\sum_{j=0}^{c_n-1} \frac{\lambda_{t_n}^j}{\mu_{t_n}^j j!} + \frac{\lambda_{t_n}^{j-c_n}}{\mu_{t_n}^{j-c_n} c_n!} \frac{c_n \mu_{t_n}}{c_n \mu_{t_n} - \lambda_{t_n}})^{-1} \tag{5.8}$$

By adopting the value of $\rho_0$ in 5.3, the probability for each number of states can be obtained. Based on the results in 5.3, the expected number of offloading tasks in the task type $n$-based queue can be calculated as the average value in 5.9.

$$\begin{aligned} \mathrm{E}[t_{n\_q}] &= \sum_{j=c_n}^{\infty} (j - c_n)\rho_j \\ &= \frac{\lambda_{t_n}}{c_n \mu_{t_n} - \lambda_{t_n}} * \frac{\lambda_{t_n}^{c_n}}{\mu_{t_n}^{c_n} c_n!} * \frac{c_n \mu_{t_n}}{c_n \mu_{t_n} - \lambda_{t_n}} \rho_0 \end{aligned} \tag{5.9}$$

Then, the expected waiting time in the queue can be computed from Little's formula:

$$\begin{aligned} \mathrm{E}[t_{n\_wait}] &= \frac{\mathrm{E}[t_{n\_q}]}{\lambda_{t_n}} \\ &= \frac{1}{c_n \mu_{t_n} - \lambda_{t_n}} * \frac{\lambda_{t_n}^{c_n}}{\mu_{t_n}^{c_n} c_n!} * \frac{c_n \mu_{t_n}}{c_n \mu_{t_n} - \lambda_{t_n}} \rho_0 \end{aligned} \tag{5.10}$$

As shown in formula 5.11, the total processing time during offloading includes the queueing time and the service time, the expected total time for each offloading task with type $n$ in the system can be calculated as the sum:

$$\begin{aligned} \mathrm{E}[t_{n\_sys}] &= \mathrm{E}[t_{n\_wait}] + \mathrm{E}[t_{n\_proc}] \\ &= \frac{1}{c_n \mu_{t_n} - \lambda_{t_n}} * \frac{\lambda_{t_n}^{c_n}}{\mu_{t_n}^{c_n} c_n!} * \frac{c_n \mu_{t_n}}{c_n \mu_{t_n} - \lambda_{t_n}} \rho_0 + \frac{1}{\mu_{t_n}} \end{aligned} \tag{5.11}$$

We define the negotiated task offloading processing deadline on the Cloudlet resources for the offloading tasks as a vector $< t_{1d}, t_{2d}, t_{3d}, ..., t_{nd} >$. Then the maximum average waiting time in queue before violating the task processing deadline with type $n$ can be calculated as:

$$t_{n\_max} = t_{nd} - \mathrm{E}[t_{n\_proc}] = t_{nd} - \frac{1}{\mu_{t_n}} \tag{5.12}$$

According to the $t_{n\_max}$ value in the above formula, the potential blocking rate of the task $n$-based offloading requests that are used to ensure QoS-based offloading processing time constraints can be calculated as in 5.13 and 5.14:

$$\lambda_{t_n\_max} := \arg\max_{\lambda'} \mathrm{E}[t_{n\_wait}](\lambda') <= t_{n\_max} \tag{5.13}$$

$$\lambda_{n_b} = \begin{cases} 0 & \forall \lambda_{t_n} =< \lambda_{t_n\_max}, \\ \lambda_{t_n} - \lambda_{t_n\_max} & \forall \lambda_{t_n} > \lambda_{t_n\_max} \end{cases} \tag{5.14}$$

To obtain the value of $\lambda'$ in formula 5.13, we adopt a binary search-based algorithm to approach the maximum arrival rate in the system, under the offloading processing time constraint. As in Algorithm 7, if the offloading request arrival rate is small enough, the service migration rate is zero, shown in lines 3 and 4. Otherwise, a binary search process is employed to find the service blocking rate from line 5 to line 20. The inputs for the service request arrival rate are predicted from our estimation model 5.1.6. In addition, this calculating process is finished before the offloading service process starts, so that the computing will not increase the offloading run-time overhead.

**Algorithm 7:** Time Constraint-based Offloading Task Adaptive Migration

```
1  Require: λ_{t_n}, c_n, μ_{t_n}
2  round : max searching loops
3  if E[t_{n_wait}](λ_{t_n}) <= t_{n_max} then
4      Return : λ_{nb} ← 0
5  else
6      max ← ⌊λ_{t_n}⌋
7      min ← c_n
8      val ← t_{n_max}
9      while (min <= max do
10         mid ← min + (max − min)/2)
11         if E[t_{n_wait}](mid) > t_{n_max} then
12             max ← mid − 1
13         else
14             if E[t_{n_wait}](mid) < t_{n_max} then
15                 min ← mid + 1
16         if start == end then
17             Return : λ_{nb} ← mid
18         if round == 0 then
19             Return : λ_{nb} ← min
20         round ← round − 1
21 Output: λ_{nb}
```

Based on the value of $\lambda_{n_b}$, the offloading request arrival rate for task $n$-based offloading can be updated as follows. The extra task requests are transferred to migration-based offloading resources, as shown in Figure 5.3.

$$\lambda_{t_n} = \begin{cases} \lambda_{t_n} & \forall \lambda_{t_n} < \lambda_{t_n\_max}, \\ \lambda_{t_n\_max} & \forall \lambda_{t_n} >= \lambda_{t_n\_max} \end{cases} \tag{5.15}$$

The migration-based offloading on the Cloudlet device-based resources performs similarly. The processing is also based on the $M/M/d$ queue, where $d$ is the number of resources available to execute the offloading request. The arrival rate for the migration-based offloading is calculated in the following formula:

$$\lambda_d = \mu_{cl} P_d + \sum_{i=1}^{n} \lambda_{i_b} \tag{5.16}$$

Given that the offloading task dispatching process in the Resource Allocation Engine and the Location Mapping Engine only require linear computing time to find the offloading destination for each request, we assume that the offloading request processing time on the Cloudlet does not include the offloading scheduling cost. In this case, the migration rate for the Cloudlet resource queue $\lambda_{cl_b}$ can be obtained by calculating the service blocking rate of the device-based resource queue as:

$$\lambda_{cl_b} = \lambda_{d_b} \tag{5.17}$$

In this case, the offloading task arriving rate for Cloudlet and remote Cloud can be updated as in the following formulas 5.18 and 5.19.

$$\lambda_{cl} = \mu_l P_{cl} - \lambda_{cl_b} \tag{5.18}$$

$$\lambda_c = \mu_l P_c + \lambda_{cl_b} \tag{5.19}$$

Finally, the offloading dropping rate on the Cloud processing queue can be obtained by 5.20, where $N_{net}$ is the maximum communication units on the Cloudlet.

$$\lambda_{drop} = \lambda_c - N_{net} \tag{5.20}$$

### 5.1.4 Offloading Cost Model

Based on the queue-based offloading processing model, the overall offloading time can be obtained by summing up the offloading service processing time and service transmission time for each offloading request on the resources. According to the type of execution resources, the single offloading request mean processing time cost $T_i$ is calculated by calculating the task offloading communication time $T_{i\_comm}$ and offloading processing time $T_{i\_proc}$ together:

$$T_i = T_{i\_comm} + T_{i\_proc} \tag{5.21}$$

$$T_{i\_comm} = \begin{cases} S_i/R_L + D_L & \forall i \in task-based, \\ (S_i + S_b)/R_L + D_L & \forall i \in device-based, \\ S_i/R_W + D_W & \forall i \in Cloud-based \end{cases} \tag{5.22}$$

$$T_{i\_proc} = \begin{cases} \mathrm{E}[t_{i\_sys}] & \forall i \in task-based, \\ \mathrm{E}[t_{d\_sys}] & \forall i \in device-based, \\ 0 & \forall i \in Cloud-based \end{cases} \tag{5.23}$$

where $S_i$ is the package size of used for type $i$-based offloading overlay and $S_b$ is the package size of the base image. $R_L$ and $R_W$ are the date rates for LAN and WAN links, while $D_L$ and $D_W$ are the link media delay for LAN and WAN, respectively.

Consequently, the mean energy cost $E_{md\_i}$ on the mobile device $md$ for the offloading task $i$ can be computed:

$$E_{md\_i} = \begin{cases} E_{md\_w} * T_{i\_comm} + E_{md\_idle}\mathrm{E}[t_{i\_sys}] & \forall i \in task-based, \\ E_{md\_w} * T_{i\_comm} + E_{md\_idle}\mathrm{E}[t_{d\_sys}] & \forall i \in device-based, \\ E_{md\_w} * T_{i\_comm} & \forall i \in Cloud-based \end{cases} \tag{5.24}$$

where $E_{md\_w}$ is the mean transmission energy cost on mobile device $md$ and $E_{md\_idle}$ is the mean idle energy cost.

Given the average task processing time on the mobile device as $t_{md}$ and the processing energy as $E_{md\_proc}$, the tradeoff between energy consumption and delay highly depends on the communication overhead and the execution capability between the mobile device and the Cloudlet, based on the works [183], [244] and [158].

$$Decision = \begin{cases} non-offloading & t_{md} < T_i,\ E_{md\_proc} < E_{md\_w}, \\ offloading & t_{md} > T_i,\ E_{md\_proc} > E_{md\_w}, \\ tradeoff & others \end{cases} \tag{5.25}$$

In this chapter, we focus on the condition that tasks are sent to the Cloudlets by the mobile devices. Considering the offloading execution and energy efficiency, we define the time-energy joint cost model as follows:

$$C(md, i) = \alpha * T_i * \Delta t + (1 - \alpha) * E_{md\_i} * \Delta E \tag{5.26}$$

where $\alpha$ is the weight coefficient to control the importance between execution efficiency and energy efficiency. $\Delta t$ and $\Delta E$ are the coefficients that unify the order of magnitude for these two parameters.

In this case, we define the Cloudlet-based heterogeneous offloading resource allocation solution $S$ as an $n+1$ dimensional vector $< c_1, c_2, c_3, ...c_n, d >$, where $c_i$ is the number of resources allocated for task type-$i$ and $d$ is the number of resources for device-based

offloading tasks on the Cloudlet. Then, the Cloudlet-based heterogeneous offloading resource allocation problem can be defined to find the solution $S$ which can minimize the joint cost for all tasks from all devices under the coverage:

$$F(s) = \arg\min_{s \in S} \sum_{md=1}^{N_d} \sum_{i=1}^{N_t} C(md, i) \tag{5.27}$$

$$s.t. \quad \forall i \in N_t, \quad \mathrm{E}[t_{i\_sys}] < t_{id} \tag{5.28}$$

$$s.t. \quad \forall i \in N_t, \quad \mathrm{E}[_{d\_sys}] < t_{id} \tag{5.29}$$

$$s.t. \quad \forall c \in S, \quad \sum_{j=1}^{n} c_j + c_d <= C \tag{5.30}$$

where $N_d$ is the total number of mobile devices and $N_t$ is the total number of task types.

The constraints 5.28 and 5.29 indicate the offloading time on the Cloudlet task-based offloading and device-based offloading should be less than the deadline of that type of tasks. The constraint 5.30 implies the total number of allocated resources should not exceed the capability of the Cloudlet.

### 5.1.5 PSO-based Heterogeneous Offloading Resource Allocation

According to 5.27, the deadline-constraint resource allocation problem is a NP problem. A complete search in the solution space cannot be accomplished in polynomial time. In this case, we adopted the PSO heuristic, which has been proved to be fast convergence and effective in solving Cloud-based scheduling problems [185].

Although the constraints 5.28 and 5.29 can be ensured by adopting the proposed task adaptive migration, inappropriate resource allocation solutions may still lead to high offloading request dropping rate $\lambda_{drop}$ as in formula 5.20. These dropping tasks have to be processed on the model element itself, which leads to high energy costs during the processing. Considering the impact of offloading rejection, we introduce the penalty function to punish the offloading dropping rate. The penalty function is defined as follows:

$$P(s) = -\sum_{j=1}^{d_t} E_m(j) \tag{5.31}$$

where $P(s)$ is the function to calculate the energy consumption on all mobile devices due to the offloading request rejection. $d_t$ is the number of rejected request type and $E_m$ is the total energy cost for that type.

In this case, we can define the PSO fitness function as:

$$Fit(s) = F(s) - P(s) \tag{5.32}$$

Concerning the constraint 5.30 and the heterogeneousness of the computing resources, we define the position of the particle as a $x \times y$ matrix $Pos$, where $x$ is the number of virtual instances, and $y$ is the number of task types which equals ($N_t$). Given $u(i)$ as the computing units in the $i$th virtual instance, the total available computing units $C$ in the Cloudlet equals:

$$C = \sum_{i=1}^{x} u(i) \tag{5.33}$$

In this case, the overall allocated computing units will not exceed the size of the Cloudlet. We use binary values in each position of the matrix as: $Pos(x', y') \in \{0, 1\}$. Given $Pos_k$ as the $k$th particle's position matrix, $Pos_k(x', y') = 0$ indicates that the virtual instance $x'$ is allocated for the task type $y'$. In order to ensure the QoS, we do not allow load overcommitment, and restrict that each virtual instance can only serve one type of resource at the same time:

$$\sum_{y'=1}^{y} Pos_k(x', y') <= 1, \forall x' \in [1, x] \tag{5.34}$$

Similarly, the velocity of the particle is created as a $x \times y$ matrix $Vel$. Given the $k$th particle velocity $Vel_k$, its element value $Vel_k(x', y')$ is in the range of $[-V_{max}, V_{max}]$.

The PSO algorithm execution starts by randomly initiating the $Pos$ and $Vel$ for the particles. Updates of the position matrix and velocity matrix of the particle $k$ in the swarm occurs according to the PSO routine as follows:

$$\begin{aligned} Vel_k^{t+1}(x, y) =& wVel_k^t + c_1 r_1 (pbest_k^t(x, y) - Pos_k^t(x, y)) \\ &+ c_2 r_2 (nbest_k^t(x, y) - Pos_k^t(x, y)) \end{aligned} \tag{5.35}$$

$$Pos_k^{t+1}(x,y) = \begin{cases} 1 & if\ Vel_k^{t+1}(x,y) = max\{Vel_k^{t+1}(i,y)\}, \\ & \forall i \in \{1,2,...x\} \\ 0 & otherwise \end{cases} \tag{5.36}$$

where $w$ is the inertia weight; $c_1$ and $c_2$ are the learn factors; $r_1$ and $r_2$ are random factors. An example of the value updating process is illustrated as below in the 3 x 4 matrix which consists of 3 types of tasks and 4 resources. In this case, resource 1 and 2 are scheduled for task 3. Resource 3 is for task 1 and resource 4 is for task 2.

$$Vel_k = \begin{bmatrix} 1 & 2 & 5 & -1 \\ 0 & 1 & -1 & 0 \\ 3 & 3 & 2 & -3 \end{bmatrix} \rightarrow Pos_k = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

For a complete search in the solution space, given $C$ as the number of total resource units in the Cloudlet and $N_t$ as the number of offloading task types, the Brutal Force requires the computation complexity $O(C^{N_t})$. In terms of the proposed heuristic, given the number of particles as $N_p$, the number of iterations as $i$, the computational complexity is decided by the calculation of the speed matrix and the position matrix, which can be obtained as $O((C * N_t * N_p)^i)$. Compared to the complete search, the proposed heuristic can scale up properly as with the increasing of resources $C$ and offloading tasks $N_t$.

### 5.1.6 SARIMA-based Load Prediction and Proactive Heterogeneous Offloading Resource Allocation

As shown in the previous sections, the proposed Cloudlet-based heterogeneous offloading model can effectively allocate and balance the computing loads to minimize the offloading execution time cost and energy cost, according to the given $\lambda_{t_n}$ for the incoming tasks. Unfortunately, the offloading task load may vary significantly due to the heterogeneous location of Cloudlets and service time of day. A single $\lambda_{tn}$ cannot correctly and adequately reflect the seasonal fluctuation and potential trends of the task load.

In this case, instead of utilizing the general $\lambda_{t_n}$ as the load of the task type $n$, we define a series $\lambda_{t_n}^t$ as $< \lambda_{t_n}^1, \lambda_{t_n}^2, \lambda_{t_n}^3, ...\lambda_{t_n}^m... >$ to describe the task arriving rate input during each time interval $\Delta t$ and assume each input element $\lambda_{t_n}^m$ is governed by the Poisson process. By adopting the model $SARIMA(p,d,q,P,D,Q)_s$, the value of $\lambda_{t_n}^m$ can be calculated via formula 5.37:

$$\Phi_P(B^s)\phi_p(B)\nabla_s^D\nabla^d\lambda_{t_n} = \Theta_Q(B^s)\theta_q(B)w_t \tag{5.37}$$

where $B$ is the backward operator. $\Phi_P(B^s)$ and $\phi_p(B)$ are the seasonal autoregression with order $P$ and ordinary autoregression with order $p$. $\nabla_s^D$ and $\nabla^d$ are the differencing

functions for the seasonal and ordinary difference components. The seasonal and non-seasonal moving average operations are represented by $\Theta_Q(B^s)$ and $\theta_q$, where $Q$ and $q$ is the order of the terms. $w_t$ is the residual white noise and $s$ is the seasonality factor. In this case, the load prediction problem is translated into finding the best values for the order set $< p, d, q, P, D, Q >$.

The load prediction and the overall system-level resource allocation is presented in the following Algorithm 8. The algorithm contains two functions ($FuncPred$ and $FuncAllo$) which handle the prediction of task load and resource allocation respectively.

The prediction function is shown from line 1 to line 13. First, a first order seasonal difference operation is performed to reduce the influence of seasonality (line 2). Then, according to the result of the Ljung-Box test, several rounds of ordinary differencing will be executed to make the residuals trend-independent (line 3). Since the values of $d, D$ is set, the next step is to find the values of $p, P, q, Q$. The potential values of $p, P, q, Q$ can be obtained by observing the results of Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) (line 6 to line 7). Based on the principle of parsimony, the potential value candidates are further filtered. By traversing all possible permutations, the best model is selected with the best Akaike Information Criterion (AIC). If the model passed the residual test, it is selected for the further forecasting (line 10 to line 13).

The PSO-based allocation function starts in line 14. If the processing time equals the resource allocation interval, it will perform the resource allocation function as from line 16 to line 33. During the allocation process, it first estimates the task load in the next processing time interval by utilizing the prediction function above. Then, in the initiation state, $indiNum$ particles will be created with a randomly generated velocity matrix and a position matrix (line 21 to line 23). The maximum updating iterations are set as $g_{max}$ as in line 26. During the updating, the fitness values are calculated by the formula 5.32. By comparing the fitness value with the particle's past position and the swarm's best position, the velocity and position matrix for the next timestamp can be computed by formulas 5.35 and 5.36, respectively (line 28 to line 33). Finally, the allocation solution can be obtained by assigning the solution $S$ the swarm best value. During the processing stage (line 35 to line 38), the system executes the allocation solution and advances the time.

**Algorithm 8:** Proactive Heterogeneous Offloading Resource Allocation Algorithm

FuncPred($\lambda^t_{t_n}, s$):
$\lambda^t_{t_n} \leftarrow diff(\lambda^t_{t_n}, D = 1)$
**while** $Ljung.test(\lambda^t_{t_n}, log(\lambda^t_{t_n}.len) < \epsilon$ **do**
  $\lambda^t_{t_n} \leftarrow diff(\lambda^t_{t_n}, 1)$
  $d \leftarrow d + 1$
  $q, Q \leftarrow ACF(\lambda^t_{t_n})$
  $p, P \leftarrow PACF(\lambda^t_{t_n})$
  **for** $p, q, P, Q \in pa(p, d, q, P, D, Q)$ **do**
    $p', P', q', Q' \leftarrow AIC_{best}(< p, q, d, P, Q, D >)$
  **if** $Ljung.test(resd(\lambda^t_{t_n}, p', d, q', P', D, q')) > \epsilon$ **then**
    $model \leftarrow (\lambda^t_{t_n}, p', d, q', P', D, q')$
    *break*;
Return: $\lambda^{step}_{t_n} \leftarrow esti(model, step)$
FuncAllo($\mu, \lambda, t_d, C, md$):
**while** *true* **do**
  **if** $t == isReAllocation$ **then**
    $\forall \lambda_{t_n} \in \lambda, \quad \lambda^{t+1}_{t_n} \leftarrow FuncPred(\lambda^t_{t_n}, s)$
    $\lambda^{t+1}_d \leftarrow FuncPred(\lambda^t_d, s)$
    $\lambda^{t+1}_c \leftarrow FuncPred(\lambda^t_c, s)$
    Init Particles:
    **for** $\forall k \in Swarm[1, ...indiNum]$ **do**
      $Vel^1_k \leftarrow init(Vel)$
      $Pos^1_k \leftarrow init(Pos)$
    Update:
    $g \leftarrow 1$
    **while** $g < g_{max}$ **do**
      $g \leftarrow g + 1$ **for** $\forall k \in Swarm[1, ...indiNum]$ **do**
        **if** $\forall i \in [1, ...t], Fit_{max} == Fit(Pos^i_k)$ **then**
          $pbest^t_k \leftarrow Pos^i_k$
        **if** $\forall j \in [1, ...maxNum], Fit_{max} == Fit(pbest^t_j)$ **then**
          $nbest^t_k \leftarrow pbest^t_j$
        $Vel^{t+1}_k \leftarrow k.update(Vel^t_k, pbest^t_k, nbest^t_k, Pos^t_k)$
        $Pos^{t+1}_k \leftarrow k.update(Vel^{t+1}_k)$
    $S = nbest$
  **else**
    $sys.proc(S)$
    $\mu, \lambda, t_d, C, md \leftarrow sys.update()$
    $t \leftarrow t.next()$

Table 5.3: Experiment: System-level Hybrid Offloading Parameters

| **Parameter** | **Definition** | **Default Value** |
|---|---|---|
| $\mu_{cl}$ | proc rate of Cloudlet | 30/sec |
| $\mu_{md}$ | proc rate of mobile device | 5/sec |
| $cl_{pre}$ | preparation delay on Cloudlet | 0.06 sec |
| $c_{pre}$ | preparation delay on Cloud | 0.1 sec |
| $t_d$ | task deadline | 0.25 sec |
| $c_{block}$ | network unit of Cloudlet | 10,20,30,40,50 |
| $\lambda$ | task arrival rate | 20 to 120/sec |
| $C$ | num of res on Cloudlet | 1,2,3,4,5 |
| $\alpha$ | joint cost coefficient | 0, 0.5, 1 |
| $E_{md\_w}$ | avg trans energy cost | 0.1 W |
| $E_{md\_idle}$ | avg idle energy cost | 0.03 W |
| $E_{md\_proc}$ | avg proc energy cost | 0.84 W |
| Non-offloading | Task exec only on mobile devices | |
| Cloud-based | Cloud-based offloading | |
| Cloudlet-based | Cloudlet-based offloading | |
| Proposed | Hybrid offloading | |

Table 5.4: Experiment: Heterogeneous Offloading Resource Allocation Parameters

| **Parameter** | **Definition** | **Default Value** |
|---|---|---|
| $n_t$ | types of offloading task | 3 |
| $n_{cl}$ | num of Cloudlet res | 6 |
| $w$ | inertia weight | 0.9 |
| $r_1, r_2$ | random factor | 0 to 1 |
| $c_1, c_2$ | learn factor | 1.5 |
| $\alpha$ | joint cost coefficient | 1 |
| $V_{max}$ | maximum speed | 50 |
| $iter$ | rounds of iterations | 30 |
| $\mu_{t_1}, \mu_{t_2}, \mu_d$ | proc rate | 50, 20, 5/sec |
| $t_{1d}, t_{2d}, t_{dd}$ | task deadline | 0.2, 0.2, 0.3 sec |
| $cl[i]$ | num of cores in res $i$ | 1 to 6 |
| $\lambda_{t_2}$ | arrival rate for task 2 | 40/sec |
| $\lambda_{t_1}$ | arrival rate for task 1 | 30 to 150/sec |
| $\lambda_d$ | arrival rate for task $d$ | 10 to 50/sec |
| Device-based | Migration-based offloading | |
| Task-based | Partition-based offloading | |
| Load-based | Priority-based allocation | |
| Proposed | Heterogeneous offloading | |

Table 5.5: Experiment: Prediction-based Offloading Parameters

| **Parameter** | **Definition** | **Default Value** |
|---|---|---|
| $n_t$ | types of offloading task | 2 |
| $n_{cl}$ | num of computing res | 10 |
| $s$ | seasonal lag | 24 |
| $p, d, q, P, D, Q$ | prediction parameters | 1,2 |
| $t_{fr}$ | web req for Wiki fr main page | based on Fig.5.10 |
| $t_{ja}$ | web req for Wiki ja main page | based on Fig.5.10 |
| $\mu_{t_{fr}}, \mu_{t_{ja}}$ | proc rate | 10, 5/sec |
| $t_d$ | task deadline | 0.3/sec |
| $\mu_{md}$ | proc rate on mobile device | 1/sec |
| $cl[i]$ | num of cores in res $i$ | 1 |
| Prob-based | Mean-based prediction | |
| Real-proc | Pre-known the load | |
| Naive-proc | Non-prediction | |
| Proposed | TS-based prediction | |

## 5.2 Performance Evaluation and Discussion

In this section, the proposed energy-aware proactive heterogeneous MCC offloading model is evaluated based on three experiment sets. In the first experiment set, we compared our model with three traditional methods, motivating the work – intra-Cloudlet offloading scheduling can drastically improve offloading efficiency. In the second experiment set, we compared our algorithm to three recent studies and two baselines, as in Fig. 8 and Fig. 9. To the best of our knowledge, this is the first work that distinguishes the task-based offloading load and device-based offloading load and tries to balance the overall inter-Cloudlet load based on two different offloading processes. In the third experiment set, we compared the proposed prediction-based model to the recent queueing-based work with two other baselines.

