NETWORK-AWARE RESOURCE MANAGEMENT FOR MOBILE CLOUD

Chathura Madhusanka Sarathchandra Magurawalage

A thesis submitted for the degree of PhD
School of Computer Science and Electronic Engineering
University of Essex
February, 2017

Abstract

The author proposes a novel system architecture for mobile cloud computing (MCC) that includes a controller for managing computing and communication resources in Cloud Radio Access Network (C-RAN) environment. The gathered monitoring information in the controller is used when making resource allocation/management decisions. A unified protocol has been proposed, which utilises the same packet format for mobile task offloading and resource management. Moreover, the packet format and the message types of the protocol have been presented.

An MCC scenario (i.e., cloudlet+clone) that consists of a cloudlet layer has been studied, in which the cloudlets are deployed next to Wi-Fi access points and serve as a localised service point in proximity to mobile devices to improve the performance of mobile cloud services. On top of this, an offloading algorithm is proposed with the main aim of deciding whether to offload to a clone or a cloudlet.

The architecture described above has been implemented as a prototype by focussing on resource management in the mobile cloud. A partial implementation of a resource monitoring module that monitors both computing and communication resources have also been presented. Auto-scaling enables efficient computing resource management in the mobile cloud. An empirical performance analysis of cloud vertical scaling for mobile cloud resource management has been conducted.

The working procedures of the proposed unified protocol have been illustrated to show the mobile task offloading and resource allocation functions. Simulation results of cloudlet+clone mobile task offloading algorithm demonstrate the effectiveness and efficiency of the presented task offloading architecture, and offloading algorithm on response time and energy consumption. The empirical vertical auto-scaling performance analysis for mobile cloud resource allocation shows that time delays when scaling resources (CPU, RAM, disk) in mobile cloud varies. Moreover, the scaling delay depends on the scaling amount at the given iteration.

Acknowledgements

I would like to express my gratitude to everyone who directly or indirectly supported me throughout my PhD. Especially, I am grateful to my mother for her numerous amounts of support and patience throughout my undergraduate and postgraduate studies. I would also like to express my sincere thanks to Prof. Kun Yang for his supervision and support. I am grateful for all co-authors of my publications for their hard work. Finally, I would like to thank UK EPSRC NIRVANA and EU Horizon 2020 iCIRRUS projects for funding my research, and its members for past and ongoing collaborations, valuable discussions and for providing constructive feedback on my research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

User equipments (UEs) (e.g., smartphone, tablet, wearable device, and digital camera) are playing an important role in new application scenarios (e.g., virtual reality, augmented reality and surveillance system). While resource-constrained UEs (CPU, GPU, memory, storage, and battery lifetime) have driven a dramatic surge in developing new paradigms to handle computation intensive tasks [1]. For example, computationally intensive applications requiring a large amount of computing capacity are not suitable to run on mobile or portable devices.

Various parties (e.g., hardware manufacturers, operating system developers, application developers) that are involved in developing new generations of mobile devices, in industry and academia, have investigated on ways to boost computing resources on mobile devices while also increasing energy efficiency. There have been a numerous amount of solutions proposed over the past few years. Some such proposed technologies predominantly focused on optimising existing hardware resources for efficient processing (e.g. energy efficient CPU scheduling), and others' focus were to increase the number of local resources in hardware while at the same time increasing the portability. However, in today's mobile devices the above issues (resource scarcity) continue to persist. Therefore, is there any other way that mobile applications may help increase mobile computing efficiency?

Mobile cloud computing (MCC) [2] [3] provides a solution where UEs offload computationally intensive tasks to a remote resourceful cloud (e.g., EC2) [4], thereby saving processing power and energy. Furthermore, Kumar *et al.* [1] discovers a computing-communication trade-off, and concludes that mobile task offloading is beneficial when the computing intensity of the task in question is higher, and when the required network resources that are required to transfer the offloading task to the remote mobile cloud is relatively lower. Kumar *et al.* also

emphasises the need for high bandwidth wireless networks for task offloading to be efficient.