The system-level environment setting is presented in Table 5.3. The Cloudlet-level heterogeneous allocation parameters are depicted in Table 5.4, while the prediction-related parameters are shown in Table 5.5. The energy consumption on the mobile devices can be profiled via the original Android Energy Profiler and third party tools such as PowerTutor [142]. In this chapter, we adopt the profiling values from [1], where we use level 4 as the mobile CPU processing level.

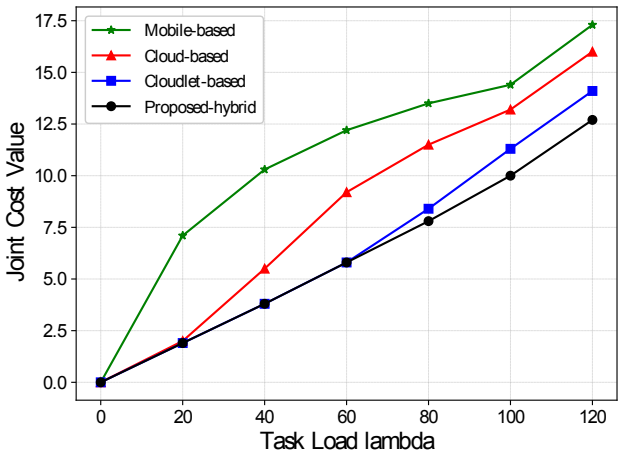
[1] Source from: https://source.android.com/devices/tech/power/ values.html (24/04/2019).

### 5.2.1 Hybrid Offloading Performance Evaluation

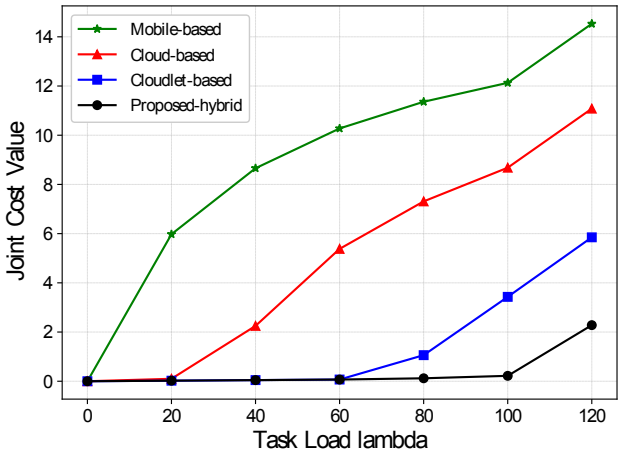
The first experiment is implemented to observe and evaluate the offloading efficiency on the system level. We compare the proposed hybrid offloading model to several existing solutions. The baseline non-offloading algorithm only processes tasks locally on mobile devices, without offloading. The Cloudlet-based and Cloud-based solutions enable offloading on resource-limited local resources and resource-unlimited remote resource, respectively. The proposed hybrid offloading model allocates resources between the Cloudlet and the Cloud, concerning resource constraints and communication overhead.

Experiment 1.1 (results in Figure 5.5) observes the impact of the joint cost coefficient, based on different offloading task loads. Experiment 1.2 (results in Figure 5.6) is implemented to analyze the offloading performance based on different computing capabilities on the Cloudlet under different task load arrival rates. Similarly, experiment 1.3 (results in Figure 5.7) concerns the offloading performance influenced by different communication capabilities.

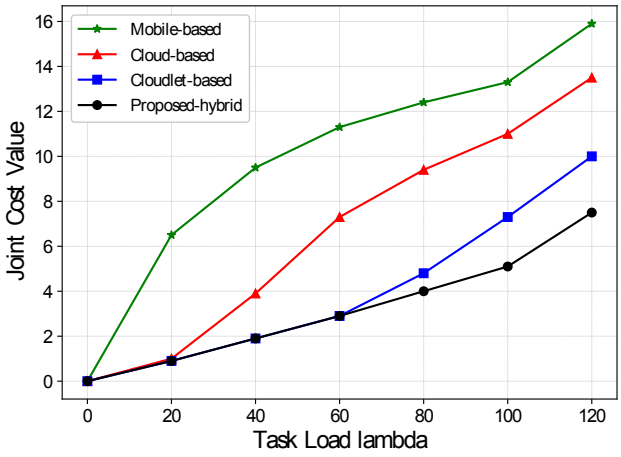
The results of experiment 1.1 are shown in Figure 5.5, which utilizes a default number of computing resources ($C = 4$) and communication resources ($c_{block} = 30$). By manipulating the value of the joint cost coefficient $\alpha$, the offloading execution time cost, energy cost and combined costs are depicted in 5.5a, 5.5b and 5.5c, respectively. As shown in 5.5a, the non-offloading baseline requires the most executing time due to limited process power on mobile devices. The Cloud-based solution can outperform the non-offloading in the low load scenario when the communication units are not exhausted. The Cloudlet-based approach can achieve a better execution time than the Cloud-based solution by tapping into the nearby Cloudlet computing resources. However, considering the task deadline, the Cloudlet-based model cannot accept all the offloading tasks with the growth of the task load, due to its limited computing capability. As a result, an obvious increasing trend of the Cloudlet-based cost can be found at task load value of 80 to 100, which indicates that the denial of offloading requests occurs. The best performance is achieved by the proposed model, which can fully utilize the Cloudlet computing and communication resources. Compared to the others, the proposed model can reduce executing times by 30% (non-offloading), 24% (Cloud-based), 11.5% (Cloudlet-based), respectively. A similar performance can be found in the experiments regarding offloading energy efficiency (Figure 5.5b) and mixed efficiency (Figure 5.5c). These results are in accordance with Figure 5.5a because the energy reservation on the mobile devices is achieved by reducing the number of task executions on these mobile elements. When the offloading




(a) Execution Time Cost (alpha = 1)




(b) Execution Energy Cost (alpha = 0)




(c) Offloading Joint Cost (alpha = 0.5)

Figure 5.5: Offloading Performance Evaluation based on different Task Load and joint cost coefficient ($C = 4$, $c_{block} = 30$ )

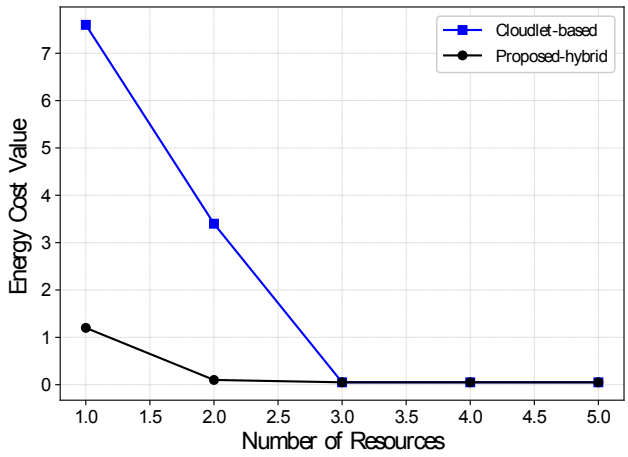
communication overhead is smaller than the offloading computing benefit, a smaller value of the execution time suggests that a larger number of tasks are migrated to high-end offloadees.

Figure 5.6 presents the results of experiment 1.2 – the offloading energy cost based on different computing capabilities (from 1 to 5) on the Cloudlet under different task load levels – low load 5.6a, medium load 5.6b and high load 5.6c, respectively. A higher energy cost value indicates that a larger number of offloading tasks are rejected and executed locally on the mobile devices. The results in Figure 5.6a and 5.6b show that the Cloudlet-based solution requires at least three units in the low load case and five units in the medium load case to ensure no offloading requests are denied. However, the proposed hybrid solution only requires two units in the low load processing and three units in the medium load execution, due to the remote offloading by utilizing Cloudlet communication units. In the high load offloading scenario, the proposed model can achieve full offloading in 5 computing units, while the Cloudlet-based model cannot support full offloading under 5 unit computing resources.

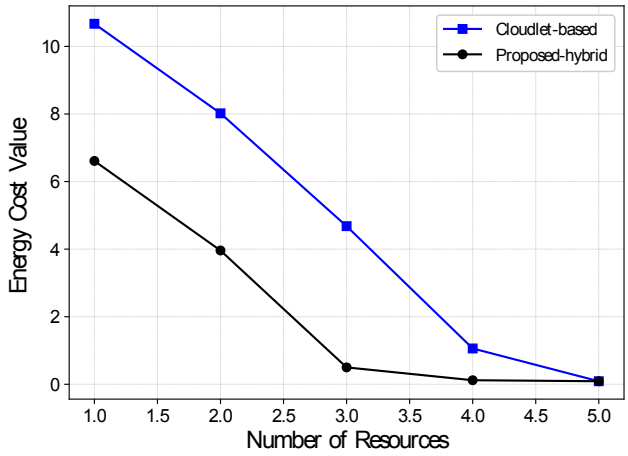
The impact of the communication capability is shown in Figure 5.7 for experiment 1.3. The task load is also divided into three levels, and the energy value reflects the rejection of offloading tasks. In the low load case, the Cloud-based offloading requires 40 communicating units to process all offloading tasks, while the proposed model did not require communicating resources due to the usage of the local Cloudlet computing utility. In the medium load case, the Cloud-based offloading cannot achieve full offloading even when the available communicating resources reach 50 units, while the proposed model can still process all offloading tasks through Cloudlet computing resources. In the high load case, the proposed hybrid model can achieve full offloading on 50 communication units while the Cloud-based model can only process less than half of the offloading tasks.

### 5.2.2 Heterogeneous Resource Allocation Performance Evaluation

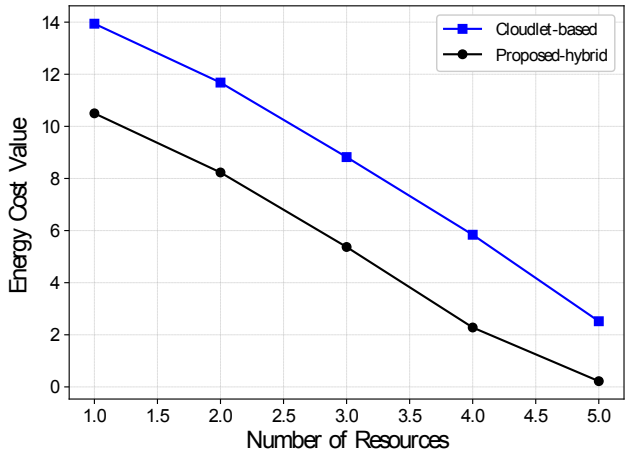
The second experiment set concerns the heterogeneous offloading resource allocation performance on the Cloudlet based on execution efficiency. In this experiment, we compare the proposed heterogeneous offloading resource allocation algorithm with three different prioritized algorithm counterparts – migration-based, partitioning-based and priority-based. To compare the performance fairly, we assume the device-based solutions can calculate the minimum resource requirements to meet the task-based request deadline




(a) Low Task Load (lambda = 40)

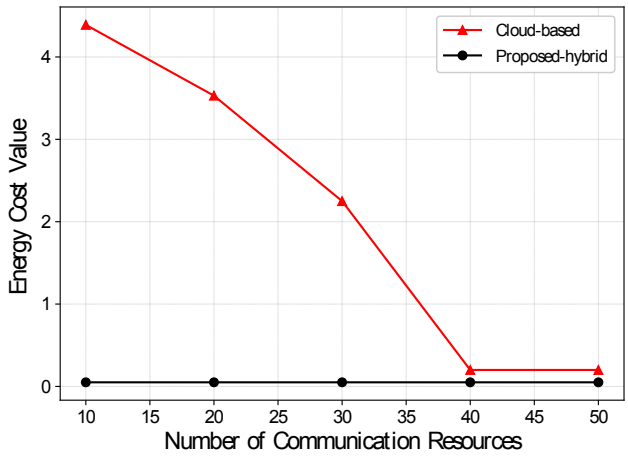



(b) Medium Task Load (lambda = 80)

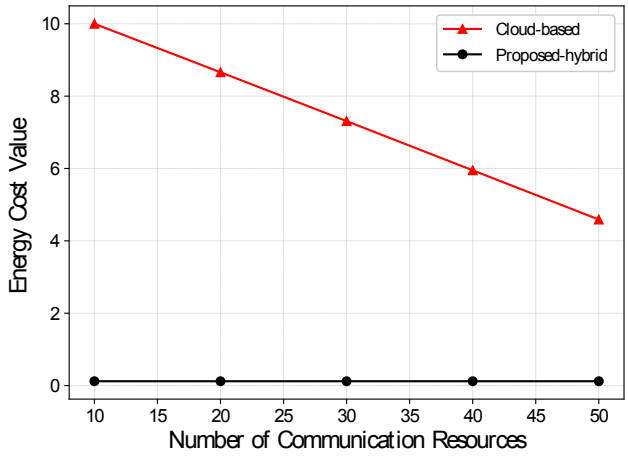



(c) High Task Load (lambda = 120)

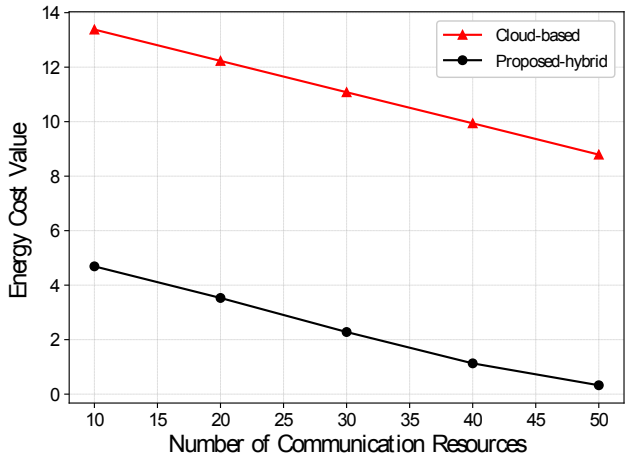
Figure 5.6: Offloading Performance Evaluation based on different Numbers of Cloudlet-based Computing Resources and Task Load ($c_{block} = 30$ )




(a) Low Task Load (lambda = 40)




(b) Medium Task Load (lambda = 80)




(c) High Task Load (lambda = 120)

Figure 5.7: Offloading Performance Evaluation based on different Numbers of Communication Resources and Load ($C = 4$ )

in advance, and then designate as many resources as possible to the migration-based offloading request. The task-based protocol prioritizes the highest load partitioning-based offloading requests, and provides the minimal resources for migration-based offloading requests. The load-based algorithm allocates resources according to the overall task load. In the first experiment 2.1, the effectiveness of the proposed PSO-based resource allocation algorithm is evaluated according to different numbers of particles in the swarm. Then, the second experiment 2.2 tests and observes the performance of the proposed algorithm with the other three counterparts, based on different load distributions.

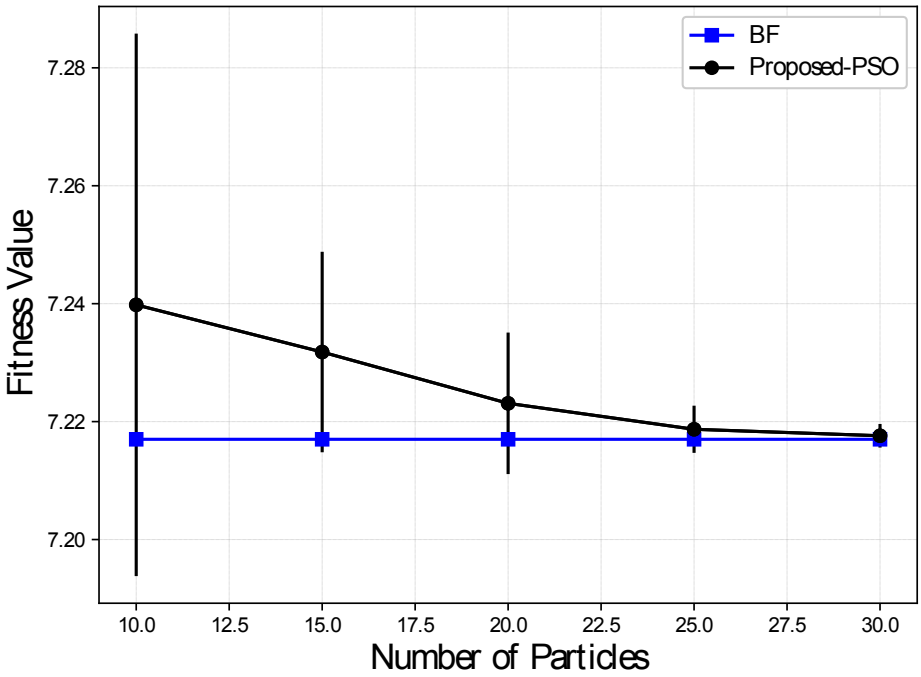
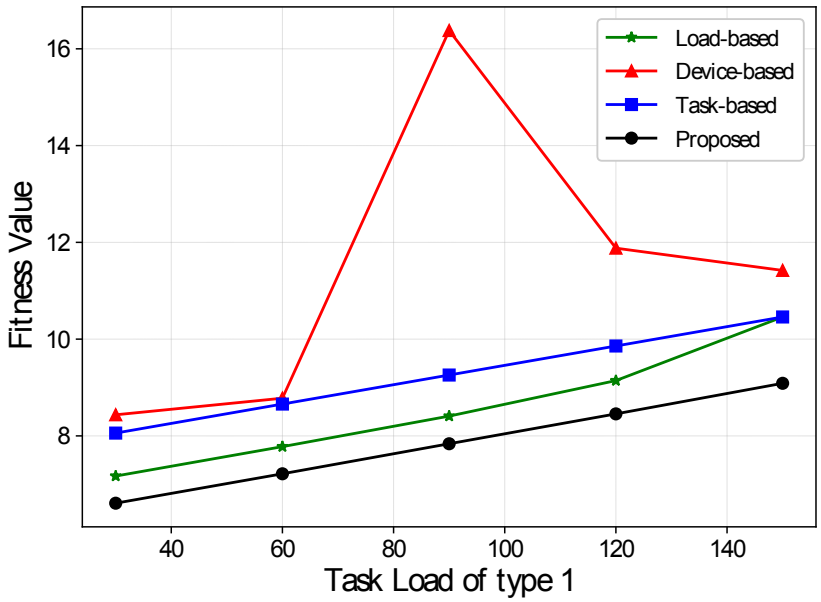
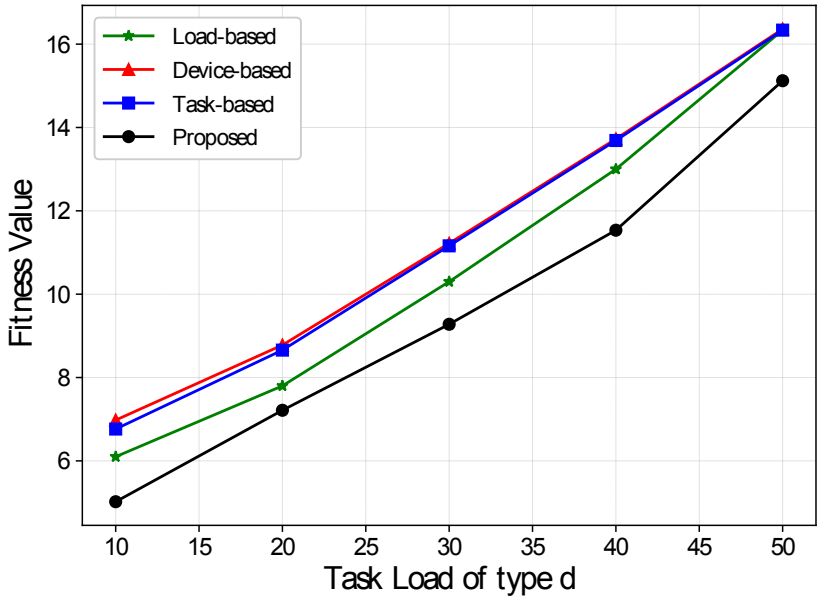



Figure 5.8: PSO-based Resource Allocation Performance Evaluation based on different Number of Particles

The results of experiment 2.1 are shown in Figure 5.8. The performance of the proposed algorithm is compared to the Brutal Force (BF) algorithm while the best solution is being searched for. The straight blue line of the BF suggests that the best solution should achieve the fitness value 7.217. With the number of particles increasing from 10 to 30, the proposed algorithm can achieve a mean fitness value between 7.24 and 7.218 with the standard deviation from 0.046 to 0.002. The results suggest that the proposed heuristic algorithm can effectively achieve near-optimal performance as the brutal force when utilizing 30 particle units.




(a) Performance based on the Load Change of Task 1




(b) Performance based on the Load Change of Task $d$

Figure 5.9: Heterogeneous Offloading Resource Allocation Performance Evaluation based on the Increasing of Task Load

In experiment 2.2, Figure 5.9 presents the offloading resource allocation performance for the four algorithms, focusing on the offloading execution efficiency. The fitness value is calculated according to the equation 5.32, based on the partitioning-based task 1 load change in 5.9a and migration-based task $d$ load change in 5.9b. As shown in 5.9a, the device-based algorithm shows the worst performance due to the ignorance of partitioning-based task load change. The peak value on $\lambda_{t_1} = 90$ indicates that the scheduled processing time is approaching the task deadline because of the unbalanced allocation strategy. Although the load increases, the fitness value decreases after the peaking point because the resources are forced to migrate from over-provisioning tasks to under-provisioning tasks to meet the task deadline. The task-based solution and load-based solution outperform the device-based solution because they prioritize fine-grain task-level requests. The load-based solution can obtain an even better result because it takes an extra step to balance the load among task-based requests. The task-based algorithm and the load-based algorithm achieve the same value on $\lambda_{t_1} = 150$ because the task requests of type 1 dominate the task load. In this case, only the minimal number of resources are allocated for task type 2 to meet the task deadline, while the rest are all scheduled to task type 1. The proposed algorithm outperforms the rest because it can investigate the potential cooperation between task-based offloading and device-based offloading, and balance the load on the system level, as shown in the equations 13 to 20. Similar performance can be observed in 5.9b, as with the device-based task load increasing. The device-based, task-based and load-based solutions eventually suggest the same resource allocation on $\lambda_d = 50$ because the task load is dominated by the device-based task requests.

### 5.2.3 Prediction-based Dynamic Resource Allocation Performance Evaluation

The third experiment set evaluates the proposed prediction-based dynamic offloading resource allocation performance on the Cloudlet for the execution efficiency. The task load utilized in this experiment is from the real world Wikipedia hourly web request record [232] in January 2012. The task type $t_{fr}$ and $t_{ja}$ is the number of requests to the French main page and Japanese main page respectively. In the proposed model, we use a record from 19 workdays as the training set (from Jan 2 to Jan 26) and the remaining three continuous workdays data (Jan 27, 30, 31) as the test set. In the traditional probability-based offloading comparison models, the mean value of the day (Jan 27, 30 and 31) is utilized as the predicted value of the incoming task load. We assume that the

load distribution is pre-known in the real-processing comparison model, which gives the optimal solution. The naive process baseline does not adapt load prediction. Instead, it utilizes the real-processing allocation strategy of the previous time slot.

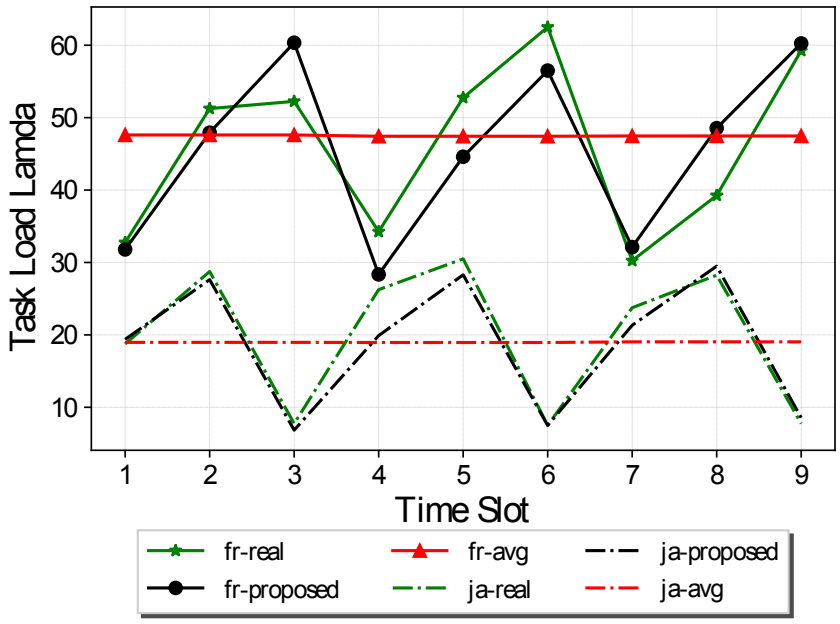



Figure 5.10: Task Load Prediction Results

The load prediction results are presented in Figure 5.10. According to Algorithm 8, the system utilizes the $< 1, 1, 1, 0, 1, 1 >_{24}$ SARIMA model, which achieves the best SSE and p-value. The time slots are the average load values during the daily period 2-5, 8-11 and 14-17 of each test workday (Jan 27, 30, 31). The results suggest that the proposed prediction model can properly react to the load change during the day.

The prediction impact on the resource allocation is depicted in Figure 5.11. In this experiment, the offloading process adopts two migration-based tasks $t_{fr}$, $t_{ja}$ with the load observed in 5.10. To focus on the prediction performance and simplify the offloading process, we assume that the offloading requests are returned to the mobile devices for local execution if the Cloudlet cannot execute the offloading request within the deadline. As shown in the results, the non-prediction allocation method cannot properly react to the unpredictable load change between two adjacent time slots – at the points 5, 7 and 9 it requires double executing time of the real process. The probability-based solutions can effectively achieve near optimal allocation solution under slight task load fluctuation.

However, this solution cannot handle the uncommon load change effectively at points 6, 7 and 9, leading to 50% to 90% additional execution cost. The proposed model can achieve the optimal allocation solution at points 1, 2, 4, 7, 9 and near-optimal solutions at the remaining points. In the overall nine rounds of allocation, the proposed model eventually achieves only 4.5 percent extra execution cost than the optimal real-process solution.

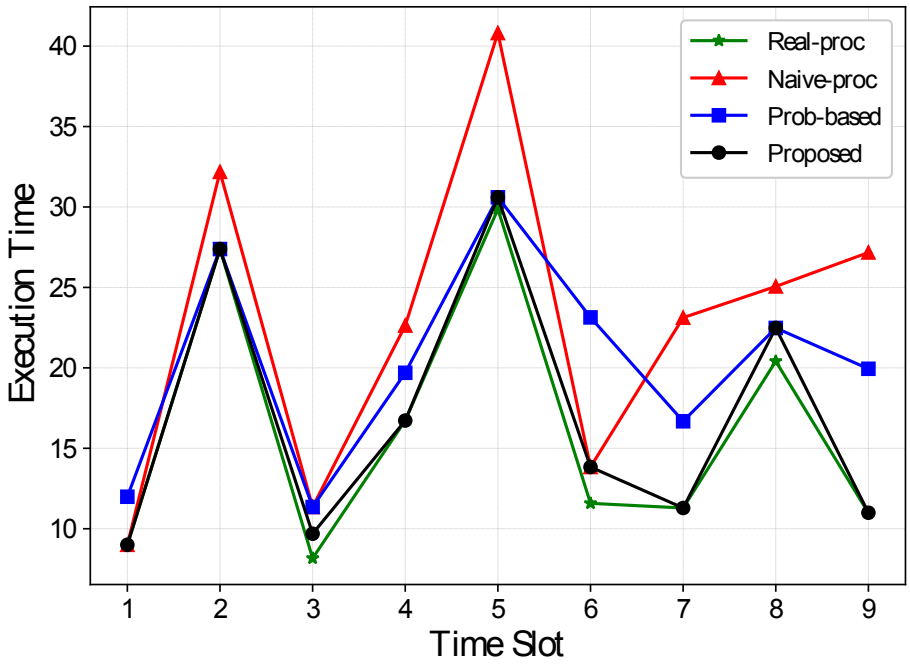



Figure 5.11: Offloading Resource allocation Performance based on the Load Prediction

## 5.3 Conclusion

In this chapter, a Cloudlet-based hybrid heterogeneous offloading model is proposed. The system is first discussed in terms of the core functioning components. The offloading behaviours are then analyzed based on the queueing model, and the energy-aware offloading allocation problem is formalized and solved by the PSO heuristic. A time series-based prediction model is integrated to achieve proactive offloading. The enabling of proposed heterogeneous offloading and fine-grain prediction benefits the overall energy-aware time-constraint offloading decision process. Based on the the experiment results,

the proposed system outperforms its counterparts on execution and energy efficiency. The current model can be further extended to include the mobile computing elements, concerning the live migration issue in the applications such as High Throughput Computing system [8], 5G technologies [97] and Cloud Radio Access Networks [85].

# Chapter 6

# A Mobility-aware Energy-efficient Model for MEC-based Offloading

In this chapter, a mobility-aware MEC-based model is propsoed, which concerns the intra- and inter-Cloudlet heterogeneous offloading process in a priority-based queueing system, aimed at optimizing energy and execution efficiency under time constraints while minimizing the number of offloading service rejections introduced by UE mobility.

The remainder of the sections are organized as follows. In Section 6.1, the system architecture and its main functioning components are presented, and the mobility-aware MEC-based heterogeneous offloading problem is formulated. Then, the intra-Cloudlet offloading scheduling issue is further explored. After that, the inter-Cloudlet load balancing problem is discussed in Section 6.2. In Section 6.3, the performance of the proposed model is compared to the contemporary solutions. In the last section, a remark is drawn for this chapter.

## 6.1 System Formulation

In this section, the proposed mobility-aware MEC-based energy- and execution-efficient offloading model is discussed in detail. First, the chapter introduces the system architecture with the main component functions. Second, the work formulates the MEC-based offloading process in the queueing model. Then, according to the queueing model, the intra-Cloudlet prioritized task allocation issue and the inter-Cloudlet multi-resource selection issue are respectively analyzed and solved. The notations used in this chapter can be found in Table 6.1.

Table 6.1: Parameter Definition

| **Parameter** | **Definition** |
|---|---|
| $UE_n$ | n's user equipment |
| $\lambda_{U_n}$ | arriving rate of tasks on n's UE |
| $\mu_f$ | filtering rate on UE |
| $\mu_{t_n}$ | transmission rate on n's UE |
| $P_{t_n}$ | transmission probability on n's UE |
| $P_{c_n}$ | computing probability on n's UE |
| $\lambda$ | overall arriving rate of tasks on Cloudlet |
| $\lambda_n$ | arriving rate of type n task on Cloudlet |
| $N$ | number of UE covered by local Cloudlet |
| $P_l$ | computing probability on local Cloudlet |
| $P_{cl}$ | computing probability on neighbor Cloudlet |
| $P_c$ | computing probability on remote Cloud |
| $C_N$ | computing capability on Cloudlet N |
| $\mu_n, E[\tau_n]$ | service rate/time of type n task in $M/G/1$ |
| $\rho_n$ | utility ratio of task type n in $M/G/1$ |
| $E[C_{p_n}]$ | overall offloading cost of task type n |
| $E[T_{p_n}]$ | processing cost of task type n |
| $E[T_{c_n}]$ | communicating cost of task type n |
| $\mu_{p_n}$ | process rate of on n's UE |
| $t_{n_d}$ | task load of type n |
| $E_{n_p}$ | process power of UE n |
| $P[Q = n]$ | probability of n tasks in the UE's |
| $E[c_n]$ | mean number of tasks in UE's processing queue |
| $T$ | number of priority classes in $M/G/1$ |
| $\lambda_l$ | arriving rate in $M/G/1$ |
| $\lambda_{l_t}$ | arriving rate of priority t task in $M/G/1$ |
| $E[W_t]$ | mean waiting time of priority t in $M/G/1$ |
| $E[Res]$ | mean residual time in $M/G/1$ |
| $E[N_t]$ | mean number of priority t tasks in $M/G/1$ |
| $E[M_t]$ | new arrival of priority t tasks during $E[W_{t+1}]$ in $M/G/1$ |
| $C_{\tau_{l_t}}$ | coefficient of service time in $M/G/1$ |
| $N'_{bw}$ | total bandwidth unit of Cloudlet N' |
| $N'_{UE}$ | shared bandwidth unit for UE of Cloudlet N' |
| $t_{n_s}$ | size of type n task |
| $R_w$ | Cloudlet to Cloud data rate |
| $D_w$ | Cloudlet to Cloud media delay |
| $E[C_{e_n}]$ | overall energy cost of task type n |
| $e_{p_n}$ | local process energy cost of task type n |
| $e_{t_n}$ | transmitting energy cost of task type n |
| $e_{w_n}$ | waiting energy cost of task type n |
| $\tau_{n_{sm}}$ | UE contact time(Smooth Mobility) |
| $t$ | UE mobility time counter |
| $X_n(t)$ | UE position at time t |
| $X_{cl}$ | Cloudlet position |
| $R_{cl}$ | Cloudlet coverage radius |
| $f_V(v)$ | UE velocity pdf function |
| $\alpha/\beta$ | UE stationary/move probability |
| $f_D(\phi)$ | UE direction pdf func |
| $\mu_V$ | UE speed change frequency |
| $\mu_D$ | UE direction change frequency |
| $f_V(a)/f_D(a)$ | UE speed/direction acceleration function |
| $\Delta t$ | UE curve interval |
| $t_{n_b}$ | task deadline of type n |

### 6.1.1 System Overview and Main Functioning Components

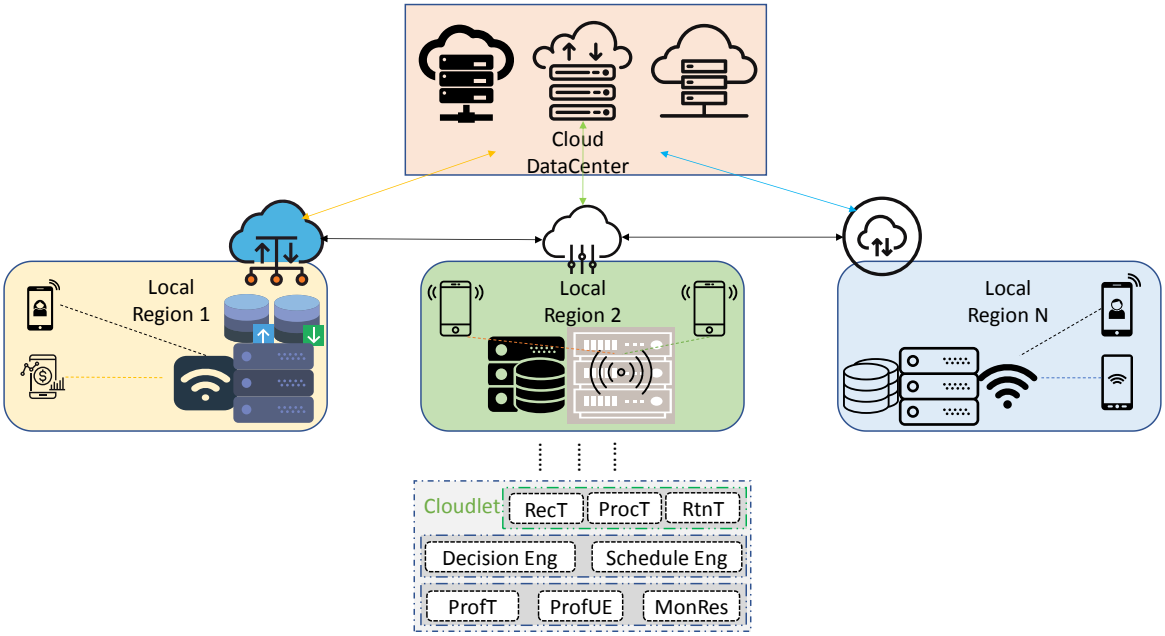



Figure 6.1: MEC-based Offloading System

The work considers the MEC-based offloading environment as shown in Figure 6.1, where the system consists of multiple Cloudlets and centralized Cloud servers. In work [220], the Cloudlet was defined as the edge computing infrastructure that could execute offloading requests for UEs under the local coverage. In this chapter, the work utilizes the general term Cloudlet to represent the regional computing capacity such as macro base stations, small cell base stations, fog servers, and other forms of local stationary computing resources that are accessible by UEs via local wireless networks. As shown in the figure, the regional Cloudlets and Cloud servers are connected through stable backhaul networks. The UEs are covered by local Cloudlets (WiFi) and the edge Cloud (cellular).

In each regional Cloudlet, the functions can be layered into three aspects: offloading task processing, offloading optimization, and information profiling. The offloading task processing handles the task receiving, task execution, and task return, towards the UEs. The offloading optimization layer handles the two core offloading decision processes – the inter-Cloudlet resource selection and intra-Cloudlet priority-based scheduling. The underlying profiling layer collects the information of UEs, tasks, and resources in the system, according to which the offloading decisions are made.

### 6.1.2 Queueing-based MEC Offloading Processing Formulation and Assumptions

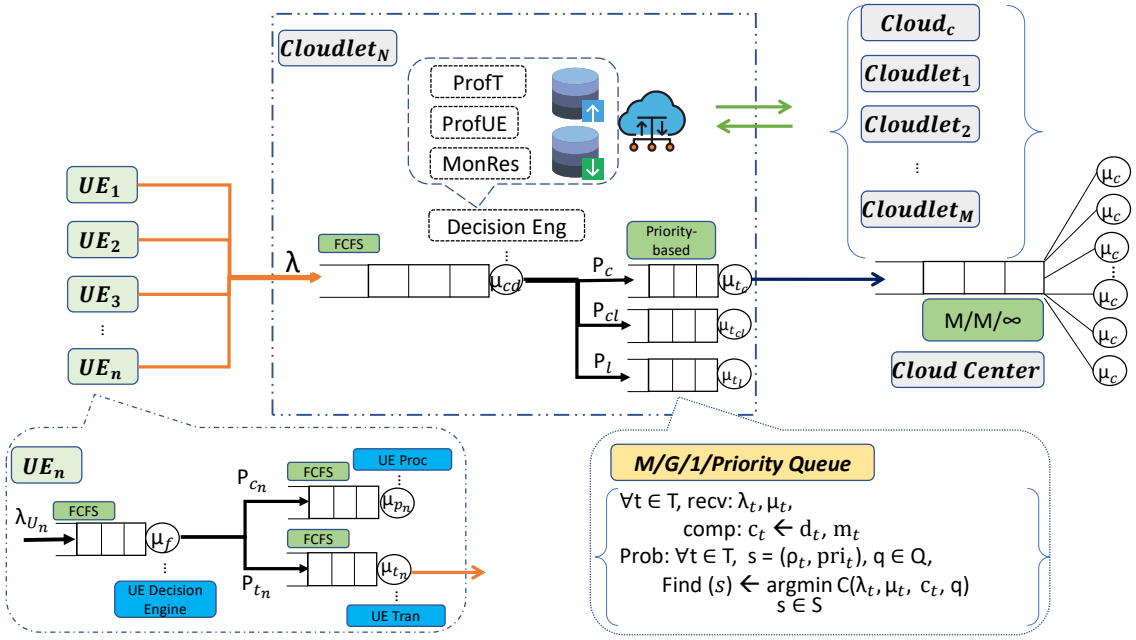



Figure 6.2: MEC-based Offloading Process

The offloading processing is depicted in Figure 6.2, which involves one decision process on the UE and two scheduling processes on the Cloudlet. The offloading request is initiated by the UE Decision Engine on the UE, which filters out the requests that cannot be offloaded, due to hardware constraints, privacy, and security concerns. Consequently, the unoffloadable requests are processed via the UE processors, while the offloading tasks are transmitted to the local Cloudlet. In this chapter, the work adopts the common assumption that the tasks triggered in the $UE_n$ follow a Poisson process with mean rate $\lambda_{U_n}$ and the service time in filtering and transmission follows the exponential distribution with rate $\mu_f$ and $\mu_{t_n}$ respectively. The work also assumes that for every UE, the processing queue model can achieve stable states with the following constraints:

$$\lambda_{U_n} < \mu_f, \tag{6.1}$$

$$\lambda_{U_n} * P_{t_n} < \mu_{t_n}, \tag{6.2}$$

$$\lambda_{U_n} * P_{c_n} < \mu_{p_n}, \tag{6.3}$$

where $P_{t_n}$ and $P_{c_n}$ stand for the transmitting probability and processing probability, respectively. In order to concentrate on the scheduling behaviors on the Cloudlet, the work simplifies the task arriving rate on the Cloudlet from UE's as $\lambda$, which equals to the sum of all tasks received from currently covered $N$ UEs:

$$\lambda = \sum_{n=1}^{N} \lambda_n. \tag{6.4}$$

For each type of task $\lambda_n$, the Decision Engine on the Cloudlet makes inter-Cloudlet allocations on the received tasks. Based on the decision, the tasks are assigned to the current Cloudlet, neighbouring Cloudlets or the remote Cloud, according to the system load, task type and UE mobility pattern. The local computing resources then execute the tasks on the current Cloudlet queue in a priority-based manner, while the tasks allocated to other Cloudlets and the remote Cloud are migrated to the corresponding destinations.

Concerning the tasks scheduled on the Cloudlets, the work also assumes that the service rate for each single task type in the processing utilities follows an exponential distribution [179]. Because the Cloudlet is required to process multiple tasks from different UEs, so the $M/G/1$ queue is adopted and the constraints are set as follows:

$$1 > \rho = \sum_{n=1}^{N} \rho_n, \tag{6.5}$$

$$\rho_n = \lambda_n * P_l/\mu_n = \lambda_n * P_l * E[\tau_n], \tag{6.6}$$

where $\rho_n$ is the server utilization ratio of the task type n, $\mu_n$ is the mean service rate of task type n. $P_l$ is the probability to compute on the local Cloudlet.

In terms of the tasks forwarded to the Cloud (with probability $P_c$), the work models the Cloud server as a $M/M/\infty$ queue, since the Cloud is designated to provide "unlimited" resources to end users. In this case, tasks on the Cloud can be processed without waiting in the queue.

### 6.1.3 Intra-Cloudlet Task Allocation

The intra-Cloudlet task allocation refers to the scheduling process in the $M/G/1$ priority queue. Distinct from previous works that assume exponential distributed service time or FCFS (first come first serve) execution manner, the proposed queueing model allows tasks to be served with different processing rates and priorities in the multi-user MEC-based offloading scenarios. In order to achieve energy- and execution-efficient offloading, the work will discuss the execution cost, energy cost, and mobility penalty

respectively. Then, the energy- and execution-efficient offloading allocation problem is formulated, and a PSO-based solution is proposed.

**Execution Cost**

In this section, the focus is on the offloading time cost in the system. The task offloading cost is defined as the period between the time when the task is triggered on the UE and the time when the task result is acknowledged on the UE. Given that the size of the offloading result is negligible compared to its corresponding migration size, it ignores the resulting return time and energy cost so that the communication cost only involves the task migration (uploading). In this case, the work defines the execution cost for the task type $n$ as the sum of the processing cost and the communication cost during the offloading:

$$E[C_{p_n}] = E[T_{p_n}] + E[T_{c_n}]. \tag{6.7}$$

As shown in the offloading processing map in Figure 6.2, the work will discuss the offloading execution cost based on the three offloading processing utilities – the local UE, the Cloudlet, and the remote Cloud server.

**Local Execution** The simplest execution scenario occurs when the tasks are filtered to the local mobile CPUs. In this case, no task transmission is required, and the overall execution cost is proportional to the task processing cost. The task processing on the UE is modeled as a $M/M/1$ FCFS queue with Poisson arrival rate $\lambda_{U_n} * P_{c_n}$ and service rate $\mu_{p_n}$. Given the task load as $t_{n_d}$ and the UE processing power as $E_{n_p}$, the service rate can be set as:

$$\mu_{p_n} = t_{n_d}/E_{n_p}. \tag{6.8}$$

Given the constraints 6.1 to 6.3, the expected number of tasks in the system can be calculated based on its geometric distribution:

$$P[Q = n] = (1 - \frac{\lambda_{U_n} * P_{c_n}}{\mu_{p_n}})(\frac{\lambda_{U_n} * P_{c_n}}{\mu_{p_n}})^n, \tag{6.9}$$

$$E[c_n] = \sum_{n=0}^{\infty} n * P[Q = n] = \frac{\lambda_{U_n} * P_{c_n}}{\mu_{p_n} - \lambda_{U_n} * P_{c_n}}. \tag{6.10}$$

In this case, the processing time in the system can be obtained according to the Little's formula:

$$E[T_{p_n}] = E[c_n]/(\lambda_{U_n} * P_{c_n}) = \frac{1}{\mu_{p_n} - \lambda_{U_n} * P_{c_n}}. \tag{6.11}$$

**Cloudlet Execution** In terms of the offloading processing on the Cloudlet, given the task arrival rate $\lambda$ on the Cloudlet as shown in Formula 6.4, the incoming task rate in the $M/G/1$ queue also follows a Poisson distribution and consists of $T$ ($T \leq N$) priority classes (different tasks may belong to the same priority class):

$$\lambda_l = \lambda * P_l = \sum_{n=1}^{N} \lambda_n * P_l = \sum_{t=1}^{T} \lambda_{l_t} \tag{6.12}$$

Assuming that tasks in the same class follow FCFS discipline, the expected waiting time of the highest priority tasks ($t = 1$) $E[W_{t_1}]$ can be calculated as follows:

$$E[W_{t_1}] = E[Res] + E[N_{t_1}]E[\tau_{t_1}], \tag{6.13}$$

where $E[Res]$ is the residual service time found in service and $E[N_{t_1}]$ is the mean number of tasks of priority 1.