Before offloading, all mobile applications go through a procedure called code partitioning [5]. In this step, the offloading framework selects the tasks that should be executed locally and tasks that may be offloaded to a remote cloud. Some tasks (such as code that perform Input and Output operations) might not inherently be able to execute remotely. Subsequently, the offloading algorithm in the offloading framework that resides on the mobile device may make decisions on *what* (which tasks), *when* (when it is beneficial to offload) and *where* to offload. Various researchers have proposed different architectures for offloading frameworks and have provided their implementations. Some of which are MAUI [6], Thinkair [7], CloneCloud [5], Cuckoo [8]. The architectures of which predominantly dependent on the offloading type/level (thread level, method level, code level), and the implementation platform (programming language, used application libraries). However, the partitioning methods of task offloading are out of the scope of this thesis.

On the other hand, task offloading generates data intensive workloads, which may become one of the leading influential factors of the unprecedented mobile traffic growth [9]. It has been predicted that mobile traffic will increase exponentially to 100 times by the year 2020 [10]. The dynamics of substantially increased data rates requires that cellular infrastructure must be flexible and reconfigurable, supporting simplified deployment and management of radio access networks. While conventional radio access network may incur high cost, high latency and data exchange inefficiencies [11]. It lacks the efficiency to support centralised interference management and the flexibility to migrate services to network edge for computation intensive applications.

To ensure highly efficient network operation and flexible service delivery when handling mobile internet traffic surging, Cloud Radio Access Network (C-RAN) [11] [12] brings cloud computing technologies into mobile networks by centralising baseband processing units (BBU) of the radio access network. It moves the BBU from traditional base station to the cloud and leaves the remote radio heads (RRH) distributed geographically. The RRHs are connected to the BBU pool via high bandwidth and low-latency fronthaul. The BBU pool may be realised by the virtual machines (VM) in data centres, and the centralised baseband processing enables BBU to be dynamically configured and shared on demand [13]. In this case, with the transition from conventional hardware-based environment to a software-based infrastructure, C-RAN may achieve flexible management of BBU resources.

It is worth mentioning that C-RAN uses centralised BBU to do baseband processing, while MCC carries out distributed task offloading by shifting computing tasks from the mobile device to mobile cloud. However, the cloud in MCC scenario is usually in a wide area network (WAN), and it is tough to control delay and jitter at WAN scale, thus offloading tasks to the public cloud may suffer from high latency via the Internet [14]. For example, augmented reality requires low latency in order to provide correct information according to user location and orientation, while offloading task to the remote cloud may incur information distortion due to delayed data transmission. As offloading tasks are delay constrained, it is always beneficial to bring computing resources as close to the user as possible [14]. Therefore, it is evident that it may significantly reduce offloading latency when UEs offload computationally intensive tasks to a computing resource-rich location within the radio access network and in proximity to UEs. Moreover, integrating MCC task offloading with the C-RAN environment stands as a valid research problem.

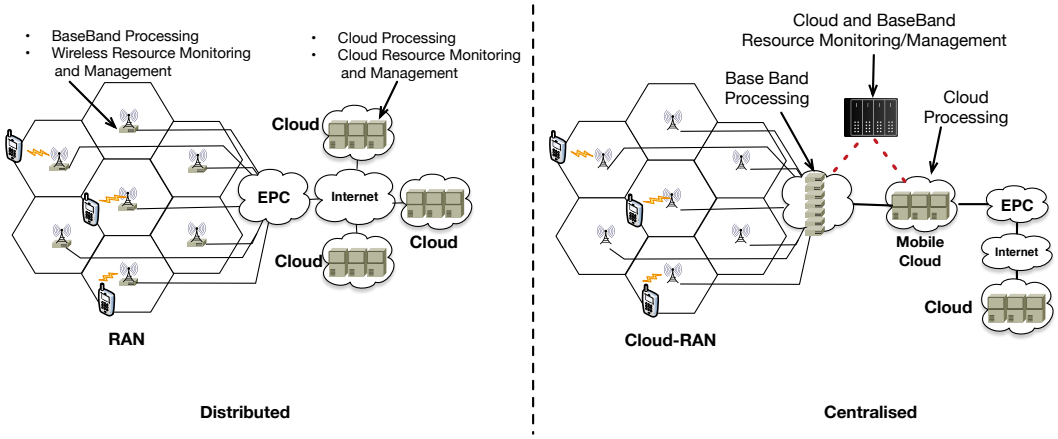

Figure 1.1: The traditional (distributed) vs the proposed (centralised) multi-resource management architectures.