According to Little's formula, the average number of tasks in the queue is equal to the product of the arrival rate and mean waiting time:

$$E[N_{t_1}] = \lambda_{t_1} * E[W_{t_1}], \tag{6.14}$$

It can obtain the average waiting time for the tasks of the first order priority through 6.6, 6.13 and 6.14 as:

$$E[W_{t_1}] = \frac{E[Res]}{1 - \rho_{t_1}}. \tag{6.15}$$

Given $E[M_{t_1}]$ as the number of new incoming tasks of type 1 during $E[W_{t_2}]$, the average waiting time for the second priority of tasks can be recursively calculated as:

$$E[W_{t_2}] = E[W_{t_1}] + E[N_{t_2}]E[\tau_{t_2}] + E[M_{t_1}]E[\tau_{t_1}], \tag{6.16}$$

where $E[M_{t_1}]$ can be calculated as:

$$E[M_{t_1}] = \lambda_{t_1} * E[W_{t_2}]. \tag{6.17}$$

Similar to Formula 6.15, $E[W_{t_2}]$ can also be solved by adopting the Little's formula as:

$$E[W_{t_2}] = \frac{E[Res]}{(1 - \rho_{t_1})(1 - \rho_{t_1} - \rho_{t_2})}. \tag{6.18}$$

Adopting the same process, the tasks of priority $n$ can be computed as:

$$E[W_{t_n}] = \frac{E[Res]}{(1 - \sum_{t=1}^{n} \rho_t)(1 - \sum_{t=1}^{n-1} \rho_t)}. \tag{6.19}$$

Given that the residual processing time $E[Res]$ is the remainder of any type of tasks found in execution, the value of $E[Res]$ can be calculated as:

$$E[Res] = \frac{\lambda_l E[\tau^2]}{2}, \tag{6.20}$$

where $E[\tau^2]$ is the second moment of processing time of all types of tasks. According to Formula 6.12, the value of $E[\tau^2]$ can be calculated by summing up the fraction of tasks of each type:

$$E[\tau^2] = \frac{1}{\lambda_l} \sum_{t=1}^{T} \lambda_{l_t} E[\tau_{l_t}^2]. \tag{6.21}$$

Given the coefficient of service time variation as $C_\tau^2 = \sigma^2/E[\tau]$, it can consequently calculate the expected waiting time of priority $n$ tasks according to 6.19 and Pollaczek-Khinchin mean value formulas as:

$$E[W_{t_n}] = \frac{\sum_{j=1}^{T} \lambda_{l_j} E[\tau_{l_j}]^2 * (1 + C_{\tau_{l_j}}^2)}{2(1 - \sum_{j=1}^{n} \rho_{t_j})(1 - \sum_{j=1}^{n-1} \rho_{t_j})}. \tag{6.22}$$

Finally, the overall time spent on the Cloudlet execution is calculated as:

$$E[T_{p_n}] = E[\tau_{t_n}] + E[W_{t_n}] \tag{6.23}$$

In terms of the communication cost of the Cloudlet-based offloading, the work assumes that the size of the task that needs to be migrated as $t_{n_s}$ and the data transmission rate for the UE $i$ under Cloudlet $j$ as $r_{ij}$, the transmission cost for the task type $n$ can be calculated as in formula 6.24:

$$E[T_{c_n}] = t_{n_s}/r_{ij}, \tag{6.24}$$

where the transmission rate is calculated as in Formula 6.25:

$$r_{ij} = \epsilon W_j log_2(1 + \frac{p_{ij} * H_{ij}}{\sigma^2}) \tag{6.25}$$

$\epsilon$ ($\in (0, 1]$) is a variable that reflect the gap between real data transmission rate and the achievable channel capacity and the gap is typically introduced by modulation and encoding [141]. $W_j$ is the channel bandwidth of Cloudlet $j$. $p_{ij}$ is the transmission power of UE $i$ to Cloudlet $j$. $H_{ij}$ is the channel gain between UE $i$ and Cloudlet $j$. $\sigma$ is the Gaussian noise.

**Cloud Execution** As discussed in the previous section, the Cloud server is a $M/M/\infty$ queue that can provide theoretically "unlimited" computing capability to users. In this

case, the Cloud-based offloading time cost only involves the task transmission time between the UE and the Cloud, which includes two task migration steps – from UE to Cloudlet and from Cloudlet to Cloud. Given the stable wired network from Cloudlet to Cloud with data rate $R_w$ and media delay $D_w$, the migration time from local Cloudlet to remote Cloud can be calculated as:

$$E[T_{c_{rn}}] = t_{n_s}/R_w + D_w. \tag{6.26}$$

In this case, the overall transmission cost can be obtained via 6.24 and 6.26:

$$E[T_{c_n}] = E[T_{c_{rn}}] + t_{n_s}/r_{ij}. \tag{6.27}$$

**Overall Execution Cost** In summary, based on the calculation Formula 6.11, 6.23 and 6.27, Formula 6.7 can be extended as below, for the case of local execution, Cloudlet execution and Cloud execution, respectively:

$$E[C_{p_n}] = \begin{cases} E[T_{p_n}](6.11) & \forall n \in UE, \\ E[T_{p_n}](6.23) + E[t_{c_n}](6.24) & \in Cloudlet, \\ E[T_{c_n}](6.27) & \in Cloud. \end{cases} \tag{6.28}$$

**Energy Cost**

In this chapter, the term "energy cost" specifically refers to the energy consumption on the UEs during task offloading. In order to evaluate the energy cost during offloading, the work defines the $n's$ UE energy consumption tuple $C_{e_n}$ as $< e_{p_n}, e_{t_n}, e_{w_n} >$, representing the energy cost on local processing power, task transmission power, and idle power waiting for execution results, respectively, where $e_{t_n}$ equals $p_{nj}$, for that specific Cloudlet $j$. Different UE models may utilize different power consumption specifications, which can be read via the component power profile provided by the corresponding manufacturers, as discussed in our previous work [53]. The energy cost can be calculated similarly to the execution cost as follows, for the offloading situation in UE, Cloudlet and Cloud, respectively:

$$E[C_{e_n}] = \begin{cases} E[T_{p_n}](6.11) * e_{p_n} & \forall n \in UE, \\ E[T_{p_n}](6.23) * e_{w_n} + E[t_{c_n}](6.24) * e_{t_n} & \in Cloudlet, \\ E[T_{c_n}](6.27) * e_{t_n} & \in Cloud. \end{cases} \tag{6.29}$$

As shown in Formula 6.29, the expected energy cost in the local UE processing is the product of mean processing time $E[T_{p_n}]$ and the corresponding processing cost $e_{p_n}$. In

the case of Cloudlet processing, the energy cost is the sum of the transmission cost $E[t_{c_n}] * e_{t_n}$ and the waiting cost $E[T_{p_n}] * e_{w_n}$. The Cloud-based processing only requires the transmission cost $E[T_{c_n}] * e_{t_n}$ since it can provide "unlimited" resources to execute the offloading task.

Given the minimum average task processing time on Cloudlet and Cloud as $t_{off}$ ($\min(E[C_{p_n}], n \in UE | E[C_{p_n}], n \in Cloud$) and the minimum offloading processing energy as $E_{off}$ ($\min(E[C_{e_n}], n \in UE | E[C_{e_n}], n \in Cloud$), the tradeoff between energy consumption and delay highly depends on the communication overhead and the execution capability between the mobile device and the Cloudlet and Cloud:

$$Decision = \begin{cases} non-offloading & E[C_{p_{ue}}] < t_{off},\ E[T_{p_{ue}}] < E_{off}, \\ offloading & E[C_{p_{ue}}] > t_{off},\ E[T_{p_{ue}}] > E_{off}, \\ tradeoff & others \end{cases} \tag{6.30}$$

where $E[C_{p_{ue}}]$ is the $E[C_{p_n}]$ and $E[T_{p_{ue}}]$ is the $E[C_{e_n}]$ when $n \in UE$. In this chapter, the focus is on the condition that tasks are sent to the Cloudlets by the mobile devices.

### Mobility-based Contact Time and Task Deadline

In the real offloading scenarios, the connectivity between the UE and the Cloudlet is not guaranteed to be continuous during the offloading execution. The mobility patterns of the UEs may lead to significant downgrading of the offloading performance if the offloading execution is interrupted by the intermittent connections. The offloading tasks are typically dropped if they cannot be accomplished during the contact time before disconnection occurs, introducing additional waiting costs and energy consumption. In this chapter, the work adopts the SMOOTH random mobility model [25] into the local UEs such that the offloading task contact time can be calculated, which is further utilized as one of the parameters to prioritize the offloading execution order, as shown in Formula 6.22. Unlike the memoryless mobility models such as Random Walk that may introduce unrealistic sudden stops and sharp turns, the SMOOTH model can correlate the new speed and direction values to the previous records to achieve realistic turns and movements.

Based on the mobility model, the work defines the offloading contact time $\tau_{sm}$ as the period when the $n's$ UE remain in contact with the Cloudlet before moving out of the range, given that the offloading starts at time zero ($t = 0$):

$$\tau_{n_{sm}} = min\ t\{t - 1 : \|X_n(t) - X_{cl}\| > R_{cl}\}, \tag{6.31}$$

where $t$ is the time counter, $X_n(t)$ is the position of $n's$ UE , $X_{cl}$ is the position of

the Cloudlet and $R_{cl}$ is the coverage radius of the Cloudlet. The velocity of the UE is initialized according to the preferred speed pdf as:

$$f_V(v) = \begin{cases} \alpha & v = 0, \\ \beta & v = v_{mov}, \\ \frac{1-\alpha-\beta}{v_{mov}} & 0 < v < v_{mov}, \end{cases} \tag{6.32}$$

where $f_V(v = 0)$ describes the probability when UE is static, and $f_V(v = v_{mov})$ stands for the case when UE is moving. In addition to these two preferential velocities, the remaining part of the speed probability is assumed to be uniformly distributed, as shown in Formula 6.32.

Similarly, the direction of the UE is also initialized according to an angle-based uniform distribution pdf:

$$f_D(\phi) = \frac{1}{2\pi}, \ \forall\phi, \ 0 \leq \phi \leq 2\pi. \tag{6.33}$$

The speed and direction change over time according to a Poisson-based frequency with mean $\mu_V$ and $\mu_D$ for speed update and direction update, respectively. During the updating phase, a target speed $v^*$ will be randomly selected according to the pdf Formula 6.32 while the work assumes a fixed target direction $d^*$ for the UE, since the UE has a certain travelling objective. Given the current speed $v(t)$ and target speed $v^*$, the speed acceleration function can be defined as:

$$f_V(a) = \begin{cases} 1/a_{max} & 0 < a < a_{max}, \\ 1/a_{min} & a_{min} < a < 0, \\ 0 & others, \end{cases} \tag{6.34}$$

where $a_{max}$ and $a_{min}$ are the acceleration and deceleration boundaries.

In terms of the direction, the accelerator is calculated according to the curve time $\Delta t$:

$$f_D(a) = \frac{d^* - f_D(t)}{\Delta t}, \tag{6.35}$$

where $\Delta t$ follows a uniform distribution in the defined interval $[t_{min}, t_{max}]$.

In this case, given the task deadline $t_{n_b}$, the trajectory of the UE can be calculated according to Formulas 6.32 to 6.35. Then, based on Formula 6.31, the task deadline should be modified as follows:

$$t_{n_b} = \begin{cases} \tau_{n_{sm}} & \tau_{n_{sm}} < t_{n_b}, \\ t_{n_b} & others, \end{cases} \tag{6.36}$$

where the updated value $t_{n_b}$ represents the maximum offloading time that the UE can spend on the Cloudlet.

**Cost Function, Problem Formulation and PSO-based Solution**

Considering the offloading execution and energy efficiency, based on the results from Formulas 6.28 and 6.29, the work defines the joint cost function for the $n's$ offloading task as:

$$Coef(n) = (1-\gamma) * C_{p_n} * \Delta p + \gamma * C_{e_n} * \Delta e, \tag{6.37}$$

where $\gamma$ is utilized as the weight coefficient to balance the importance between offloading execution time and the offloading energy cost. In the case that the energy cost and time cost may result in different orders of magnitude, the work introduces the factors $\Delta p$ and $\Delta e$ to unify the order of magnitude.

Given the joint cost function in 6.37, the intra-Cloudlet offloading scheme aims to find the best task priority and utilization ratio for each offloading task $(pri_n, \rho_n)$ under coverage, such that the overall offloading joint cost is minimized. The allocation function is shown in:

$$F(s) = arg \min_{s \in S} \sum_{n=1}^{N} Coef(n). \tag{6.38}$$

s.t. $\forall n \in Cloudlet_{N'}$:

$$E[C_{p_n}] < t_{n_b}, \tag{6.39}$$

$$\sum_{n=1}^{N} \rho_n < 1, \tag{6.40}$$

where the solution in 6.38 is a $n$ dimensional vector pair $< (pri_1, \rho_1), (pri_1, \rho_1), ..., (pri_n, \rho_n) >$. The constraint 6.39 restricts the task offloading time to be less than its task deadline according to the equation 6.36, while the constraint 6.40 indicates that the overall resource utilization on the Cloudlet should not exceed its computing capability.

Unfortunately, the $n$ dimensional solution space makes the complete search such as Brutal Force infeasible, especially when the number of local UEs scales up. In order to promptly achieve the optimal or near-optimal allocation, the work adopts the PSO heuristic method to find the priority and utilization pair for the offloading tasks.

Concerning the constraint as shown in Formula 6.39, the work introduces the penalty function to punish the potential solution that may lead to violations of the task deadline. Given the fact that all dropping tasks (due to task time constraints 6.39) are to be processed on the UEs, the work defines the penalty function as the sum of the joint waiting cost and the joint reprocessing cost calculated by Formula 6.37:

$$\begin{aligned} P(s) = -\sum_{n}^{N^*} \{ & [E[T_{p_n}] * (1-\gamma) * \Delta p] + \\ & [E[T_{p_n}] * e_p * \gamma * \Delta e] \} \end{aligned} \tag{6.41}$$

where $N^*$ represents the amount of dropping tasks.

Given the allocation function 6.38 and the penalty function 6.41, the model can be define the fitness function as the difference between them:

$$fit(s) = F(s) - P(s). \tag{6.42}$$

The particle position space $P_{pos}$ is defined as a 3D vector $P_{pos} < x, y, z >$ where $x$-axis represents the ID of the virtual resource on the Cloudlet, $y$-axis stands for the type of the offloading task, and $z$-axis records the priority of the task. Concerning the constraint in Formula 6.40, the work sets the overall number of resources on the $x$-axis $< x_1, x_2, ...x_I >$ less than the computing capability on the Cloudlet $N$:

$$\sum_{i=1}^{I} x_i = C_N. \tag{6.43}$$

For each position in the position space, the work assigns a binary value $\{0, 1\}$ to represent the allocation of the task priority and resource utility ratio. For example, the assignment $P_{pos} < x', y', z' >= 1$ indicates that the task $y'$ is assigned to the resource $x'$ with priority $z'$.

Given the fact that each resource can only service one type of offloading task at one time, and one type of offloading task can only belong to one priority class, the work can further reduce the search space by restricting the particle position value as:

$$\sum_{y'=1}^{U_N} P_{pos}(x', y', z') <= 1, \forall x' \in X, \tag{6.44}$$

$$\sum_{z'=1}^{T} P_{pos}(x', y', z') <= 1, \forall y' \in Y. \tag{6.45}$$

For each position $P_{pos} < x', y', z' >$ in the position space, a velocity value $P_v < x', y', z' >$ is assigned accordingly, within the limit of $[-V_{max}, V_{max}]$.

During the offloading decision interval, the PSO first initializes the position space and velocity space by assigning values randomly. Then, based on the particle's past fitness value as calculated in Formula 6.42, the velocity and position matrix update in the following equations, for the $n's$ particle at the time $t + 1$:

$$\begin{aligned} P_{v_n}^{t+1}(x, y, z) =& wP_{v_n}^{t}(x, y, z) \\ &+ c_1 r_1 (pbest_{v_n}^{t}(x, y, z) - P_{v_n}^{t}(x, y, z)) \\ &+ c_2 r_2 (nbest_{v}^{t}(x, y, z) - P_{v_n}^{t}(x, y, z)), \end{aligned} \tag{6.46}$$

$$P_{pos_n}^{t+1}(x,y,z) = \begin{cases} 1 & \forall x' \in X, if: \\ & P_{v_n}^{t+1}(x,y,z) = max\{P_{v_n}^{t+1}(x',y,z)\}, \\ 0 & otherwise, \end{cases} \tag{6.47}$$

where $wP_{v_n}^t(x,y,z)$ is the inertia product based on the current value at time $t$. $c_1$ and $r_1$ are the learn factor and random factor to particle's past value $pbest_{v_n}^t(x,y,z)$ comparison while $c_2$ and $r_2$ are for the swarm global value $nbest_v^t(x,y,z)$ comparison, respectively.

### 6.1.4 Inter-Cloudlet Offloading

The intra-Cloudlet offloading process is discussed in the previous section. However, given the fact that some regional areas are covered by multiple Cloudlets, UE is required to decide its corresponding destination Cloudlet before the aforementioned intra-Cloudlet scheduling protocol starts to work.

**Algorithm 9:** Inter-Cloudlet Offloading Algorithm

```
1  # Selection Function
2  FuncSelect(UE_n, t_n, Cl_list):
3      ∀i ∈ Cl_list :
4          fit_i ← fit(Cl_i)
5      dst ← min(fit_i)
6  Return: dst
7  # Migration Function
8  FuncMigr(UE_n, t_n, Cl_list):
9      while λ*_n < λ_n do
10         Cl_b_list ← Cl_b_list + dst
11         Cl-list* ← Cl_list − Cl_b_list
12         dst* ← FuncSelect(UE_n, t_n, Cl_list*)
13         if Cl*_list is empty then
14             dst* ← rC
15             break;
16 Return: dst*
```

The inter-Cloudlet offloading algorithm is proposed in Algorithm 9, which includes the selection process and migration process. In the selection process, as shown in line 1 to line 6, the UE selects the default destination Cloudlet by predicting its fitness value on the available Cloudlets. It takes the input of UE mobility information $UE_n$ (used

in Section 3.3.3), task processing information $t_n$ (used in Section 3.3.1) and a list of Cloudlets $Cl\_list$ that the UE is currently under its coverage. For each Cloudlet in the Cloudlet list, the work predicts the task processing fitness value according to Formula 6.42. Then, the Cloudlet that contains the minimum fitness value is selected as the default Cloudlet. The run-time migration function starts from line 7, which utilizes the same input as the initialization function. The function is triggered when the UE detects that some offloading load requests are rejected, due to new arrival offloading loads. It first adds the current Cloudlet to the Cloudlet block list $Cl_b\_list$ and then repeats the initialization function to find the new destination Cloudlet that is not in the block list. In this case, the new destination is selected accordingly, to reduce the number of offloading request rejections. If the task loads cannot be fully processed by local Cloudlets, the remaining offloading tasks are forwarded to the remote Cloud $rC$.

## 6.2 Performance Evaluation and Discussion

In this section, the performance of the proposed offloading model is observed and evaluated in the context of intra-Cloudlet offloading (Section 4.2) and inter-Cloudlet offloading scenarios (Section 4.3), respectively. Before the evaluation starts, the task processing model and energy cost model are first validated (Section 4.1). Based on the validation results, in the intra-Cloudlet offloading scenarios, the proposed offloading model is compared to Min-Min heuristic, Min-Max heuristic [193], and Deadline-greedy algorithms [188] in terms of execution efficiency and energy efficiency. The work further divides the offloading cost into UE processing cost, transmission cost and waiting cost on Cloudlet, observing and analyzing the execution time and energy consumption of each sub-part. In the inter-Cloudlet offloading scenarios, the proposed resource allocation protocol is compared to distance-based allocation and priority-based allocation, evaluating the time cost and energy cost on the overall system level and the UE task, based on the task load change and task deadline change.

The validation task set is presented in Table 6.2, which is based on the work. The intra-Cloudlet related environment setting is shown in Table 6.3, and the inter-Cloudlet parameters are depicted in Table 6.4, the experiment settings are based on the works [220], [25], and [148].

Table 6.2: Experiment: Validation Experiment Set

| **Parameter** | **Definition and Default Value** |
|---|---|
| $e_{w_i}$ | UE $i$ Transmission Power (50 mW) |
| $e_{p_i}$ | UE Processing Power (640 mW) |
| $e_{t_n}$ | UE Idle Power (20 mW) |
| $d_{ij}$ | Distance between UE $i$ and Cloudlet $j$ (20 m) |
| $W_j$ | Bandwidth of Cloudlet $j$ (5 Mhz) |
| $\delta$ | Path Loss Factor (Channel Gain Factor) (4) |
| $H_{ij}$ | Channel Gain between UE $i$ and Cloudlet $j$ ($H = d_{ij}^{-\delta}$) |
| $\sigma^2$ | White Noise (-100dbm) |
| $\epsilon$ | Modulation and Encoding Factor (0.5) |
| $ue_{CPU}$ | UE CPU Cycles (1 G cycles/s) |
| $Cl_{CPU}$ | Cloudlet CPU Cycles (5 G cycles/s) |
| $t_l$ | Default Task Load Cycles on UE (3 G cycles) |
| $t_s$ | Default Task Size (1 MB) |
| $ol_n$ | Overlay Factor of Task $n$ (5.952) |
| $t_{s_n}$ | Total Task Size of Type $n$ including Overlay ($t_{s_n} = t_s * ol_n$) |
| $LDR$ | Load to Data Ratio (10) |
| $t_{l_n}$ | Total CPU Cycles including Overlay ($t_{l_n} = t_{s_n}/LDR$) |
| $\Delta p, \Delta e$ | Coef() unifying factor (1, 0.1) |
| **Control** | **Range** |
| $e_{w_i}$ | 50 - 100 mW |
| $\epsilon$ | 0.5 - 1 |
| $\delta$ | 2 - 4 |
| $d_{ij}$ | 5 - 30 m |

### 6.2.1 Validation

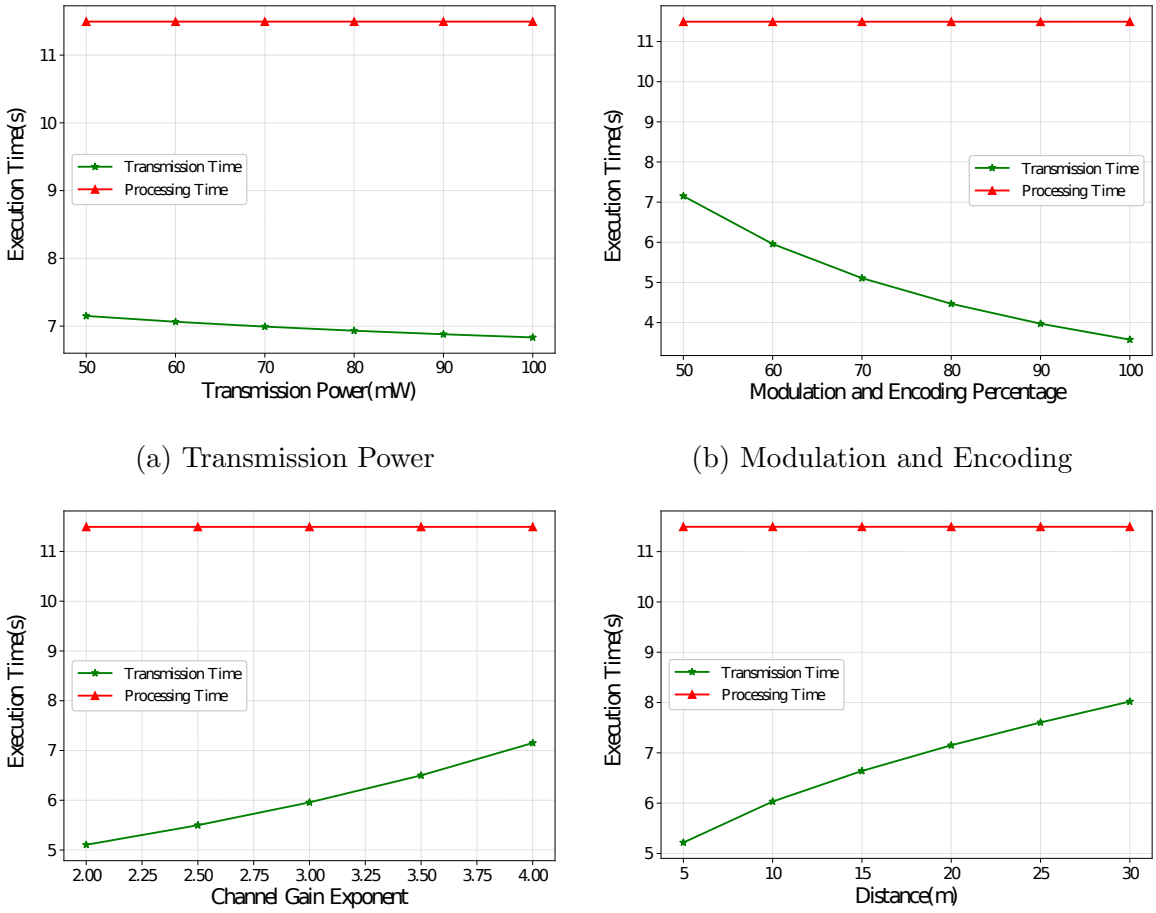
In this experiment set, the processing model and energy model is first validated, as shown in Figure 6.3 and Figure 6.4, specifically concerning the offloading data transmission cost and Cloudlet processing cost.

Figures 6.3a and 6.4a demonstrate the impact of transmission power on transmission time and energy cost. A trade-off can be observed between transmission speed and transmission energy reservation. Figures 6.3b and 6.4b present the gap between the real channel capacity and maximum channel capacity. The Modulation and encoding factor value 100 means the maximum channel capacity without influence of encoding and modulation. Figures 6.3c and 6.4c describe the influence of path loss factor (channel gain factor) from rural area (2) to urban area (4). Figures 6.3d and 6.4d illustrate the impact of communication distance. A similar performance tend can be observed on distance and channel gain since they all influence the power ratio during the transmission.

### 6.2.2 Intra-Cloudlet Task Allocation Evaluation

In this section, the focus is on the performance evaluation (execution efficiency and energy efficiency) in the scope of intra-Cloudlet offloading. It compares the performance with three selective search algorithm Min-Min, Min-Max and deadline-greedy, in terms of scheduling priority and allocation strategy. Min-Min orders the smallest task first with minimum resources. Min-Max also orders the smallest task first but with maximum resources. Deadline-greedy orders the task with tightest deadline first. Based on the priority order, it then traverses all possible resource allocation strategies and find the optimized amount of resources for each type of tasks. To fairly compare the performance of the proposed allocation model to other priority-based algorithms, it assumes that the updated task deadline value $t_{n_b}$ (knowledge regarding the contact time calculated according to the mobility patterns) is known among all the algorithms. Given the task set as shown in Table 6.3, it evaluates the total task execution time in the system, based on different priority strategies. If the task deadline cannot be met on the Cloudlet during the offloading processing, the tasks will be dropped. The dropped tasks are then processed by the UE computing power.

The results of the offloading efficiency based on task load change are shown in Figures 6.5 and 6.6 for execution efficiency and energy efficiency. As shown in Figure 6.5a and Figure 6.6a, the overall execution time and energy cost increase with the task load. All algorithms can adjust the offloading allocation strategy according to the load change and



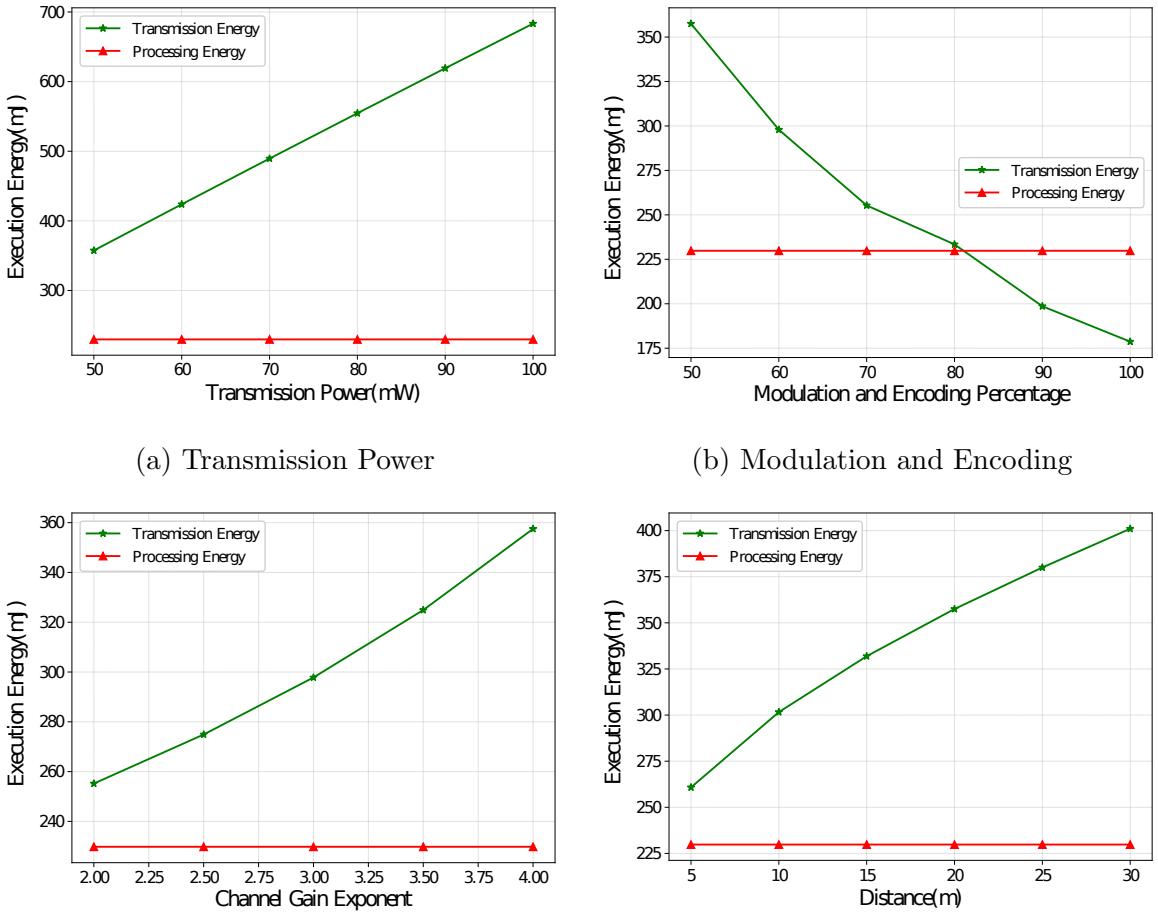

(a) Transmission Power

(b) Modulation and Encoding

(c) Channel Gain

(d) Distance

Figure 6.3: Offloading Processing Time and Transmission Time Validation




(a) Transmission Power

(b) Modulation and Encoding

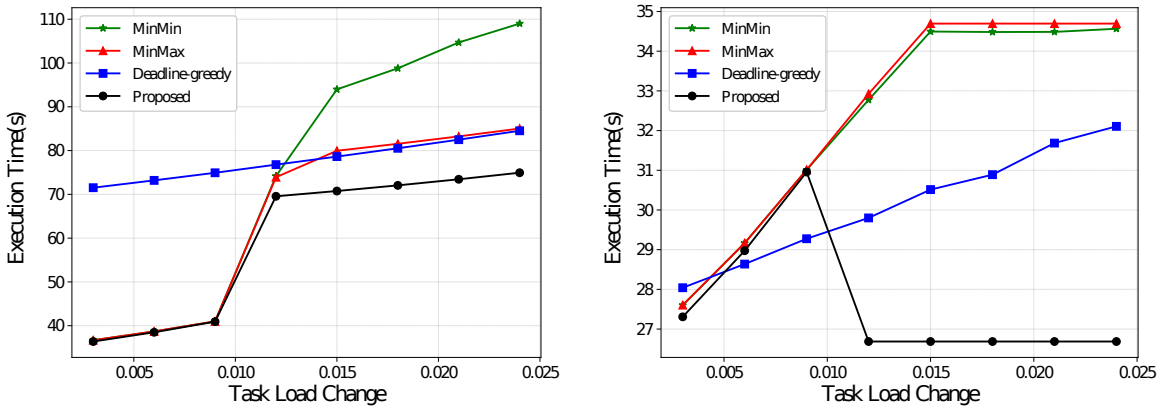
(c) Channel Gain

(d) Distance

Figure 6.4: Offloading Processing Energy and Transmission Energy Validation

Table 6.3: Experiment: Intra-Cloudlet Offloading Parameters

| **Parameter** | **Definition and Default Value** |
|---|---|
| $N$ | Number of Offloading Task (3) |
| $n_{cl}$ | Number of Total Computing Cores (6) |
| $w$ | Inertia Weight (0.9) |
| $r_1, r_2$ | Random Factor (0 to 1) |
| $c_1, c_2$ | Learn Factor (1.5) |
| $\gamma$ | Joint Cost Coefficient (0, 1) |
| $V_{max}$ | Maximum Speed (50) |
| $iter$ | rounds of iterations (30) |
| $ol_{t_1}$ | Overlay Factor of Task 1 (5.592) |
| $ol_{t_2}$ | Overlay Factor of Task 2 (10) |
| $ol_{t_3}$ | Overlay Factor of Task 3 (2.5) |
| $\mu_{t_n}$ | Task $n$ Processing Rate ($\mu_{t_n} = t_{l_n}/Cl_{CPU}$) |
| $t_{1_b}$ | Task Deadline of Task 1 (15 s) |
| $t_{2_b}$ | Task Deadline of Task 2 (20 s) |
| $t_{3_b}$ | Task Deadline of Task 3 (10 s) |
| $\lambda_{t_2}$ | Arrival Rate for Task 1 (0.012/s) |
| $\lambda_{t_1}$ | Arrival Rate for Task 2 (0.1/s) |
| $\lambda_{t_3}$ | Arrival Rate for Task 3 (0.04/s) |
| $\mu_l$ | UE proc rate ($\mu_l = t_l/ue_{CPU}$) |
| **Control** | **Range** |
| $\lambda_{t_2}$ | 0.003 - 0.024 /s |
| $t_{1_b}$ | 10 - 20 s |
| **Comparison** | **Description** |
| Min-Min | Minimum Task Orders First with Minimum Resources |
| Max-Min | Minimum Task Orders First with Maximum Resources |
| D-Greedy | Tightest Deadline Task Orders First with Optimized Resources |
| Proposed | Orders and Resources Allocated on the Proposed Model |



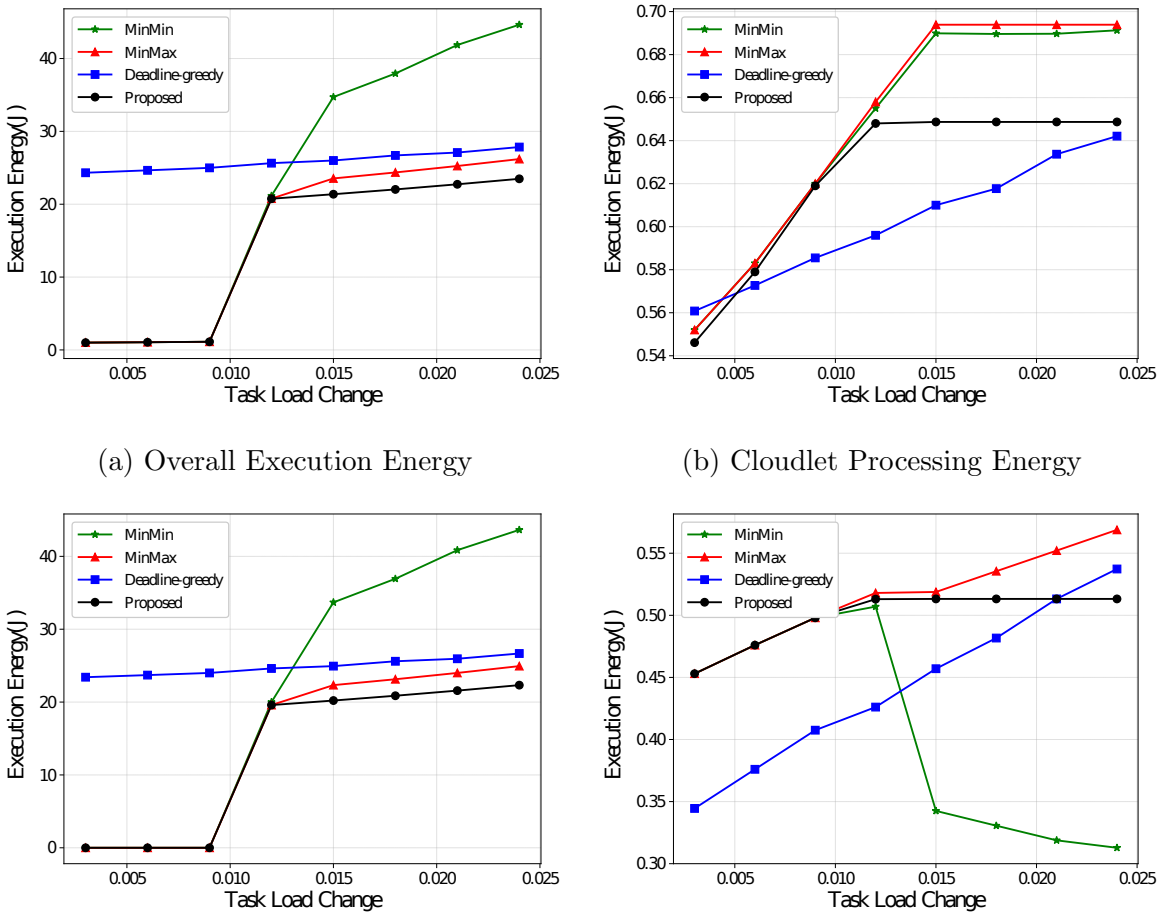

(a) Overall Execution Time

(b) Cloudlet Processing Time

(c) UE Processing Time

(d) Transmission Time

Figure 6.5: Execution Efficiency based on the Load Change of Task 2 ($\lambda_{t_2} = 0.003$ *to* $0.024$)



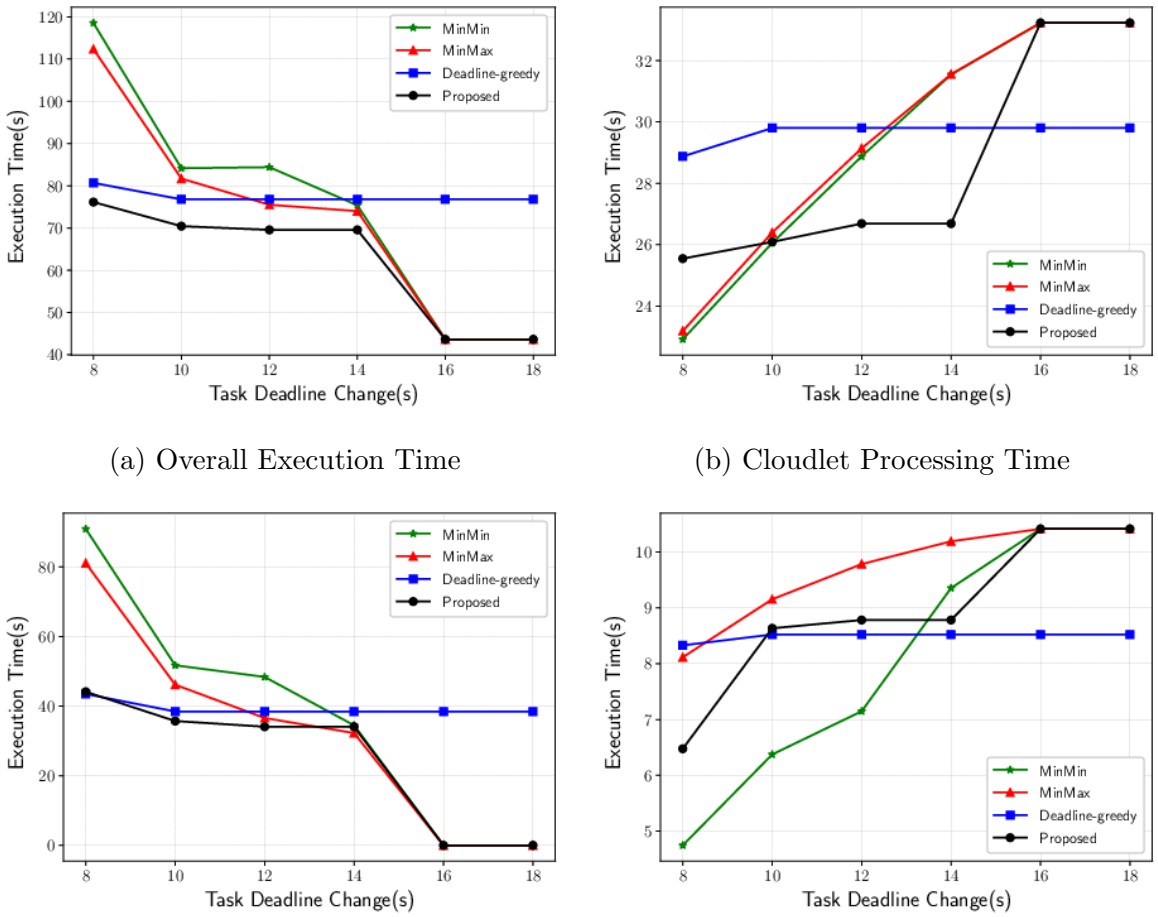

(a) Overall Execution Energy

(b) Cloudlet Processing Energy

(c) UE Processing Energy

(d) Transmission Energy

Figure 6.6: Energy Efficiency based on the Load Change of Task 2 ($\lambda_{t_2} = 0.003$ *to* 0.024)



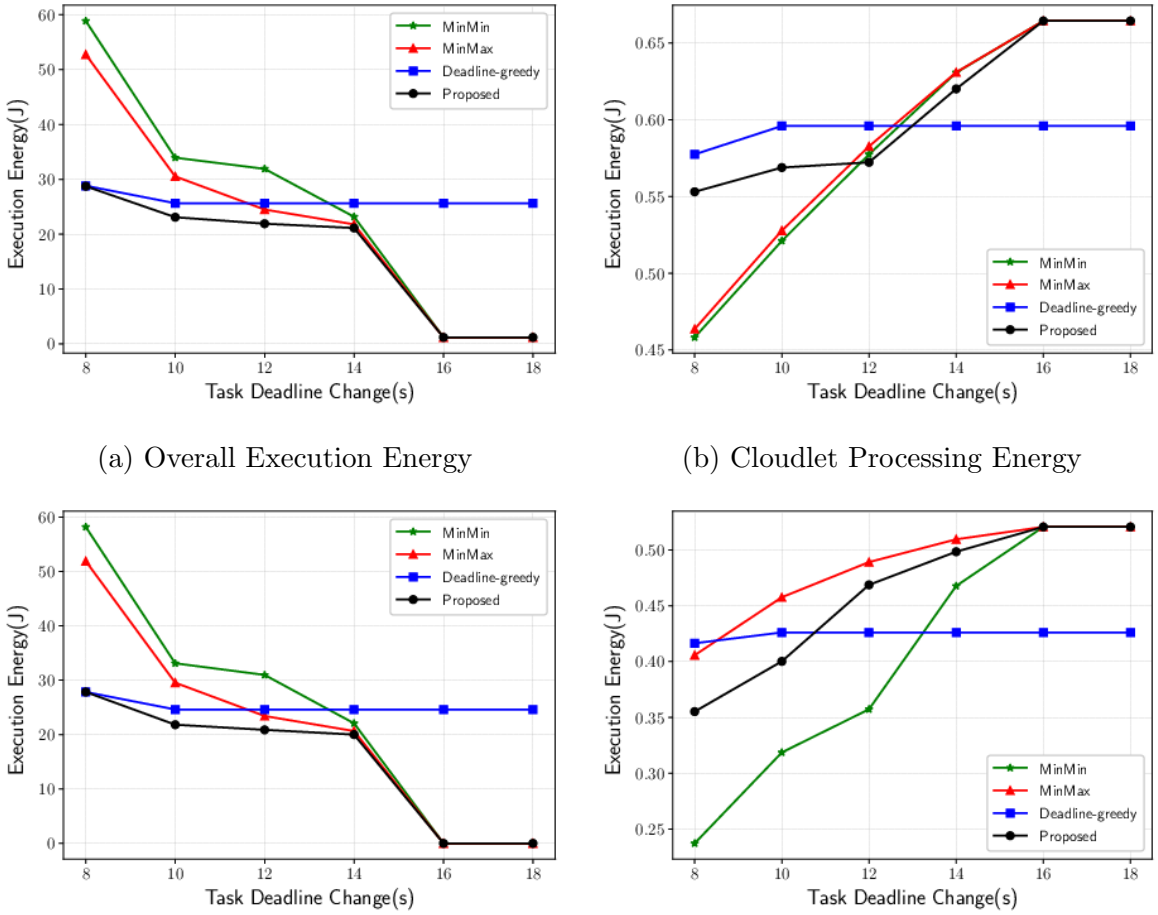

(a) Overall Execution Time

(b) Cloudlet Processing Time

(c) UE Processing Time

(d) Transmission Time

Figure 6.7: Execution Efficiency based on the Load Change of Task 1 ($t_{1_b} = 10$ *to* 20)




(a) Overall Execution Energy

(b) Cloudlet Processing Energy

(c) UE Processing Energy

(d) Transmission Energy

Figure 6.8: Energy Efficiency based on the Deadline Change of Task 1 ($t_{1_b} = 10$ *to* 20)

the proposed solution outperforms the others in execution efficiency and energy efficiency. The drastic increase around the value 0.012 indicates the dropping of offloading tasks occurred and the dropped tasks are then processed by the low end UE computing power (also in Figure 6.5c and Figure 6.6c). Figure 6.5b and Figure 6.6b demonstrate the offloading processing time on the Cloudlet and the energy cost when UE waits for the processing. As shown in the results (value 0.012 to 0.024) of Figure 6.5b, compared to the others, the proposed solution can better trade off the computation gain and communication overhead when congestion occurs in the Cloudlet processing queue. It intends to return tasks back to UE such that the load in the Cloudlet is largely reduced. Also, it still keeps the minimum overall execution time. Figure 6.5c and Figure 6.6c concerns the execution time and energy cost for the dropped tasks. These figures depict similar curves as in Figure 6.5a and Figure 6.6a, which indicates that the UE processing is the most time and energy consuming phase. Figure 6.5d and Figure 6.6d illustrate the transmission time and transmission energy cost for offloading tasks. The proposed solution requires around 50% extra time and energy for transmission when compared to Min-Min. This result indicates that, in the proposed solution, more tasks are able to be assigned to the high end Cloudlet via proper task priority rearranging and resource re-allocation. As a result, the proposed model can achieve the best overall performance in execution efficiency and energy efficiency, as depicted in Figure 6.5a and Figure 6.6a.

Besides the influence of load change observed in Figure 6.5 and 6.6, the impact of task deadline variation on offloading efficiency is concerned in Figure 6.7 and Figure 6.8, for energy consumption and execution time. Figures 6.7a and 6.8a demonstrate the overall offloading time and energy cost as with the task deadline changes. As the deadline becomes looser, all algorithms can achieve a better offloading performance. Similar to the results of load change, the proposed model also gets the best overall execution efficiency and energy efficiency, due to the better adjustment of the task deadline. Figures 6.7b and 6.8b present the UE offloading cost in the Cloudlet processing stage. As shown in the results, a higher utilization of the Cloudlet (deadline-greedy) server may not necessarily lead to better execution and energy efficiency, especially when the task deadline has already been tight. Concerning the communication and queueing overhead, balancing the load between UE and the Cloudlet server can not only reduce the execution time but increase energy reservation during offloading. Figures 6.7c and 6.8c show the time cost and energy cost on for the dropped offloading tasks by UE. When the task deadline value is higher than 16, the execution time and energy consumption on UE is 0 for the proposed model, which indicates that the Cloudlet can properly handle all tasks without

blocking. Figures 6.7d and 6.8d introduce the results related to transmission. As the task deadline becomes looser, more tasks can be offloaded to the Cloudlet, resulting in a higher transmission cost for all algorithms.

Based on the results in Figures 6.5, 6.6, 6.7 and 6.8, the conclusions can be summarized as follows. In the low task load and loose deadline offloading scenarios, maximizing the utilization of Cloudlet resources can achieve the best offloading execution and energy efficiency. In the high task load and task deadline scenarios, the communication and queuing overhead can largely reduce the overall performance of the Cloudlet. Proper scheduling of the task priority and appropriate allocation of limited Cloudlet resources can help to mitigate the congestion occurred on the Cloudlet and achieve an overall better offloading execution and energy efficiency.