Figure 1.1 shows the hybrid deployment of C-RAN with MCC for computation offloading (on the right side), as opposed to the traditional mobile cloud on the internet depicted on the left side. Connected with geographically distributed RRHs and centralised BBUs, UEs get access to the VMs (called mobile clones) in a mobile cloud for computation offloading. For computation offloading requests, data is first transmitted by the base station (RRH and BBU) via the uplink. Once processed by an MC in the mobile cloud, the results will be returned to UEs via

the downlink. However, the focus of most of the work in this thesis is on the mobile cloud side.

As shown in figure 1.1, a new category of resource has been introduced to the traditional mobile operator's network. This is not only beneficial to the mobile users (UEs) for offloading computationally intensive tasks to the cloud, but there are a number of aspects that operators may benefit from. Operators are now not only a network pipe provider, but they may also offer computing services to the subscribers. Moreover, the operators may let the subscribers pay more on top of the current price plan for extra computing services that they receive, by introducing new price plans for mobile task offloading. One may introduce new components into existing systems, but such components still have to be managed for utilising the resources efficiently.

Software Defined Networking (SDN) [15] is a centralised architecture that augments a data plane and a control plane from traditional networks. Centralised controller nodes have been introduced into the network for managing resources. In literature, such SDN-based architectures and management controllers have been introduced to wireless networks in multiple accounts [16] [17]. Their main focus has been on defining SDN modules and interfaces (northbound and southbound) for wireless networks [16] for efficient service deployments. Moreover, SDN has been proposed for Long Term Evolution (LTE) wireless network control [16] (e.g. Interference mitigation, network access selection).

Both mobile cloud and C-RAN resources that are co-located in the operators' network have to be managed efficiently for gaining optimum performance. Therefore a central controller has been introduced for managing C-RAN and Mobile cloud resources. Such a controller may still be able to control wireless network resources as in [16], while also managing computing resources when mobile tasks are offloaded to the cloud. Figure 1.1 depicts a controller that manages mobile cloud and baseband resources in C-RAN. The dotted lines illustrate the connectivity to both the BBU pool and the mobile cloud for sending control signals. In this architecture, a signalling protocol needs to be designed for task offloading and managing wireless resources in operator's network and computing resources in the mobile cloud. However, the work on this thesis mainly focuses on mobile task offloading, mobile cloud resource management and on developing a prototype of such a system architecture to demonstrate that the described architecture may be implemented.

The cloud resources in the mobile cloud have to be managed efficiently while also taking delay constraints of compute (task) offloading into consideration. Auto-scaling [18] enables cloud administrators to dynamically scale computing re-

sources in the cloud for their applications to adapt to workload fluctuations. There are two types of auto-scaling. 1) Horizontal scaling adds and removes virtual machines (VM) from an existing VM pool that serves an application, 2) Vertical scaling adds and removes virtual resources from an existing VM. However, horizontal scaling has been used more often, in comparison to vertical scaling in literature. Moreover, horizontal scaling allows the application to achieve higher throughput levels per each addition, but the deployment cost is greater than vertical scaling [19].

When scaling vertically, the resource provisioning introduces delay, which makes the desired effect arrive late. But not many previous auto-scaling techniques has taken these delays into consideration. Therefore, recent literature [20] stresses the need for future work on auto scaling, taking auto scaling delay times into consideration. Auto-scaling may scale either virtual disk, memory or CPU resources (e.g. vCPU scaling is the most used auto-scaling resource) or a combination of types of resources. Also, one may scale up and scale down VMs depending on the current amount of workload. Moreover, when scaling resources, one may scale "continuously" by adding/removing just one resource from the existing VM or may add or remove more than one resource from the VM (non-continuously). All of the aforementioned scenarios should be taken into consideration when designing efficient offloading techniques. Therefore, it is important to understand the trends of such resize delays before designing auto-scaling algorithms, especially when auto-scaling for delay sensitive applications.