### 6.2.3 Inter-Cloudlet Offloading Evaluation

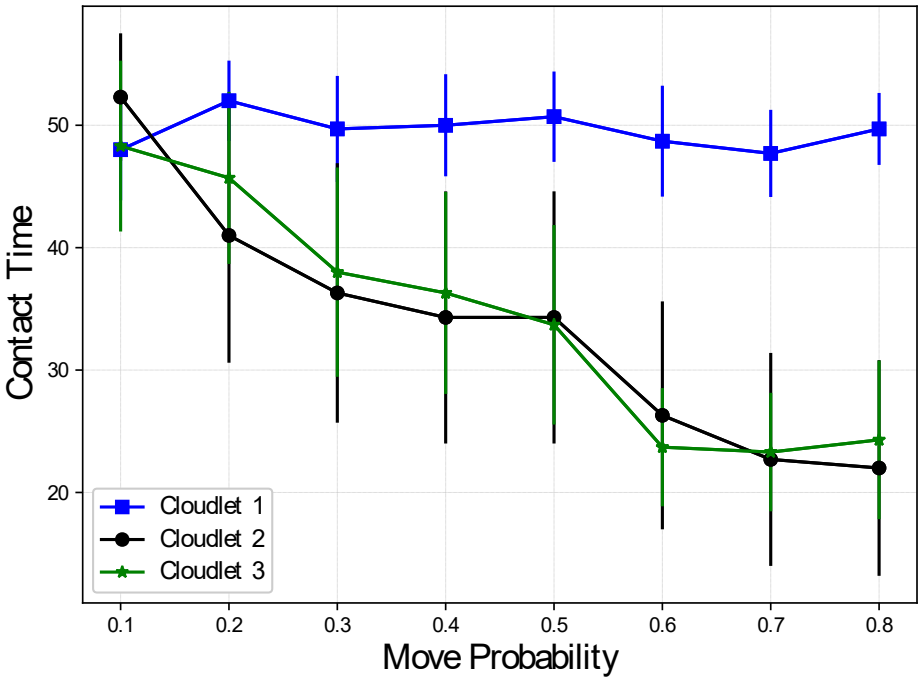
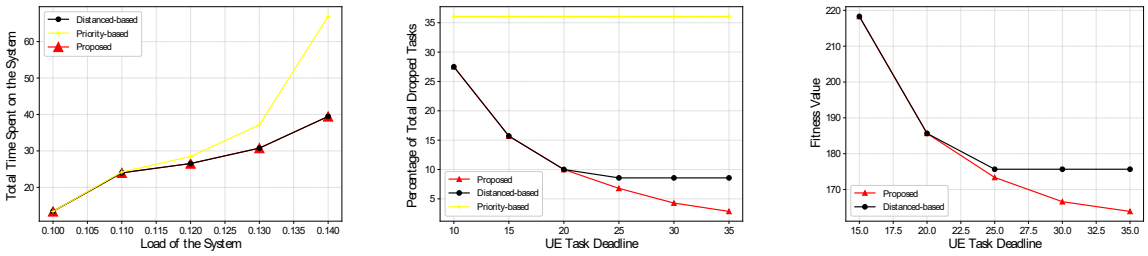



Figure 6.9: UE Contact Time based on Move Mobility Probability

In this section, the experiment evaluates the system performance under the inter-Cloudlet offloading scenarios where multiple Cloudlets are available to the UE. In order to observe the offloading performance, it first validates the UE (at position (0,0)) mobility

Table 6.4: Experiment: Inter-Cloudlet Offloading Parameters

| **Parameter** | **Definition and Default Value** |
|---|---|
| $(x_u, y_u)$ | UE Position (0,0) |
| $v_{mov}$ | Max Speed (1.5 m/s) |
| $f_V(v)$ | Preferred Speed and Probability ($f(0) = 0.1$ to 0.8) |
| | ($f(v_{mov}) = 0.8$ to 0.1) |
| $a_{max}$ | Max Acceleration (1 $m/s^2$) |
| $a_{min}$ | Min Acceleration (-1 $m/s^2$) |
| $f_D(a)$ | Direction Acceleration 0 |
| $f_D(t=0)$ | Default Direction 0 |
| $d^*$ | Final Direction (360 Degree) |
| $\mu_V, \mu_D$ | Mean Time Between Events (5 s) |
| $\Delta t$ | Curve Time (1 - 5 s) |
| $(x_{cl}, y_{cl})$ | Cloudlet Position $Cl_1$ (0,17.3) |
| | $Cl_2$ (-8.66,-5) |
| | $Cl_3$ (8.66,5) |
| $r$ | Coverage Radius (20 m) |
| $\mu_b$ | Background Task Process Rate (0.3 /s) |
| $t_{b_b}$ | Background Task Deadline (8 s) |
| $T_p$ | Transmission Time (0.5 s) |
| $\mu_U$ | UE Task Process Rate (0.3 /s) |
| $t_{U_b}$ | UE Task Deadline (10 to 35 s) |
| $T_c$ | Transmitting Time (0.5 s) |
| $\gamma$ | Joint Cost Coefficient 0.5 |
| $e_{p_n}$ | Processing Energy Cost (300 mW) |
| $e_{t_n}$ | Transmission Energy Cost (30 mW) |
| $ew_n$ | Idle Energy Cost (20 mW) |
| **Control** | **Range** |
| $\lambda_b$ | Average Background Load 0.1 to 0.14 /s |
| $\lambda_U$ | UE Task Load 0.1 to 0.14 /s |
| **Comparison** | **Description** |
| Distance-based | Select Cl by Min Distance |
| Priority-based | Select Cl by Highest Priority |
| Proposed | Select Cl by Min Fitness Value |




(a) Total Offloading Time based on Task Load Change ($t_{U_b} = 10$; $\lambda_b, \lambda_U = 0.1$ *to* 0.14)

(b) Total Task Block Ratio based on UE Task Deadline Change ($\lambda_b, \lambda_U = 0.14$; $t_{U_b} = 10$ *to* 35)

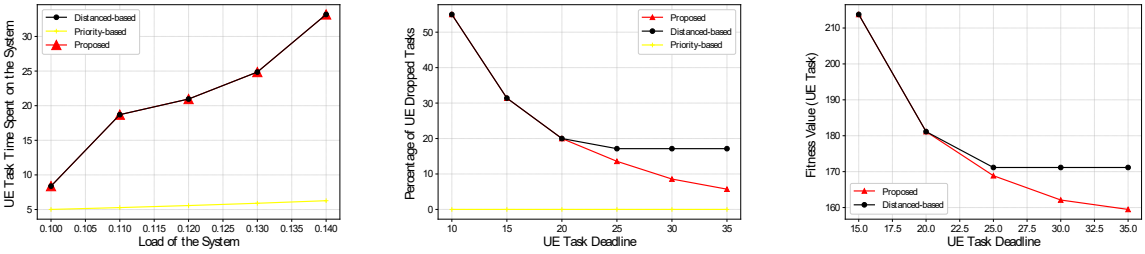
(c) Total Offloading Fitness Value based on UE Task Deadline Change ($\lambda_b, \lambda_U = 0.14$; $t_{U_b} = 15$ *to* 35)

Figure 6.10: Inter-Cloudlet Offloading – System Level (UE Task + Background Task) Total Execution Time, Number of Dropped Tasks and Fitness Values ($t_{b_b} = 8$; $\mu_b, \mu_U = 0.3$)

trace which is covered by three Cloudlets ($Cl_1$ (0,17.3), $Cl_2$ (-8.66,-5), $Cl_3$ (8.66,5)). Based on the trace, it calculates the contact time for each of them accordingly. Then, our proposed inter-Cloudlet offloading algorithm is compared to the distance-based algorithm and priority-based algorithm in terms of execution cost and energy cost. The distance-based algorithm selects the Cloudlet with the minimum distance, while the priority-based algorithm selects the Cloudlet that gives the UE offloading task the highest priority. In order to fairly compare the performance, it assumes that each Cloudlet contains identical resources, coverage radius, background loads, bandwidth units, and UE energy cost factors. The $Cl_3$ is set as the Cloudlet that guarantees new arrival tasks the highest execution priority.

Based on the Smooth mobility model, the contact time of the UE is shown in Figure 6.9. The Cloudlet $Cl_1$ obtains the longest contact time because it covers the most traces generated by the UE, given the UE initial direction and end direction. Compared to $Cl_1$, $Cl_2$ and $Cl_3$ are more sensitive to the movement of the UE such that the contact time of $Cl_2$ and $Cl_3$ decreases drastically with the increasing of move probability. It is worth mentioning that the contact time of $Cl_2$ on 0.1 move probability is larger than $Cl_1$. This is because the default the distance between $Cl_2$ to UE is less than that of $Cl_1$, which dominates the contact time in the low mobility scenarios.

In the comparison of the inter-Cloudlet offloading experiment, the experiment adopts the contact time based on the worst case scenario from Figure 6.9 where the UE move




(a) UE Task Offloading Time based on Task Load Change ($t_{U_b} = 10$; $\lambda_b, \lambda_U = 0.1$ *to* $0.14$)

(b) UE Task Block Ratio based on UE Task Deadline Change ($\lambda_b, \lambda_U = 0.14$; $t_{U_b} = 10$ *to* $35$)

(c) UE Task Offloading Fitness Value based on UE Task Deadline Change ($\lambda_b, \lambda_U = 0.14$; $t_{U_b} = 15$ *to* $35$)

Figure 6.11: Inter-Cloudlet Offloading – UE Task Execution Time, Number of Dropped Tasks and Fitness Values ($t_{b_b} = 8$; $\mu_b, \mu_U = 0.3$)

mobility equals 0.8. In addition, it introduces the same background load with identical processing time and task deadline on the three Cloudlets for fair comparison, as shown in Table 6.4.

The evaluation results are shown in Figure 6.10 and 6.11 in terms of offloading time (Figures 6.10a and 6.11a), service denial ratio (Figures 6.10b and 6.11b) and offloading fitness value (Figures 6.10c and 6.11c) respectively.

In Figure 6.10a, the total offloading time increases as with the growing of the task load. The proposed algorithm and the distance-based algorithm achieve the same performance because they have the same updated task deadline, as in Formula 6.36 – the original UE task deadline is tighter than the UE contact time. The proposed algorithm and the distance-based algorithm outperform the priority-based counterpart due to the awareness of task load. This is because the load-awareness enables the Cloudlet to partially deny the UE tasks such that the overall task execution time can be reduced. The corresponding UE task offloading time is depicted in Figure 6.11a. It can be observed that although the priority-based algorithm can process the UE tasks more efficiently than the proposed algorithm, the priority-based algorithm requires 45.3% additional overall execution time for all offloading tasks on the system level than the proposed one, as shown in Figure 6.10a.

Figures 6.10b and 6.11b demonstrate the denial of service ratio based on the variation of task deadlines on the overall system level and on the UE task, respectively. The priority-based scheduling achieves a constant performance, because it always schedules

UE tasks ahead of the background tasks without the consideration of UE contact time and task deadline change, leading to the highest task drop ratio on the system level. The proposed scheduling model and the distance-based scheduling model achieve the same performance at the UE task deadline points 10, 15 and 20 because the original UE task deadline has a larger impact on the UE contact time. However, when the original task deadline value is larger than the UE contact time (at UE task deadline 25, 30, 35), the additional UE trace information can further help the proposed algorithm to select the best Cloudlet such that the task drop rate is minimized. When the task deadline reaches 35, the proposed scheduling model can reduce the task drop rate to 92.07% and 66.7% separately, compared to the priority-based model and distance-based model.

Figures 6.10c and 6.11c show the fitness value changes in the same environment as Figures 6.10b and 6.11b. The fitness value is calculated from Formula 6.42 as the sum of offloading execution time and UE energy cost. Similar to the task drop rate, the fitness value of priority-based scheduling results in a constant value 828.53 for the system level fitness. This high fitness value is obtained because all dropping tasks are processed on the UE side, leading to high processing energy costs on the UE. Also, the proposed model achieves better fitness value than the distance-based model after the task deadline reaches 20 because fewer tasks are dropped.

Based on the results of the second experiment set, it can be observed that the proposed scheduling model can achieve the best overall offloading performance in terms of offloading time, task drop rate and mixed cost among all the tasks in the system by coordinating background tasks with UE mobility information.

## 6.3 Conclusion

In this chapter, a MEC-based mobility-aware energy-efficient offloading prototype is proposed. The system is first discussed in terms of the offloading use case with a description of the core components. Then, the multi-user intra-Cloudlet offloading process is modeled as the $M/G/1$ queue and the offloading scheduling problem are formalized. In addition, the multi-Cloudlet resource selection problem is discussed in the context of system-level energy cost and time cost. According to the experimental results, it can observed that the trade-off between offloading computation gain and communication overhead should be carefully balanced. The transmission cost may exceed the task execution cost due to the system load and UE-related mobility and task patterns. Also, it can be witnessed that the queueing overhead can drastically impact the task execution

efficiency. A proper task blocking strategy can not only reduce the load of the system but achieve smaller overall offloading time.

# Chapter 7

# Conclusion and Future Work

In this chapter, the summary of the entire thesis is first presented. Then, the future work and research direction is briefly discussed.

## 7.1 Conclusion

The research efforts produced in the work of this thesis consist in a Cloudlet-based hybrid offloading model in the MCC-based environment, as discussed previously. In this chapter, a brief summary is presented to my thesis.

- **A survey and an overview of the state-of-the-art on energy-awareness in the MCC-based offloading and related Green Cloud computing strategies**. The state-of-the-art of works related to energy-aware is extensively surveyed in the scope of MCC-based offloading. The origin of MCC is illustrated based on the nature of Mobile Computing and the comparison between Grid Computing and Cloud Computing. The offloading process, which consists of partitioning of tasks, profiling, decision-making, and offloading components, has been extensively explored in several works, considering the local devices, Cloudlets, and the remote Cloud. The networking aspects, as a determinant factor in performance, have also been investigated. All these elements have been broadly described and discussed in the survey study, guided through the functionality, issues, and solutions of the schemes, architectures, and techniques. This work gathered and examined the existing studies, systematically classifying according to the perspective of the mobile device and the Cloud, observing performance aspects in terms of the energy-aware

offloading processing and the trade-off between energy reservation and execution efficiency.

- **A Cloudlet-based multi-layered prototype for offloading in the MCC environment**. A layered cloud platform was proposed for the distributed simulations. Concerning usability, energy consumption, security, reliability, and elasticity, relevant components are implemented to support fast deployment, to ease the management of underlying resources, to reduce energy usage, and to enable fine-grained resource handling during runtime. The design of a multi-layered scheme in different scenarios was described. In the experiments, the performance of this cloud simulation platform was evaluated and discussed in detail. Based on the experimental results, it concludes that the use of cloud technologies is a promising method for facilitating distributed simulations, especially when the network environment presents greater efforts towards optimization and performance.

- **A task-centric mobile Cloud-based system to enable energy-aware fine-grained offloading**. The system is first presented with analysis and discussion of the local mobile devices, Cloudlets, remote Cloud and networking, regarding the roles and core functions. The offloading scenario is formulated with the task module, computing module, and network module respectively and a collaborative offloading scheduling algorithm is proposed. Compared to the device-based solutions, the proposed task-centric Cloudlet-based system can achieve fine-grain dynamic resource control in the Cloudlet and remote Cloud. In addition, the leverage of global execution knowledge also benefits the offloading decision-making. Based on the experiments, the proposed system formulation is proved to be valid in the real data; the scheduling protocol is evaluated in terms of execution efficiency, energy efficiency on the task level and the Cloudlet level.

- **An efficient and scalable proactive offloading model for sustainable Cloudlet resource allocation in heterogeneous mobile Cloud environments**. A Cloudlet-based hybrid heterogeneous offloading model is proposed. The system is first discussed in terms of the core functioning components. The offloading behaviours are then analyzed based on the queueing model, and the energy-aware offloading allocation problem is formalized and solved by the PSO heuristic. A time series-based prediction model is integrated to achieve proactive offloading. The enabling of proposed heterogeneous offloading and fine-grain prediction bene-

fits the overall energy-aware time-constraint offloading decision process. Based on the the experiment results, the proposed system outperforms its counterparts on execution and energy efficiency.

- **A mobility-aware energy-efficient model for MEC-based offloading**. The system is first discussed in terms of the offloading use case with a description of the core components. Then, the multi-user intra-Cloudlet offloading process is modeled as the $M/G/1$ queue and the offloading scheduling problem are formalized. In addition, the multi-Cloudlet resource selection problem is discussed in the context of system-level energy cost and time cost. According to the experimental results, the proposed model can achieve intra-Cloudlet and inter-Cloudlet offloading execution efficiency and energy efficiency, compared to its counterparts.

## 7.2 Future Work: AI-based Smart Edge

Provisioning diverse types of services timely for multi-tenants is non-trivial, especially in the large scale scenarios where the number of resources is relatively inadequate compared to the users. Many AI-based efforts have been made in the EC academia and other domains, which are demonstrated promising performance. Herein, in the future work, the plan is to first categorize these AI-based protocols based on the EC hierarchy and EC technology and concern these AI-based research topics on different scales. Then, the next step is to extend the current research to cope with these opportunities and potential issues on adapting AI-based solutions in the smart EC environment.

### 7.2.1 UE Layer

The UEs are edge service initiators which necessitate the service partitioning, edge platform selection, and incentive to participate.

#### Service Partitioning

The service partitioning concerns the processing logic and task load, which is typically model by directed acyclic graph (DAG). The vertex in DAG represents one sub-task in the service request processed on one resource while the values indicate the processing cost on that corresponding resource. The edges and arrows among the vertices illustrate the

dependency of sub-tasks and execution logic, in which the values are commonly utilized to describe the communication cost between the dependent sub-tasks.

#### Network Selection

To establish the connection between UE and EC platform, the UEs first decide to initialize the communication channel selection. Due to the nature of mobility, UEs usually maintain these connections intermittently, which complicates the network selection problem since service provisioning may be interrupted by possible communication failures. Besides, the network condition changes dynamically, while network context information may be only partially available to UEs. In this case, queueing models are generally used to translate the uncertainty of the environment to probabilities.

#### Incentive and Pricing

Due to the time-sensitivity and privacy concerns, resource sharing among UEs remains discouraged. Incentivizing UEs to participate in service provisioning and utilization is crucial to the success of edge technologies, involving the pricing schemes, negotiation, and argumentation protocols. The cost of resources is represented by a tuple of parameters that identify the resource availability and capacity. For example, network resources utilize binary indicators for the medium availability while computation resources adopt the CPU speed and utilization ratio to estimate the processing capacity.

### 7.2.2 Edge Platform Layer

The core edge platform layer concerns the initial edge resource placement, centralized UE prediction, data-related caching and profiling, and intra-edge resource management.

#### Placement

The edge server placement draws increasing attention in the context of smart cities since it is the essential enabler in terms of the service provisioning performance, concerning both the UE's availability and the server's utilization efficiency.

#### Load Prediction

Profiling the load in the system is significant for proper scheduling decisions. Concerning the mobility nature of UEs, and high variance of service requests, load prediction

on the edge platform prevails to be challenging. The traditional regression-based methods, such as linear regression and autoregressive integrated moving average, focus on the evaluation of the time, space, and load relations. However, these schemes cannot extract essential features to lean from mobility, task, and user patterns in the non-stationary non-periodic context. In addition, the impact of the current decision on future time intervals cannot be fully considered.

#### Caching

Caching policies are widely utilized in wired networks for content delivery, sorting and storing services typically according to the frequency and recency in a relatively stable environment. However, the dynamic edge networks complex the caching context, involving network topology changes, device mobility, interference, fading wireless channels. These impact factors sophisticate the modeling process of such an execution environment and the context-awareness of edge service provisioning.

#### Resource Allocation

Allocating services is one of the core function on the edge platform. Many centralized methods have been proposed in the literature for batch scheduling from CC and MCC. However, the context of EC is complicated, as discussed in the previous subsection, where profiling and metadata gathering may achieve only partial awareness of the environment. The potentially significant profiling and scheduling delay in centralized allocation protocols, in turn, impede the utilization of the profiling data, because the environment is changing dynamically and quickly. In this case, decentralized resource management schemes are preferred where UEs can perform as rational and intelligent agents to schedule services for themselves.

### 7.2.3 Inter-edge Layer

Besides the intra-edge orchestration functions, the edge platform is also required to coordinate with other edge servers located in distributed geographical locations and Cloud resources, which include the multi-site service migration management and overall failover solutions.

**Multi-site Service Migration**

Service migration is enabled via virtualization technologies, which facilitates virtual instances to migrate among hosts or containers lively. Live migration is widely utilized for load balancing in the context of CC, and the migration protocols are adapted in many works to tackle seamless service provisioning and handover issues in the edge environment, where UEs may have high mobility and pass through several BS. In these works, the mobility pattern of UE is first predicted, based on which the cost and benefit of migration to each BS in the vicinity are estimated. According to the overall gain of the migration, the destination BS is selected heuristically.

**Failover**

Due to the possible error on edge servers, failures may occasionally occur during edge-based service provisioning. Detecting such failures timely and recovering with minimal overhead can significantly mitigate the effects of system errors.

This is almost the end of the thesis. Thank you again for your time to read my thesis.

## 7.3 Current Works to Be Extended

In this section, the studies of my research group can be potentially extended by my thesis are discussed.

### 7.3.1 VANET

The VANET-based studies concerns the offloading cases specifically on the road network edge [75], [65] and [225]. For the dissemination, the works [213] and [198] target the video dissemination issues by relay multiple nodes and safty models. The work [212] uses unicast for video streaming. For the localization, the work [74] adapts periodic messages to reduce the overhead. The work [4] tries to achieve context-awareness by location-based service discovery method. The work [5] targets the nonline-of-sight issue by location-based verification. For the overall VANET architecture, the work [102] proposes a hierarchical software-defined structure while the work [257] utilizes multi-hop clustering. In addition, the work [253] and [252] concern the traffic congestion and traffic light issues in the context of VANET. All the aforementioned works can further improve their execution efficiency by adapting the proposed 3-tier edge offloading scheme.

### 7.3.2 Underwater Sensor Networks

The underwater sensor networks [104], [107], [109], [106] present a specific edge offloading scenario where UEs are moving deeply in the sea, targeting issues such as node recovery [105] and depth adjustment [103]. If the ship can provide Cloudlet-like services, the proposed model in the thesis can also adapted in this area.

### 7.3.3 Ad Hoc Networks

Besides the works in the context of VNET and under water, many works are also proposed in the general Ad Hoc networks for algorithm design [37], [184]; modeling [36]; and performance analysis [35], [42]. Location is one of the core information to be identified in wireless sensor networks [63]. The works [70], [74] and [66] utilize periodic messages between neighbours to locate the UEs. The work [52] and [196] adapt voronoi model for localization. The work [111] using recursion protocols to target the mobile devices. Routing is the most important function in the ad hoc networks [38], [76], [258], [82]. Many studies have been proposed targeting different aspects, such as data propagation [61], data collection [203], mobility-awareness [12], performance analysis [121], [41], architecture design [67], [84], congestion control [44], and mining [79]. Concerning the security, several works [50], [34], [49], [83], [71], [210], [48], [72] , [64], [73], [62], [58] try to enhance the network security via trust-based solutions such as reputation and anonymity. For the reliability, the works [68], [197], [55], [59], [120], [57], [51], [122], [40], [211], [69], [56], [216], [202], [54], [39] adapt different monitoring, synchronization, and fault tolerant strategies to achieve fast error detection and recovery. However, concerning the scale of works, most of them are based only on simulations. By adapting the proposed offloading model, these experimental models can be implemented in the real world scenarios.

### 7.3.4 Edge-based Simulations

Concerning the scale of the experiments, simulations are widely utilized in the context of edge-based offloading. The works [119], [60], [46] and [45] propose grid-based architecture for the simulation workbench while [110] and [43] further tackle the load balancing issues. Some other works tackles the parallel processing issues in the simulations, such as [18], [81], [80], [77], [78], [47]. By implementing the proposed edge offloading model in this thesis, the aforementioned simulators can be further validated based on more complex and real execution scenarios.

# Bibliography


[1] Mohammad Aazam, Sherali Zeadally, and Khaled A. Harras. Offloading in fog computing for iot: Review, enabling technologies, and research opportunities. *Future Generation Computer Systems*, 2018.

[2] Saeid Abolfazli, Zohreh Sanaei, Ejaz Ahmed, Abdullah Gani, and Rajkumar Buyya. Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges. *IEEE Communications Surveys & Tutorials*, 16(1):337–368, 2014.