1.1 Contributions

This section introduces the reader to the list of novel contributions that have been made in the following chapters while providing a brief description of each contribution.

- In Chapter 3, I propose a novel architecture and a new protocol design for task offloading and resource management in the mobile cloud. Mobile cloud computing resources are brought closer to the user and put alongside the C-RAN BBU pool. The centralised local controller receives instantaneous monitoring information from both wireless and mobile cloud sides for making resource management decisions. The controller accepts task offloading requests only if there are sufficient resources available to serve the request. If the incoming requests are accepted, adequate resources are allocated to

serve the incoming workload using the proposed protocol. Secondly, I introduce a new mobile task offloading software architecture to complement the scenario and the offloading algorithm that I have presented in Chapter 4. I have carried out the former work, as a part of the Horizon 2020 project iCIRRUS and may be found in publicly available project deliverable D4.3 [21].

- In Chapter 4, I propose a network-aware and energy-efficient mobile compute task offloading algorithm, that considers an offloading scenario where computing resources are placed both on the internet (i.e. clones) as well as next to high bandwidth Wi-Fi access points (i.e data caching enabled cloudlets). The approach is influenced by the observations and analysis made by Kumar, K *et al.* [1], that larger bandwidth networks should be used for compute task offloading. The algorithm takes both available computing and network resources into consideration and decides where to offload by selecting either the Wi-Fi network or the cellular network to offload. The offloading decision depends on the time efficiency (how fast the offloading execution may be finished) and the energy efficiency. For calculating energy consumption when offloading using Wi-Fi and cellular networks, I have employed existing energy models that are formulated based on empirical results.

- I address the question "why vertical scaling is important for delay-constrained applications such as mobile task offloading?", in the Chapter 5. First I conduct an extensive performance analysis on cloud vertical scaling. Then I present the known issues of cloud vertical scaling and explain the importance of such an analysis of cloud vertical scaling delays. Empirical performance data of vertical cloud auto-scaling have been gathered from the prototype presented in Chapter 6, then an analysis of gathered data have been carried out to show trends in virtual CPU, RAM and storage resource scaling delay times, for giving an insight into cloud vertical scaling for designing future auto-scaling algorithms.

- Finally, I prototype the system architecture presented in Chapter 3, in Chapter 6. The implementation includes both the C-RAN side with the software BBU and the mobile cloud side with the Android VMs (clones) for mobile task offloading. A wireless monitoring module and an intuitive dashboard have been implemented as a part of the Resource Monitor in the controller. Authorised UEs may connect to the cellular network and get access

to the services that are made available by the mobile cloud. I have adopted the Thinkair [7] framework into the prototype, to showcase task offloading functionalities in the C-RAN mobile cloud environment. An Android clone that runs on OpenStack-based clouds has been developed, which is publicly available to be downloaded and tested by other researchers [22].

Chapter 2

Literature Review

2.1 Introduction

Mobile Cloud Computing is a much welcomed new paradigm among the research community that brings Mobile Computing and Cloud Computing closer together for increasing computing and energy efficiency of UEs. These advancements have been proven to be achievable through mobile task offloading and by bringing computing resources closer to the UEs. This chapter provides an insight into this exciting new area starting with detailed descriptions of MCC architectures, then moving into a technical review and a comparison of energy models that are used in MCC for calculating power consumption of UEs when making task offloading decisions. Then the state-of-the-art frameworks of task offloading, the key enabling technique of MCC, are introduced highlighting their key technical features as well as their drawbacks.

Once the offloading decisions are made, the tasks are offloaded and executed in the cloud. During this procedure, the resources in the cloud should be efficiently managed and allocated to accommodate incoming requests. Cloud auto scaling is a technique that employs the elasticity, one of the key features of cloud computing, that may be used to dynamically up and down scale CPU, I/O and network resources that are placed at the edge of the network while meeting SLA/SLO requirements. Most influential work in academia and industry has been presented by carefully reviewing the fundamental parameters of the scaling techniques and algorithms. Cloud Radio Access Networks (C-RAN), another promising new area that introduces cloud computing technologies into Radio Access Networks, of which the technical benefits and recent related work have been reviewed for illus-

trating the importance of integrating Mobile Clouds into C-RAN environments.

2.2 Architecture

2.2.1 Mobile Cloud

The use of cloud computing resources by an application in the mobile device, by commissioning processing fragments of it to the cloud is named offloading. *What* to offload to the cloud is determined by partitioning algorithms. *Where* and *when* to offload would be determined by the scheduling algorithm of the mobile application so that only the computation intensive part of the application would be offloaded to the cloud. Subsequently, the output/result of the computation would be returned to the mobile application. An alternative approach is a virtual machine based process/runtime migration. Yang *et al.* [23] have disclosed one of the earliest work that has been done to solve the partitioning problem for mobile data stream applications. The framework enables mobile devices to work collaboratively since cloud resource providers, argue the execution of the mobile application on mobile devices using cloud resources and extend the access to cloud services from mobile applications. The proposed framework considered to be an acyclic data flow graph enclosing components(vertexes) and channels(edges). The component has input ports and output ports. The so-called component is primarily a thread of a process or a process. By using a genetic algorithm, Yang *et al.* disclosed the optimal algorithm that offloads the task elements to the cloud.

CloneCloud [5] is an elastic cloud execution framework for mobile devices. It incorporates automatic static and dynamic analysis of code at runtime before migration/offloading and dynamic profiling to partition applications, without having to make any changes to the code. Code partitioning is at the thread level with fine granularity. This approach is less scalable because bootstrapping is required for every new application and only limited execution environments and input conditions are considered during off- line code preprocessing.

The Thinkair [7] and Cuckoo [8] frameworks provide scalable, online, and method-level code offloading mechanisms with on-demand resource allocation. Cuckoo computational offloading is possible to any resource running a Java VM. Thinkair provides scalability by connecting to a virtual machine with a real smartphone operating system in the cloud for offloading purposes. Thus, it eliminates the environmental, input, and output restrictions placed on offloaded code by other offloading frameworks such as CloneCloud [5]. Thinkair execution controller, de-

cides *whether* a method should be offloaded or not. This decision is based on the information that is collected by hardware, software and network profilers. Java reflection is used if it decides to offload a method to the cloud. After executing the method in the cloud, the result and any modified local state would be sent back to the mobile device. Thinkair exploits parallel- ism on the cloud side by allowing the creation, resumption, and destruction of multiple VMs in the cloud for parallelisation. If a task that is offloaded to the cloud needs to be executed parallelly, Thinkair will divide the tasks into subtask and execute them on secondary VMs with elastic cloud resources. When the parallel execution is done, it sends the results back to the mobile devices. The spawned VMs will be destroyed, or paused temporarily for future use. Thinkair API, allows software developers to annotate easily the remotable methods to be offloaded when it is running with low resources on mobile devices. Also, thinkair does not choose where the cloud clone is hosted. The clone host location may affect the network latency and round trip time, which may have an adverse effect on the energy efficiency. Another drawback is that the clone needs to be created first, and the IP of it needs to be provided to the thinkair client that runs on the mobile devices beforehand, to be able to connect to the clone.

Clone2Clone [24] allows users to create their clone in the cloud or to request CloneDS, the Clone2Clone directory service, to create a clone on their behalf. User based resource cloning in the cloud, a new paradigm in which there is dedicated VM in the cloud for each mobile device. Clones may form secure peer-to-peer networks for content sharing, searching, and distributed code execution. Such systems eliminate the dilemma of having unpredictable and energy inefficient wireless networks, considering that clones in the cloud have stable always-on, high-bandwidth networks. Other mechanisms may be used to offload computationally heavy tasks to the cloud [7] [6] [25] [5], for which the architecture is also limited to networks with low bandwidth and high latency. The paper investigates into deploying the clone in a private network as well as in a public network (Amazon EC2). Before forming the P2P connection, one may need to know the IP of the other party that it would like to connect to. CloneDS is a directory service that maps users to clones and clones to IPs. Such that when needed, clones may get other clones IPs by sending lookup requests to the CloneDS. Clonebox a content sharing platform that is built on top of Clone2Clone, CloneDoc [26] a collaborative document editing tool that is put together with Clone2Clone, CloudShield [27] a worm containment mechanism for mobile devices and cloud, which built on top of Clone2Clone.