[3] Saeid Abolfazli, Zohreh Sanaei, Mojtaba Alizadeh, Abdullah Gani, and Feng Xia. An experimental analysis on cloud-based mobile augmentation in mobile cloud computing. *IEEE Transactions on Consumer Electronics*, 60(1):146–154, 2014.

[4] Kaouther Abrougui, Azzedine Boukerche, and Richard Werner Nelem Pazzi. Design and evaluation of context-aware and location-based service discovery protocols for vehicular networks. *IEEE Transactions on Intelligent Transportation Systems*, 12(3):717–735, 2011.

[5] Osama Abumansoor and Azzedine Boukerche. A secure cooperative approach for nonline-of-sight location verification in vanet. *IEEE Transactions on Vehicular Technology*, 61(1):275–285, 2011.

[6] Ejaz Ahmed, Abdullah Gani, Muhammad Khurram Khan, Rajkumar Buyya, and Samee U Khan. Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges. *Journal of Network and Computer Applications*, 52:154–172, 2015.

[7] Yuan Ai, Mugen Peng, and Kecheng Zhang. Edge computing technologies for internet of things: a primer. *Digital Communications and Networks*, 4(2):77–86, 2018.

[8] Osama Alrajeh, Matthew Forshaw, Andrew Stephen McGough, and Nigel Thomas. Simulation of virtual machine live migration in high throughput computing environments. In *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–8. IEEE, 2018.

[9] EC Amazon. Amazon elastic compute cloud (amazon ec2). *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[11] E. Baccarelli, N. Cordeschi, A. Mei, M. Panella, M. Shojafar, and J. Stefa. Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: review, challenges, and a case study. *IEEE Network*, 30(2):54–61, 2016.

[12] Athanasios Bamis, Azzedine Boukerche, Ioannis Chatzigiannakis, and Sotiris Nikoletseas. A mobility aware protocol synthesis for efficient routing in ad hoc mobile networks. *Computer Networks*, 52(1):130–154, 2008.

[13] Sergio Barbarossa, Stefania Sardellitti, and Paolo Di Lorenzo. Joint allocation of computation and communication resources in multiuser mobile cloud computing. In *2013 IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 26–30. IEEE, 2013.

[14] M. V. Barbera, S. Kosta, A. Mei, V. C. Perta, and J. Stefa. Mobile offloading in the wild: Findings and lessons learned through a real-life experiment with a new cloud-aware system. In *Proceedings of the IEEE Conference on Computer Communications INFOCOM*, pages 2355–2363, 2014.

[15] Marco V Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *INFOCOM, 2013 Proceedings IEEE*, pages 1285–1293. IEEE, 2013.

[16] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 164–177. ACM, 2003.

[17] Ken Barr, Prashanth Bungale, Stephen Deasy, Viktor Gyuris, Perry Hung, Craig Newell, Harvey Tuch, and Bruno Zoppis. The vmware mobile virtualization platform: is that a hypervisor in your pocket? *ACM SIGOPS Operating Systems Review*, 44(4):124–135, 2010.

[18] Rodolfo Bezerra Batista, Azzedine Boukerche, and Alba Cristina Magalhaes Alves de Melo. A parallel strategy for biological sequence alignment in restricted memory space. *Journal of Parallel and Distributed Computing*, 68(4):548–561, 2008.

[19] Mohamed Ben Belgacem and Bastien Chopard. A hybrid hpc/cloud distributed infrastructure: Coupling ec2 cloud resources with hpc clusters to run large tightly coupled multiscale applications. *Future Generation Computer Systems*, 42:11–21, 2015.

[20] Paolo Bellavista, Antonio Corradi, Luca Foschini, and Alessandro Pernafini. Data distribution service (dds): A performance comparison of opensplice and rti implementations. In *2013 IEEE symposium on computers and communications (ISCC)*, pages 000377–000383. IEEE, 2013.

[21] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[22] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing*, pages 826–831. IEEE Computer Society, 2010.

[23] Anton Beloglazov and Rajkumar Buyya. Openstack neat: a framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds. *Concurrency and Computation: Practice and Experience*, 27(5):1310–1333, 2015.

[24] Muli Ben-Yehuda, Michael D Day, Zvi Dubitzky, Michael Factor, Nadav Har'El, Abel Gordon, Anthony Liguori, Orit Wasserman, and Ben-Ami Yassour. The turtles project: Design and implementation of nested virtualization. In *Osdi*, volume 10, pages 423–436, 2010.

[25] Christian Bettstetter. Smooth is better than sharp: a random mobility model for simulation of wireless networks. In *Proceedings of the 4th ACM international*

*workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 19–27. ACM, 2001.

[26] Upendra Bhoi and Purvi N Ramanuj. Enhanced max-min task scheduling algorithm in cloud computing. *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*, pages 259–264, 2013.

[27] Andrew D Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems (TOCS)*, 2(1):39–59, 1984.

[28] Luiz F Bittencourt, Javier Diaz-Montes, Rajkumar Buyya, Omer F Rana, and Manish Parashar. Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing*, 4(2):26–35, 2017.

[29] Gunter Bolch, Stefan Greiner, Hermann De Meer, and Kishor S Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[30] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[31] Vinicius CM Borges, JS Dias, Anubis GM Rossetto, and Mario Antonio Ribeiro Dantas. Summit an architecture for mobile devices to coordinate the execution of applications in grid environments. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2007. WETICE 2007. 16th IEEE International Workshops on*, pages 217–222. IEEE, 2007.

[32] Dan Bornstein. Dalvik vm internals. In *Google I/O developer conference*, volume 23, pages 17–30, 2008.

[33] Dejene Boru, Dzmitry Kliazovich, Fabrizio Granelli, Pascal Bouvry, and Albert Y Zomaya. Energy-efficient data replication in cloud computing datacenters. *Cluster Computing*, 18(1):385–402, 2015.

[34] A Boukerch, Li Xu, and Khalil El-Khatib. Trust-based security for wireless ad hoc and sensor networks. *Computer Communications*, 30(11-12):2413–2427, 2007.

[35] Azzedine Boukerche. Performance comparison and analysis of ad hoc routing algorithms. In *Conference Proceedings of the 2001 IEEE International Performance,*

*Computing, and Communications Conference (Cat. No. 01CH37210)*, pages 171–178. IEEE, 2001.

[36] Azzedine Boukerche. *Handbook of algorithms for wireless networking and mobile computing*. CRC Press, 2005.

[37] Azzedine Boukerche. *Algorithms and protocols for wireless and mobile ad hoc networks*, volume 77. John Wiley & Sons, 2008.

[38] Azzedine Boukerche and Kaouther Abrougui. An efficient leader election protocol for wireless quasi-static mesh networks: Proof of correctness. In *2007 IEEE International Conference on Communications*, pages 3491–3496. IEEE, 2007.

[39] Azzedine Boukerche, Ioannis Chatzigiannakis, and Sotiris Nikoletseas. A new energy efficient and fault-tolerant protocol for data propagation in smart dust networks using varying transmission range. *Computer communications*, 29(4):477–489, 2006.

[40] Azzedine Boukerche, Xiuzhen Cheng, and Joseph Linus. Energy-aware data-centric routing in microsensor networks. In *Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 42–49, 2003.

[41] Azzedine Boukerche, Xuzhen Cheng, and Joseph Linus. A performance evaluation of a novel energy-aware data-centric routing algorithm in wireless sensor networks. *Wireless Networks*, 11(5):619–635, 2005.

[42] Azzedine Boukerche and Amir Darehshoorzadeh. Opportunistic routing in wireless networks: Models, algorithms, and classifications. *ACM Computing Surveys (CSUR)*, 47(2):1–36, 2014.

[43] Azzedine Boukerche and Sajal K Das. Dynamic load balancing strategies for conservative parallel simulations. In *Proceedings 11th Workshop on Parallel and Distributed Simulation*, pages 20–28. IEEE, 1997.

[44] Azzedine Boukerche, Sajal K Das, and Alessandro Fabbri. Analysis of a randomized congestion control scheme with dsdv routing in ad hoc wireless networks. *Journal of Parallel and Distributed Computing*, 61(7):967–995, 2001.

[45] Azzedine Boukerche, Sajal K Das, and Alessandro Fabbri. Swimnet: a scalable parallel simulation testbed for wireless and mobile networks. *Wireless Networks*, 7(5):467–486, 2001.

[46] Azzedine Boukerche, Sajal K Das, Alessandro Fabbri, and Oktay Yildiz. Exploiting model independence for parallel pcs network simulation. In *Proceedings Thirteenth Workshop on Parallel and Distributed Simulation. PADS 99.(Cat. No. PR00155)*, pages 166–173. IEEE, 1999.

[47] Azzedine Boukerche and Caron Dzermajko. Performance evaluation of data distribution management strategies. *Concurrency and Computation: Practice and Experience*, 16(15):1545–1573, 2004.

[48] Azzedine Boukerche, Khalil El-Khatib, Li Xu, and Larry Korba. A novel solution for achieving anonymity in wireless ad hoc networks. In *Proceedings of the 1st ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 30–38, 2004.

[49] Azzedine Boukerche, Khalil El-Khatib, Li Xu, and Larry Korba. Sdar: a secure distributed anonymous routing protocol for wireless and mobile ad hoc networks. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 618–624. IEEE, 2004.

[50] Azzedine Boukerche, Khalil El-Khatib, Li Xu, and Larry Korba. An efficient secure distributed anonymous routing protocol for mobile and wireless ad hoc networks. *computer communications*, 28(10):1193–1203, 2005.

[51] Azzedine Boukerche and Xin Fei. A coverage-preserving scheme for wireless sensor network with irregular sensing range. *Ad hoc networks*, 5(8):1303–1316, 2007.

[52] Azzedine Boukerche and Xin Fei. A voronoi approach for coverage protocols in wireless sensor networks. In *IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference*, pages 5190–5194. IEEE, 2007.

[53] Azzedine Boukerche, Shichao Guan, and Robson Eduardo De Grande. A task-centric mobile cloud-based system to enable energy-aware efficient offloading. *IEEE Transactions on Sustainable Computing*, 2018.

[54] Azzedine Boukerche, Sungbum Hong, and Tom Jacob. A distributed algorithm for dynamic channel allocation. *Mobile Networks and Applications*, 7(2):115–126, 2002.

[55] Azzedine Boukerche, Sungbum Hong, and Tom Jacob. An efficient synchronization scheme of multimedia streams in wireless and mobile systems. *IEEE transactions on Parallel and Distributed Systems*, 13(9):911–923, 2002.

[56] Azzedine Boukerche, Kathia Regina Lemos Jucá, João Bosco Sobral, and Mirela Sechi Moretti Annoni Notare. An artificial immune based intrusion detection model for computer and telecommunication systems. *Parallel Computing*, 30(5-6):629–646, 2004.

[57] Azzedine Boukerche and Xu Li. An agent-based trust and reputation management scheme for wireless sensor networks. In *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, volume 3, pages 5–pp. IEEE, 2005.

[58] Azzedine Boukerche, Renato B Machado, Kathia RL Jucá, João Bosco M Sobral, and Mirela SMA Notare. An agent based and biological inspired real-time intrusion detection and security model for computer network operations. *Computer Communications*, 30(13):2649–2660, 2007.

[59] Azzedine Boukerche, Anahit Martirosyan, and Richard Pazzi. An inter-cluster communication based energy aware and fault tolerant protocol for wireless sensor networks. *Mobile Networks and Applications*, 13(6):614–626, 2008.

[60] Azzedine Boukerche, Nathan J McGraw, Caron Dzermajko, and Kaiyuan Lu. Grid-filtered region-based data distribution management in large-scale distributed simulation systems. In *38th Annual Simulation Symposium*, pages 259–266. IEEE, 2005.

[61] Azzedine Boukerche and Sotiris Nikoletseas. Protocols for data propagation in wireless sensor networks. In *Wireless communications systems and networks*, pages 23–51. Springer, 2004.

[62] Azzedine Boukerche and Mirela Sechi M Annoni Notare. Behavior-based intrusion detection in mobile phone systems. *Journal of Parallel and Distributed Computing*, 62(9):1476–1490, 2002.

[63] Azzedine Boukerche, Horacio ABF Oliveira, Eduardo F Nakamura, and Antonio AF Loureiro. Localization systems for wireless sensor networks. *IEEE wireless Communications*, 14(6):6–12, 2007.

[64] Azzedine Boukerche, Horacio ABF Oliveira, Eduardo F Nakamura, and Antonio AF Loureiro. Secure localization algorithms for wireless sensor networks. *IEEE Communications Magazine*, 46(4):96–101, 2008.

[65] Azzedine Boukerche, Horacio ABF Oliveira, Eduardo F Nakamura, and Antonio AF Loureiro. Vehicular ad hoc networks: A new challenge for localization-based systems. *Computer communications*, 31(12):2838–2849, 2008.

[66] Azzedine Boukerche, Horacio ABF Oliveira, Eduardo Freire Nakamura, and Antonio AF Loureiro. Dv-loc: a scalable localization protocol using voronoi diagrams for wireless sensor networks. *IEEE Wireless Communications*, 16(2):50–55, 2009.

[67] Azzedine Boukerche, Richard Werner Nelem Pazzi, and Regina B Araujo. Hpeq a hierarchical periodic, event-driven and query-based wireless sensor network protocol. In *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l*, pages 560–567. IEEE, 2005.

[68] Azzedine Boukerche, Richard Werner Nelem Pazzi, and Regina Borges Araujo. A fast and reliable protocol for wireless sensor networks in critical conditions monitoring applications. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 157–164, 2004.

[69] Azzedine Boukerche, Richard Werner Nelem Pazzi, and Regina Borges Araujo. Fault-tolerant wireless sensor network routing protocols for the supervision of context-aware physical environments. *Journal of Parallel and Distributed Computing*, 66(4):586–599, 2006.

[70] Azzedine Boukerche, Richard WN Pazzi, and Jing Feng. An end-to-end virtual environment streaming technique for thin mobile devices over heterogeneous networks. *Computer Communications*, 31(11):2716–2725, 2008.

[71] Azzedine Boukerche and Yonglin Ren. A security management scheme using a novel computational reputation model for wireless and mobile ad hoc networks. In

*Proceedings of the 5th ACM symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 88–95, 2008.

[72] Azzedine Boukerche and Yonglin Ren. A trust-based security system for ubiquitous and pervasive computing environments. *Computer communications*, 31(18):4343–4351, 2008.

[73] Azzedine Boukerche and Yonglin Ren. A secure mobile healthcare system using trust-based multicast scheme. *IEEE Journal on Selected Areas in Communications*, 27(4):387–399, 2009.

[74] Azzedine Boukerche, Cristiano Rezende, and Richard W Pazzi. Improving neighbor localization in vehicular ad hoc networks to avoid overhead from periodic messages. In *GLOBECOM 2009-2009 IEEE Global Telecommunications Conference*, pages 1–6. IEEE, 2009.

[75] Azzedine Boukerche and E Robson. Vehicular cloud computing: Architectures, applications, and mobility. *Computer networks*, 135:171–189, 2018.

[76] Azzedine Boukerche and Steve Rogers. Gps query optimization in mobile and wireless networks. In *Proceedings. Sixth IEEE Symposium on Computers and Communications*, pages 198–203. IEEE, 2001.

[77] Azzedine Boukerche and Amber Roy. Dynamic grid-based approach to data distribution management. *Journal of Parallel and Distributed Computing*, 62(3):366–392, 2002.

[78] Azzedine Boukerche, Amber Roy, and Neville Thomas. Dynamic grid-based multicast group assignment in data distribution management. In *Proceedings Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2000)*, pages 47–54. IEEE, 2000.

[79] Azzedine Boukerche and Samer Samarah. A novel algorithm for mining association rules in wireless ad hoc sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 19(7):865–877, 2008.

[80] Azzedine Boukerche and Carl Tropper. A static partitioning and mapping algorithm for conservative parallel simulations. In *Proceedings of the eighth workshop on Parallel and distributed simulation*, pages 164–172, 1994.

[81] Azzedine Boukerche and Carl Tropper. A distributed graph algorithm for the detection of local cycles and knots. *IEEE Transactions on Parallel and Distributed Systems*, 9(8):748–757, 1998.

[82] Azzedine Boukerche, Begumhan Turgut, Nevin Aydin, Mohammad Z Ahmad, Ladislau Bölöni, and Damla Turgut. Routing protocols in ad hoc networks: A survey. *Computer networks*, 55(13):3032–3080, 2011.

[83] Azzedine Boukerche and Damla Turgut. Secure time synchronization protocols for wireless sensor networks. *IEEE Wireless Communications*, 14(5):64–69, 2007.

[84] Azzedine Boukerche, Anis Zarrad, and Regina Araujo. A cross-layer approach-based gnutella for collaborative virtual environments over mobile ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(7):911–924, 2009.

[85] Karen Boulos, Kinda Khawam, Melhem El Helou, Marc Ibrahim, Steven Martin, and Hadi Sawaya. A hybrid approach for rrh clustering in cloud radio access networks based on game theory. In *Proceedings of the 16th ACM International Symposium on Mobility Management and Wireless Access*, pages 128–132. ACM, 2018.

[86] Aaron B Brown and Margo I Seltzer. Operating system benchmarking in the wake of lmbench: A case study of the performance of netbsd on the intel x86 architecture. *ACM SIGMETRICS Performance Evaluation Review*, 25(1):214–224, 1997.

[87] Dario Bruneo, Marco Scarpa, Angelo Zaia, and Antonio Puliafito. Communication paradigms for mobile grid users. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 669–676. IEEE, 2003.

[88] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[89] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[90] Valeria Cardellini, Vittoria De Nitto Personé, Valerio Di Valerio, Francisco Facchinei, Vincenzo Grassi, Francesco Lo Presti, and Veronica Piccialli. A game-theoretic approach to computation offloading in mobile cloud computing. *Math. Program.*, 157(2):421–449, June 2016.

[91] Candace K Chan, Hailin Peng, Gao Liu, Kevin McIlwrath, Xiao Feng Zhang, Robert A Huggins, and Yi Cui. High-performance lithium battery anodes using silicon nanowires. *Nature nanotechnology*, 3(1):31–35, 2008.

[92] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[93] Ying Chang-tian and Yu Jiong. Energy-aware genetic algorithms for task scheduling in cloud computing. In *ChinaGrid Annual Conference (ChinaGrid), 2012 Seventh*, pages 43–48. IEEE, 2012.

[94] Ying Changtian and Yu Jiong. Energy-aware genetic algorithms for task scheduling in cloud computing. In *2012 Seventh ChinaGrid Annual Conference*, pages 43–48. IEEE, 2012.

[95] Huankai Chen, Frank Wang, Na Helian, and Gbola Akanmu. User-priority guided min-min scheduling algorithm for load balancing in cloud computing. In *Parallel Computing Technologies (PARCOMPTECH), 2013 National Conference on*, pages 1–8. IEEE, 2013.

[96] X. Chen, L. Jiao, W. Li, and X. Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2016.

[97] Enrique Chirivella-Perez, Jose M Alcaraz Calero, Qi Wang, and Juan Gutiérrez-Aguado. Towards a realistic 5g infrastructure emulator for experimental service deployment and performance evaluation. In *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–7. IEEE, 2018.

[98] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based

on the ratio of off-chip access to on-chip computation times. *IEEE transactions on computer-aided design of integrated circuits and systems*, 24(1):18–28, 2005.

[99] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[100] Brian P Clarke and Marty Humphrey. Beyond the" device as portal": Meeting the requirements of wireless and mobile devices in the legion grid computing system. In *ipdps*, 2002.

[101] Luis Corral, Anton B Georgiev, Alberto Sillitti, and Giancarlo Succi. A study of energy-aware implementation techniques: Redistribution of computational jobs in mobile apps. *Sustainable Computing: Informatics and Systems*, 7:11–23, 2015.

[102] Sergio Correia, Azzedine Boukerche, and Rodolfo I Meneguette. An architecture for hierarchical software-defined vehicular networks. *IEEE Communications Magazine*, 55(7):80–86, 2017.

[103] Rodolfo WL Coutinho, Azzedine Boukerche, Luiz FM Vieira, and Antonio AF Loureiro. Gedar: geographic and opportunistic routing protocol with depth adjustment for mobile underwater sensor networks. In *2014 IEEE International Conference on communications (ICC)*, pages 251–256. IEEE, 2014.

[104] Rodolfo WL Coutinho, Azzedine Boukerche, Luiz FM Vieira, and Antonio AF Loureiro. Geographic and opportunistic routing for underwater sensor networks. *IEEE Transactions on Computers*, 65(2):548–561, 2015.

[105] Rodolfo WL Coutinho, Azzedine Boukerche, Luiz FM Vieira, and Antonio AF Loureiro. A novel void node recovery paradigm for long-term underwater sensor networks. *Ad Hoc Networks*, 34:144–156, 2015.

[106] Rodolfo WL Coutinho, Azzedine Boukerche, Luiz FM Vieira, and Antonio AF Loureiro. Design guidelines for opportunistic routing in underwater networks. *IEEE Communications Magazine*, 54(2):40–48, 2016.

[107] Rodolfo WL Coutinho, Azzedine Boukerche, Luiz FM Vieira, and Antonio AF Loureiro. Underwater wireless sensor networks: A new challenge for topology control–based systems. *ACM Computing Surveys (CSUR)*, 51(1):1–36, 2018.

[108] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[109] Amir Darehshoorzadeh and Azzedine Boukerche. Underwater sensor networks: A new challenge for opportunistic routing protocols. *IEEE Communications Magazine*, 53(11):98–107, 2015.

[110] Robson E De Grande and Azzedine Boukerche. Dynamic balancing of communication and computation load for hla-based simulations on large-scale distributed systems. *Journal of Parallel and Distributed Computing*, 71(1):40–52, 2011.

[111] Horacio Antonio Braga Fernandes De Oliveira, Azzedine Boukerche, Eduardo Freire Nakamura, and Antonio Alfredo Ferreira Loureiro. An efficient directed localization recursion protocol for wireless sensor networks. *IEEE Transactions on Computers*, 58(5):677–691, 2008.

[112] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[113] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 27–33. Ieee, 2010.

[114] Aaron Yi Ding, Bo Han, Yu Xiao, Pan Hui, Aravind Srinivasan, Markku Kojo, and Sasu Tarkoma. Enabling energy-aware collaborative mobile data offloading for smartphones. In *2013 IEEE International Conference on Sensing, Communications and Networking (SECON)*, pages 487–495. IEEE, 2013.

[115] Yan Ding, Gaochao Xu, Chunyi Wu, Liang Hu, Yunan Zhai, and Jia Zhao. Explore virtual machine deployment to mobile cloud computing for multi-tenancy and energy conservation in wireless network. *Cluster Computing*, 20(4):3263–3274, 2017.

[116] X. Dong, J. Zheng, Y. Cai, J. Yang, and Y. Wang. Share communication and energy resources for mobile cloud computing: An optimal cooperative contract approach.

In *Proceedings of the 9th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6, 2017.

[117] Corentin Dupont, Thomas Schulze, Giovanni Giuliani, Andrey Somov, and Fabien Hermenier. An energy aware framework for virtual machine placement in cloud federated data centres. In *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*, pages 1–10. IEEE, 2012.

[118] Truong Vinh Truong Duy, Yukinori Sato, and Yasushi Inoguchi. Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

[119] Elie El Ajaltouni, Azzedine Boukerche, and Ming Zhang. An efficient dynamic load balancing scheme for distributed simulations on a grid infrastructure. In *2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 61–68. IEEE, 2008.

[120] Mourad Elhadef, Azzedine Boukerche, and Hisham Elkadiki. Diagnosing mobile ad-hoc networks: two distributed comparison-based self-diagnosis protocols. In *Proceedings of the 4th ACM international workshop on Mobility management and wireless access*, pages 18–27, 2006.

[121] Mourad Elhadef, Azzedine Boukerche, and Hisham Elkadiki. Performance analysis of a distributed comparison-based self-diagnosis protocol for wireless ad-hoc networks. In *Proceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems*, pages 165–172, 2006.

[122] Mourad Elhadef, Azzedine Boukerche, and Hisham Elkadiki. A distributed fault identification protocol for wireless and mobile ad hoc networks. *Journal of parallel and distributed computing*, 68(3):321–335, 2008.

[123] Amir Erfan Eshratifar and Massoud Pedram. Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment. In *Proceedings of the 28th Edition on Great Lakes Symposium on VLSI*, pages 1–6. ACM, 2018.