CDroid [28] uses another MCC approach in which a secure tunnel is estab-

lished between a mobile device and its cloud clone for all Internet traffic. The cloud clone appears as a local resource for the mobile device. This technique improves web navigation, compression and caching of web pages, and blocking unwanted advertising and virus scanning applications before the installation. A major drawback of this is that it requires an always-on connection to the cloud clone, and all traffic must go through the clone, which is not energy-efficient.

VM based Cloudlet [14] [29] [30] are decentralised and widely distributed internet infrastructure components whose computation and nearby mobile devices may leverage storage resources. These Cloudlets are ideal, clusters of multi-core computers with gigabit internal Ethernet connectivity and a high-bandwidth wireless LAN. Satyanarayanan *et al.* emphasised the necessity of physical proximity of the cloudlet to the wireless AP so that mobile devices benefit from only having to traverse one hop using high-bandwidth, low-latency wireless networks [14]. Moreover, if it is to offload it to the cloud, it has to send data through a WAN using a low-bandwidth, low-latency wireless network. If there are no any nearby Cloudlets, then the mobile application will fall back to the distance cloud, or in worse case scenario, it will solely depend on its local resources.

Kimberley is the current reference for cloudlet implementation. It uses two approaches to introducing the VM state to the infrastructure. 1) The VM is suspended, and its processor, memory, and disk states are transferred to the cloudlet, then the VM resumes at the exact point at which it was suspended. 2) In dynamic VM synthesis, the mobile device delivers a small VM overlay to the cloudlet infrastructure, which already possesses the base VM from which this overlay was derived. Parallelized VMs may be spawned on demand on the cloudlet side to achieve faster execution.

Kimberley does not consider energy utilisation on the mobile side, but resource-heavy operations may drain the battery of a mobile device. Device mobility is one of the major cloudlet issues for mobile users on the move while connected to cloudlets. Kimberley does not consider this factor. Our proposed architecture addresses the aforementioned limitations of Kimberley.

Placement of cloudlets belonging to various service providers next to each other may occupy much space. Collaboration among cloud service providers to form a coalition for transparent customer service could lead to broader cloudlet coverage. Niyato *et al.* [31] used game theory to model coalitions among cloud service providers. The game is modelled so that it is more desirable for the service providers to make alliances rather than to serve customers independently. The payoff for the service providers is always greater if they have formed a coalition. Cloud service providers may deploy strategies so that they may optimise their

capacity expansion tactically, to gain a better payoff, contemplating on what other providers' strategies might be.

The MCC architectures described above only consider the implications of using highly resource-rich computers on the cloud side when delegating tasks. Recent studies have investigated the use of other nearby mobile devices with common interests when offloading computationally heavy tasks. Nguyen *et al.* proved that mobility increases MCC processing capacity [32]. Mobility also affects the performance and resilience of mobile clouds. The inherited mobilisation means that addition of even a small number of highly mobile nodes to a highly localised network may significantly improve the processing capacity and resilience. The main drawback of this approach is that the battery life of current mobile devices is not sufficiently energy-efficient. Cloud clones inhabit one place at one time, with little regard for mobility, in contrast to the highly mobile devices they serve. Hence, a comprehensive investigation of clone placement and clone migration methodologies is required.

2.2.2 Wireless Resource Management

Decoupling of baseband processing components (BBUs) from the base stations and placing them in a centralised location allows sophisticated centralised resource allocation and interference management techniques where the BBUs in a pool may cooperate to improve the cellular network capacity. This particular deployment technique is named C-RAN [11]. C-RAN architecture has been used by several operators and service providers as a cost-efficient way of deploying cellular coverage. It is cost effective, as when its architecture and its components are designed, minimising capital and operational expenditures of the cellular system deployment has been two major objectives.