[124] Constantinos Evangelinos and C Hill. Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. *ratio*, 2(2.40):2–34, 2008.

[125] Xiaochen Fan, Xiangjian He, Deepak Puthal, Shiping Chen, Chaocan Xiang, Priyadarsi Nanda, and Xunpeng Rao. Ctom: Collaborative task offloading mechanism for mobile cloudlet networks. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[126] Eugen Feller, Louis Rilling, and Christine Morin. Energy-aware ant colony based workload placement in clouds. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, pages 26–33. IEEE Computer Society, 2011.

[127] Debessay Fesehaye, Yunlong Gao, Klara Nahrstedt, and Guijun Wang. Impact of cloudlets on interactive mobile cloud applications. In *Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International*, pages 123–132. IEEE, 2012.

[128] Huber Flores and Satish Srirama. Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning. In *Proceeding of the fourth ACM workshop on Mobile cloud computing and services*, pages 9–16. ACM, 2013.

[129] Cisco VNI Forecast. Cisco visual networking index: Global mobile data traffic forecast update 2009-2014. *Cisco Public Information, February*, 9, 2010.

[130] George H. Forman and John Zahorjan. The challenges of mobile computing. *Computer*, 27(4):38–47, 1994.

[131] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[132] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pages 1–10. Ieee, 2008.

[133] Steve B Furber. *ARM system Architecture*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[134] K. Gai, M. Qiu, H. Zhao, and M. Liu. Energy-aware optimal task assignment for mobile heterogeneous embedded systems in cloud computing. In *Proceedings of the IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 198–203, 2016.

[135] Keke Gai, Meikang Qiu, and Hui Zhao. Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing. *Journal of Parallel Distributed Computing*, 111(C):126–135, January 2018.

[136] Keke Gai, Meikang Qiu, Hui Zhao, Lixin Tao, and Ziliang Zong. Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. *Journal of Network and Computer Applications*, 59:46–54, 2016.

[137] Yongqiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79(8):1230–1242, 2013.

[138] Yujia Ge and Guiyi Wei. Ga-based task scheduler for the cloud computing systems. In *Web Information Systems and Mining (WISM), 2010 International Conference on*, volume 2, pages 181–186. IEEE, 2010.

[139] Preetam Ghosh, Nirmalya Roy, Sajal K Das, and Kalyan Basu. A game theory based pricing strategy for job allocation in mobile grids. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 82. IEEE, 2004.

[140] Lazaros Gkatzikis and Iordanis Koutsopoulos. Migrate or not? exploiting dynamic task migration in mobile cloud computing systems. *IEEE Wireless Communications*, 20(3):24–32, 2013.

[141] Andrea Goldsmith. *Wireless communications*. Cambridge university press, 2005.

[142] Mark Gordon, Lide Zhang, Birjodh Tiwana, R Dick, ZM Mao, and L Yang. Powertutor: A power monitor for android-based mobile platforms. *An Android Application*, 2013.

[143] Mark S Gordon, Davoud Anoushe Jamshidi, Scott A Mahlke, Zhuoqing Morley Mao, and Xu Chen. Comet: Code offload by migrating execution transparently. In *OSDI*, volume 12, pages 93–106, 2012.

[144] Hadi Goudarzi and Massoud Pedram. Energy-efficient virtual machine replication and placement in a cloud computing system. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 750–757. IEEE, 2012.

[145] Shichao Guan. *A Cloudlet-based Mobile Cloud System for Heterogeneous Energy-aware Offloading*. PhD thesis, University of Ottawa, the School of Electrical Engineering and Computer Science, 2018.

[146] Shichao Guan, Robson Eduardo De Grande, and Azzedine Boukerche. A cloudlet-based task-centric offloading to enable energy-efficient mobile applications. In *Proceedings of the IEEE Symposium on Computers and Communications*, pages 564–569, 2017.

[147] S. Guo, B. Xiao, Y. Yang, and Y. Yang. Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. In *Proceedings of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9, 2016.

[148] Songtao Guo, Jiadi Liu, Yuanyuan Yang, Bin Xiao, and Zhetao Li. Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing. *IEEE Transactions on Mobile Computing*, 18(2):319–333, 2018.

[149] Abdul Hameed, Alireza Khoshkbarforoushha, Rajiv Ranjan, Prem Prakash Jayaraman, Joanna Kolodziej, Pavan Balaji, Sherali Zeadally, Qutaibah Marwan Malluhi, Nikos Tziritas, Abhinav Vishnu, Samee U. Khan, and Albert Zomaya. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing*, 98(7):751–774, 2016.

[150] XiaoShan He, XianHe Sun, and Gregor Von Laszewski. Qos guided min-min heuristic for grid task scheduling. *Journal of Computer Science and Technology*, 18(4):442–451, 2003.

[151] Jinhua Hu, Jianhua Gu, Guofei Sun, and Tianhai Zhao. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *2010 3rd International symposium on parallel architectures, algorithms and programming*, pages 89–96. IEEE, 2010.

[152] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[153] Chih-Wei Huang, M Chen, and D Zavin. Android x86-porting android to x86. *Available on Dec*, 2011.

[154] Jim Huang. Understanding the dalvik virtual machine. *Google Technology User Groups, Taipei*, 2012.

[155] Alexandru Iosup, Simon Ostermann, M Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick HJ Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):931–945, 2011.

[156] Keith R Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J Wasserman, and Nicholas J Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 159–168. IEEE, 2010.

[157] Sung Ho Jang, Tae Young Kim, Jae Kwon Kim, and Jong Sik Lee. The study of genetic algorithm-based task scheduling for cloud computing. *International Journal of Control and Automation*, 5(4):157–162, 2012.

[158] Zhefeng Jiang and Shiwen Mao. Energy delay tradeoff in cloud offloading for multi-core mobile devices. *IEEE Access*, 3:2306–2316, 2015.

[159] 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*, 9(6):895–909, 2016.

[160] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. The one simulator for dtn protocol evaluation. In *Proceedings of the 2nd international conference on simulation tools and techniques*, page 55, 2009.

[161] Haegyeom Kim, Kyu-Young Park, Jihyun Hong, and Kisuk Kang. All-graphene-battery: bridging the gap between supercapacitors and lithium ion batteries. *Scientific reports*, 4, 2014.

[162] Kyong Hoon Kim, Anton Beloglazov, and Rajkumar Buyya. Power-aware provisioning of cloud resources for real-time services. In *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*, page 1. ACM, 2009.

[163] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230, 2007.

[164] Dzmitry Kliazovich, Pascal Bouvry, Yury Audzevich, and Samee Ullah Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5. IEEE, 2010.

[165] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. Dens: data center energy-efficient network-aware scheduling. *Cluster computing*, 16(1):65–75, 2013.

[166] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[167] D. Kumar and R. Sharma. Data synchronization and offloading techniques for energy optimization in mobile cloud computing. In *Proceedings of the International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*, pages 633–638, 2017.

[168] Jitender Kumar and Amita Malik. Maya: An approach for energy and cost optimization for mobile cloud computing environments. In V. B. Aggarwal, Vasudha Bhatnagar, and Durgesh Kumar Mishra, editors, *Big Data Analytics*, pages 575–581. Springer Singapore, 2018.

[169] Willis Lang and Jignesh M Patel. Energy management for mapreduce clusters. *Proceedings of the VLDB Endowment*, 3(1-2):129–139, 2010.

[170] Hwa Min Lee, Young-Sik Jeong, and Haeng Jin Jang. Performance analysis based resource allocation for green cloud computing. *The Journal of Supercomputing*, 69(3):1013–1026, 2014.

[171] Jacob Leverich and Christos Kozyrakis. On the energy (in) efficiency of hadoop clusters. *ACM SIGOPS Operating Systems Review*, 44(1):61–65, 2010.

[172] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong. Enacloud: An energy-saving application live placement approach for cloud computing environments. In *Proceedings of the IEEE International Conference on Cloud Computing*, pages 17–24, 2009.

[173] Xiaofang Li, Yingchi Mao, Xianjian Xiao, and Yanbin Zhuang. An improved max-min task-scheduling algorithm for elastic cloud. In *Computer, Consumer and Control (IS3C), 2014 International Symposium on*, pages 340–343. IEEE, 2014.

[174] Y. Li, M. Chen, W. Dai, and M. Qiu. Energy optimization with dynamic task scheduling mobile cloud computing. *IEEE Systems Journal*, 11(1):96–105, 2017.

[175] Fangming Liu, Peng Shu, Hai Jin, Linjie Ding, Jie Yu, Di Niu, and Bo Li. Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wireless communications*, 20(3):14–22, 2013.

[176] Jing Liu, Xing-Guo Luo, Xing-Ming Zhang, Fan Zhang, and Bai-Nan Li. Job scheduling model for cloud computing based on multi-objective genetic algorithm. *IJCSI International Journal of Computer Science Issues*, 10(1):134–139, 2013.

[177] Kaiyang Liu, Jun Peng, Heng Li, Xiaoyong Zhang, and Weirong Liu. Multi-device task offloading with time-constraints for energy efficiency in mobile cloud computing. *Future Generations Computer Systems*, 64(C):1–14, November 2016.

[178] Xing Liu, Songtao Guo, and Yuanyuan Yang. Task offloading with execution cost minimization in heterogeneous mobile cloud computing. In Liehuang Zhu and Sheng Zhong, editors, *Proceedings of the 13th International Conference on Mobile Ad-hoc and Sensor Networks*, pages 509–522. Springer Singapore, 2017.

[179] Yanchen Liu, Myung J Lee, and Yanyan Zheng. Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system. *IEEE Transactions on Mobile Computing*, 15(10):2398–2410, 2016.

[180] Xiaoqiang Ma, Yuan Zhao, Lei Zhang, Haiyang Wang, and Limei Peng. When mobile terminals meet the cloud: computation offloading as the bridge. *IEEE Network*, 27(5):28–33, 2013.

[181] Pavel Mach and Zdenek Becvar. Cloud-aware power control for real-time application offloading in mobile edge computing. *Transactions on Emerging Telecommunications Technologies*, 27(5):648–661, 2016.

[182] Y. Mao, J. Zhang, and K. B. Letaief. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605, 2016.

[183] Yuyi Mao, Jun Zhang, SH Song, and Khaled Ben Letaief. Power-delay tradeoff in multi-user mobile-edge computing systems. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2016.

[184] Anahit Martirosyan, Azzedine Boukerche, and Richard Pazzi. A taxonomy of cluster-based routing protocols for wireless sensor networks. In *2008 International Symposium on Parallel Architectures, Algorithms, and Networks (i-span 2008)*, pages 247–253. IEEE, 2008.

[185] Mohammad Masdari, Farbod Salehi, Marzie Jalali, and Moazam Bidaki. A survey of pso-based scheduling algorithms in cloud computing. *Journal of Network and Systems Management*, 25(1):122–158, 2017.

[186] D. Mazza, D. Tarchi, and G. E. Corazza. A unified urban mobile cloud computing offloading mechanism for smart cities. *IEEE Communications Magazine*, 55(3):30–37, 2017.

[187] Don McGregor, Don Brutzman, and John Grant. Open-dis: An open source implementation of the dis protocol for c++ and java. In *Simulator Interoperability Working Group (SISO) Fall Workshop*, 2008.

[188] Fidan Mehmeti and Thrasyvoulos Spyropoulos. Performance modeling, analysis, and optimization of delayed mobile data offloading for mobile users. *IEEE/ACM Transactions on Networking (TON)*, 25(1):550–564, 2017.

[189] Piyush Mehrotra, Jahed Djomehri, Steve Heistand, Robert Hood, Haoqiang Jin, Arthur Lazanoff, Subhash Saini, and Rupak Biswas. Performance evaluation of amazon ec2 for nasa hpc applications. In *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, pages 41–50. ACM, 2012.

[190] Peter Mell and Tim Grance. *The NIST definition of cloud computing*. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.

[191] Aravind Menon, Alan L Cox, and Willy Zwaenepoel. Optimizing network virtualization in xen. In *USENIX Annual Technical Conference*, number LABOS-CONF-2006-003, 2006.

[192] Mohand Mezmaz, Nouredine Melab, Yacine Kessaci, Young Choon Lee, E-G Talbi, Albert Y Zomaya, and Daniel Tuyttens. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *Journal of Parallel and Distributed Computing*, 71(11):1497–1508, 2011.

[193] Sambit Kumar Mishra, Bibhudatta Sahoo, and Priti Paramita Parida. Load balancing in cloud computing: A big picture. *Journal of King Saud University-Computer and Information Sciences*, 2018.

[194] Cisco VNI Mobile. Cisco visual networking index: global mobile data traffic forecast update, 2011–2016. *San Jose, CA*, 1, 2014.

[195] Sunil Nakrani and Craig Tovey. On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behavior*, 12(3-4):223–240, 2004.

[196] Horacio ABF Oliveira, Azzedine Boukerche, Eduardo F Nakamura, and Antonio AF Loureiro. Localization in time and space for wireless sensor networks: An efficient and lightweight algorithm. *Performance Evaluation*, 66(3-5):209–222, 2009.

[197] Horacio ABF Oliveira, Eduardo F Nakamura, Antonio AF Loureiro, and Azzedine Boukerche. Error analysis of localization systems for sensor networks. In *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 71–78, 2005.

[198] Renê Oliveira, Carlos Montez, Azzedine Boukerche, and Michelle S Wangham. Reliable data dissemination protocol for vanet traffic safety applications. *Ad Hoc Networks*, 63:30–44, 2017.

[199] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. A performance analysis of EC2 cloud computing ser-

vices for scientific computing. In *Proc. of the International Conference on Cloud Computing*, pages 115–131. Springer, 2009.

[200] Mazliza Othman, Sajjad Ahmad Madani, Samee Ullah Khan, et al. A survey of mobile cloud computing application models. *IEEE Communications Surveys & Tutorials*, 16(1):393–413, 2014.

[201] Nicholas Palmer, Roelof Kemp, Thilo Kielmann, and Henri Bal. Ibis for mobility: solving challenges of mobile computing using grid techniques. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, page 17. ACM, 2009.

[202] Richard W Pazzi and Azzedine Boukerche. Propane: A progressive panorama streaming protocol to support interactive 3d virtual environment exploration on graphics-constrained devices. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 11(1):1–22, 2014.

[203] Richard WN Pazzi and Azzedine Boukerche. Mobile data collector strategy for delay-sensitive applications over wireless sensor networks. *Computer Communications*, 31(5):1028–1039, 2008.

[204] Trevor Pering, Tom Burd, and Robert Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the 1998 international symposium on Low power electronics and design*, pages 76–81. ACM, 1998.

[205] Padmanabhan Pillai and Kang G Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 89–102. ACM, 2001.

[206] Jan Plachy, Zdenek Becvar, and Pavel Mach. Path selection enabling user mobility and efficient distribution of data for computation at the edge of mobile network. *Computer Networks*, 108:357–370, 2016.

[207] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 43–56. ACM, 2011.

[208] M Reza Rahimi, Jian Ren, Chi Harold Liu, Athanasios V Vasilakos, and Nalini Venkatasubramanian. Mobile cloud computing: A survey, state of art and future directions. *Mobile Networks and Applications*, 19(2):133–143, 2014.

[209] M. Rahman, J. Gao, and W. T. Tsai. Energy saving in mobile cloud computing*. In *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*, pages 285–291, 2013.

[210] Yonglin Ren and Azzedine Boukerche. Modeling and managing the trust for wireless and mobile ad hoc networks. In *2008 IEEE International Conference on Communications*, pages 2129–2133. IEEE, 2008.

[211] Yonglin Ren, Richard Werner, Nelem Pazzi, and Azzedine Boukerche. Monitoring patients via a secure and mobile healthcare system. *IEEE Wireless Communications*, 17(1):59–65, 2010.

[212] Cristiano Rezende, Azzedine Boukerche, Heitor S Ramos, and Antonio AF Loureiro. A reactive and scalable unicast solution for video streaming over vanets. *IEEE Transactions on Computers*, 64(3):614–626, 2014.

[213] Cristiano Rezende, Abdelhamid Mammeri, Azzedine Boukerche, and Antonio AF Loureiro. A receiver-based video dissemination solution for vehicular networks with content transmissions decoupled from relay node selection. *Ad Hoc Networks*, 17:1–17, 2014.

[214] Jeffrey Richter. *CLR via c#*, volume 4. Microsoft Press Redmond, 2006.

[215] Naidila Sadashiv and SM Dilip Kumar. Cluster, grid and cloud computing: A detailed comparison. In *Computer Science & Education (ICCSE), 2011 6th International Conference on*, pages 477–482. IEEE, 2011.

[216] Samer Samarah, Muhannad Al-Hajri, and Azzedine Boukerche. A predictive energy-efficient technique to support object-tracking sensor networks. *IEEE Transactions on Vehicular Technology*, 60(2):656–663, 2010.

[217] Joy Lal Sarkar, Chhabi Rani Panigrahi, Bibudhendu Pati, Rajani Trivedi, and Shibendu Debbarma. E2g: A game theory-based energy efficient transmission policy for mobile cloud computing. In *Proceedings of the International Conference on Advanced Computing and Intelligent Engineering*, Progress in Advanced Computing and Intelligent Engineering. Springer Singapore, 2017.

[218] M. Sarvabhatla, S. Konda, C. S. Vorugunti, and M. M. N. Babu. A network aware energy efficient offloading algorithm for mobile cloud computing over 5g network. In *Proceedings of the IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 69–74, 2017.

[219] Mahadev Satyanarayanan. Fundamental challenges in mobile computing. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 1–7. ACM, 1996.

[220] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, (4):14–23, 2009.

[221] Stefano Secci, Patrick Raad, and Pascal Gallard. Linking virtual machine mobility to user mobility. *IEEE Transactions on Network and Service Management*, 13(4):927–940, 2016.

[222] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3), 2012.

[223] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[224] W. Shi and S. Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[225] Fabrício A Silva, Azzedine Boukerche, Thais RM Braga Silva, Linnyer B Ruiz, Eduardo Cerqueira, and Antonio AF Loureiro. Vehicular networks: A new challenge for content-delivery-based applications. *ACM Computing Surveys (CSUR)*, 49(1):1–29, 2016.

[226] Subashini Subashini and Veeraruna Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1):1–11, 2011.

[227] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *USENIX Annual Technical Conference, General Track*, pages 1–14, 2001.

[228] Vanish Talwar, Dejan Milojicic, Qinyi Wu, Calton Pu, Wenchang Yan, and Gueyoung Jung. Approaches for service deployment. *IEEE Internet Computing*, 9(2):70–80, 2005.

[229] Mati B. Terefe, Heezin Lee, Nojung Heo, Geoffrey C. Fox, and Sangyoon Oh. Energy-efficient multisite offloading policy using markov decision process for mobile cloud computing. *Pervasive Mobile Computing*, 27(C):75–89, 2016.

[230] Thuan Thai and Hoang Lam. . *NET framework essentials*. " O'Reilly Media, Inc.", 2003.

[231] Zhiying Tu, Gregory Zacharewicz, and David Chen. Developing a web-enabled hla federate based on portico rti. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 2289–2301. IEEE, 2011.

[232] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. Wikipedia workload analysis for decentralized hosting. *Computer Networks*, 53(11):1830–1845, 2009.

[233] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[234] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60, 2008.

[235] Carl A Waldspurger. Memory resource management in vmware esx server. *ACM SIGOPS Operating Systems Review*, 36(SI):181–194, 2002.

[236] Shu-Ching Wang, Kuo-Qin Yan, Wen-Pin Liao, and Shun-Sheng Wang. Towards a load balancing in a three-level cloud computing network. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 1, pages 108–113. IEEE, 2010.

[237] X. Wang, J. Wang, X. Wang, and X. Chen. Energy and delay tradeoff for application offloading in mobile cloud computing. *IEEE Systems Journal*, 11(2):858–867, 2017.

[238] Yating Wang, Ray Chen, and Ding-Chau Wang. A survey of mobile cloud computing applications: perspectives and challenges. *Wireless Personal Communications*, 80(4):1607–1623, 2015.

[239] Christof Weinhardt, Dipl-Inform-Wirt Arun Anandasivam, Benjamin Blau, Dipl-Inform Nikolay Borissov, Dipl-Math Thomas Meinl, Dipl-Inform-Wirt Wibke Michalk, and Jochen Stößer. Cloud computing–a classification, business models, and research directions. *Business & Information Systems Engineering*, 1(5):391–399, 2009.

[240] Mark Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, 1993.

[241] M Weisser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.

[242] Md Whaiduzzaman, Anjum Naveed, and Abdullah Gani. Mobicore: Mobile device based cloudlet resource enhancement for optimal task response. *IEEE Transactions on Services Computing*, 11(1):144–154, 2018.

[243] Timothy Wood, Prashant J Shenoy, Arun Venkataramani, and Mazin S Yousif. Black-box and gray-box strategies for virtual machine migration. In *NSDI*, volume 7, pages 17–17, 2007.

[244] Huaming Wu, Qiushi Wang, and Katinka Wolter. Tradeoff between performance improvement and energy saving in mobile cloud offloading systems. In *2013 IEEE international conference on communications workshops (ICC)*, pages 728–732. IEEE, 2013.

[245] Huaming Wu and Katinka Wolter. Stochastic analysis of delayed mobile offloading in heterogeneous networks. *IEEE Transactions on Mobile Computing*, 17(2):461–474, 2018.

[246] Linlin Wu, Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. Sla-based resource provisioning for hosted software-as-a-service applications in cloud computing environments. *IEEE Transactions on services computing*, 7(3):465–485, 2014.

[247] Zhen Xiao, Weijia Song, and Qi Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE transactions on parallel and distributed systems*, 24(6):1107–1117, 2013.

[248] Lei Yang, Jiannong Cao, Yin Yuan, Tao Li, Andy Han, and Alvin Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. *ACM SIGMETRICS Performance Evaluation Review*, 40(4):23–32, 2013.

[249] Sonia Yassa, Rachid Chelouah, Hubert Kadima, and Bertrand Granado. Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. *The Scientific World Journal*, 2013, 2013.

[250] C. You, K. Huang, and H. Chae. Energy efficient mobile cloud computing powered by wireless energy transfer. *IEEE Journal on Selected Areas in Communications*, 34(5):1757–1771, 2016.

[251] C. You, K. Huang, H. Chae, and B. H. Kim. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 16(3):1397–1411, 2017.

[252] Maram Bani Younes and Azzedine Boukerche. Intelligent traffic light controlling algorithms using vehicular networks. *IEEE transactions on vehicular technology*, 65(8):5887–5899, 2015.

[253] Maram Bani Younes and Azzedine Boukerche. A performance evaluation of an efficient traffic congestion detection protocol (ecode) for intelligent transportation systems. *Ad Hoc Networks*, 24:317–336, 2015.

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

[255] Weishan Zhang, Shouchao Tan, Feng Xia, Xiufeng Chen, Zhongwei Li, Qinghua Lu, and Su Yang. A survey on decision making for task migration in mobile cloud environments. *Personal and Ubiquitous Computing*, 20(3):295–309, 2016.

[256] Weiwen Zhang, Yonggang Wen, and Dapeng Oliver Wu. Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In *INFOCOM, 2013 Proceedings IEEE*, pages 190–194. IEEE, 2013.

[257] Zhenxia Zhang, Azzedine Boukerche, and Richard Pazzi. A novel multi-hop clustering scheme for vehicular ad-hoc networks. In *Proceedings of the 9th ACM international symposium on Mobility management and wireless access*, pages 19–26, 2011.

[258] Zhenxia Zhang, Richard W Pazzi, and Azzedine Boukerche. A mobility management scheme for wireless mesh networks based on a hybrid routing protocol. *Computer Networks*, 54(4):558–572, 2010.

[259] Bowen Zhou and Rajkumar Buyya. Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions. *ACM Computing Surveys (CSUR)*, 51(1):13, 2018.

[260] Bowen Zhou and Rajkumar Buyya. Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions. *ACM Computer Surveys*, 51(1):13:1–13:38, 2018.

[261] Bowen Zhou, Amir Vahid Dastjerdi, Rodrigo N Calheiros, Satish Narayana Srirama, and Rajkumar Buyya. A context sensitive offloading scheme for mobile cloud computing service. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 869–876. IEEE, 2015.

[262] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation computer systems*, 28(3):583–592, 2012.