Cai *et al.*[33] have studied the topology configuration with the rate allocation problem in C-RAN. The aim of the study is to optimise the end-to-end performance of MCC users in next generation wireless networks. The paper also focuses on improving the inaccurate Channel State Information (CSI) problem in C-RAN. To hide the issues in wireless networks from the wireline hosts, a split -TCP proxy has been placed at the edge of the wireless network. The proxy receives wireless segments, stores them locally and forwards them to next second TCP connection.

While by C-RAN [11] proposes an excellent cellular deployment architecture, the wireless resources that are deployed in operator's network has to be managed efficiently for better utilising scarce wireless and baseband computing resources that are situated in operator's network. Software Define Networking (SDN) based

approaches have been proven to provide solutions to control and management related issues in future wireless networks [15]. Bernardos *et al.* [16] propose an SDN-based architecture for efficient wireless service deployments while enabling virtualization. It provides a detailed description of the modules, interfaces (northbound, southbound) and signalling between the elements of the architecture.

Ali-Ahmad *et al.* have proposed an OpenFlow-based SDN architecture [17], which also proposes local controllers for managing resources with a short-term optimisation goals scheduled in, and a regional controller, a logically centralised entity in the network that carries out long-term optimisations of the network. Moreover, the local controllers require detailed and instantaneous data from the network, while the regional controllers only require aggregate data from the network for carrying out dynamic deployment and lifecycle management of local controllers. Ali-Ahmad *et al.* further clarify that LTE interference mitigation, wireless local area network (WLAN) optimisation, LTE access selection and power cycling are some of the control applications that the architecture may be used for.

2.3 Energy Models

Perrucci *et al.*[34] have conducted a survey on energy consumption on components of mobile platforms. Perrucci *et al.* present aspects of energy consumption of various technologies such as Bluetooth, Wifi and cellular networks in great detail. The empirical results of the power consumption when the interfaces are in different states has been presented. A simple energy model for sending an SMS via 2G and 3G networks have been presented. However, the main contribution of the paper is to allow the researchers and application developers, to use the findings for future mobile protocols and applications.

Zhangt *et al.*[35] conducted an experimental study to create an automated power model construction technique that uses built-in battery voltage sensors and knowledge of battery discharge behaviour. First, he investigates into which components of a mobile device contributes more overall power consumption, then exclude the component with less significant impact on the system power consumption. Zhangt *et al.* state that the error of assuming that individual components are independent entities when calculating the power consumption is 6.27%. Hence he assumes that it is less significant to count the power consumption when it is at different cross-products power states. CPU, LCD display, GPS, WiFi, Cellular and audio components have been modelled. The energy model for CPU is based on the utilisation and frequency-voltage settings. LCD display energy model uses

a training program that turns the LCD on and off and changes its brightness between 10 brightness levels. For GPS, the power consumption influences on its state/mode detected satellites detected and their signal strength. The packets transmitted, packets received, uplink data rate and the uplink channel rate have been used to model the WiFi model. This paper does not consider the signal strength of the cellular networks. The model considers the channel state of the cellular network, transmitted and received data rate, and varied two queue sizes. For audio interfaces, the power consumption when activating digital signal processor and/or speaker amplifier have been used.

Balasubramanian *et al.*[36], in the experimental study, the energy model allows the user to empirically predict the future energy consumption of applications by, modelling the energy consumption as a function of both the size of transfer and the time of between successive transfers, of 3G, GSM and WiFi networks. The empirical results have shown that when scan cost is included, WiFi is efficient when doing large sized transfers than 3G and GSM. Granting that WiFi association overhead is comparable to the tail energy of 3G, although the data transfer itself is much more efficient than 3G. When developing the energy model for 3G, the model takes Tail, Ramp and transmission energy into consideration.

ARO [37] [38] is a Radio Resource Control state-based power model. ARO may accurately expose the cross-layer interaction among radio resource channel (RRC) state, transport layer, application layer and the user interaction layer. The typical three RRC states are **IDLE, CELL_DCH** and **CELL_FACH**. However, some UMTS networks support a hibernating state called **CELL_PCH** which is similar to IDLE but the state promotion delay from **CELL_PCH** is shorter, which is explained by the Qian *et al.*. However, Qian *et al.* does not consider the wireless signal strength that affects the energy efficiency of the device.

Experimental studies have shown that communication energy per bit may be as high as 6x when the signal is weak, than when it is strong. Schulman has shown that above statement is empirically correct in their studies. Bartendr *et al.*[39] is a system for energy-aware cellular data scheduling. It predicts the future signal strength and uses developed energy-aware scheduling algorithm for syncing and streaming workloads.

WattsOn [40] an energy estimation tool for mobile app developers to estimate power consumption of their mobile application in development environments. It is built on the power model that measures the power consumption of data transmission and the ”tail” states, at varied signal power strengths for 3G networks. PSM state model [41] has been used for the WiFi power model, which has four states - Deep Sleep (10mW), Light Sleep (120mW), Idle (400mW) and

High (600mW). Other power models that have been used in their experiments are CPU and Display. WattsOn follows an experimental approach when creating their power model. WattsOn may identify energy hungry segments during the app run, and determine which component (display, network or CPU) consumes the most energy.

Rahmati *et al.*[42] designs a context-based network estimation algorithm that uses their developed energy model for wireless data transfers on the mobile phones. Rahmati *et al.* measure the power consumption by reading the voltage drop by intercepting on the interface between the battery and the device. Also, cellular and WiFi connections, the energy model is the addition of connection establishment cost and energy per bit data transfer cost taking the signal strength into consideration. Also, the wifi interface maintenance cost has been taken into account. Rahmati *et al.* have also adopt the concept of transferring data on multiple interfaces to increase the efficiency. This experimental investigation has ignored the TCP, HTTP connection establishment, RTT and TCP slow start.

Sesame[43] is a self-modelling system for energy consumption of mobile systems that is self-constructive and provides a high rate of estimation. The linear regression based energy modelling system is automatically generated by the intelligent battery interface. The CPU frequency has been used as the predictor vector. The mobile operating system, through Advanced Configuration and Power Interface (ACPI), provides a battery driver that may read registers of the battery state. For Linux the may access battery information through a standard file system API. Fuel gauge IC being the main hardware of the smart battery interface, it measures the battery voltage, temperature and current. It may also estimate the remaining battery capacity and total charge that is drawn from the battery. By using the Principal Component Analysis (PCA), it utilises only the top two components with the greatest impact on the consumption as predictors instead of dealing with too many components. The above does reduce not only workload but also guarantees equal or high accuracy and higher sampling rate to other systems.It is observed that the energy model of a low rate based readings on the battery interface is inherently more accurate when there are systematic errors in the readings exists. Sesame has two staged methods to construct the high accurate energy model of a lower rate. At the *Stretching* stage the battery readings have been acquired at a very low-rate model, then at the *Compress* stage it compresses the low-rate model to fit into a high-rate one.

The energy model[44] that has been used by Namboodiri *et al.* take an analytical approach. The power model comprises as the sum of three components. The idle power of the device, the power consumed by the CPU at maximum load

with a CPU specific coefficient and the power consumed by the network interface. The power consumption when the network interface in *active*, *idle*, *sleep* have been modelled. However, this model's granularity is somewhat low, due to lack of varied wireless interface specific RRC state modelling.

A context-aware communication energy model[45] has been proposed, which is a function of the transmission power and congestion level. Due to lack of API support for acquiring transmission power and congestion level, Namboodiri *et al.* have introduced two contexts into the energy model. The received Signal Strength indication (RSSI) as transmission power is a function of it because better signal strength gives less bit-error-rate. The network throughput as congestion level is a function of it.

However, studies on the power consumption of LTE interfaces [46] has shown that it consumes much more energy for the same amount of content, comparing to its predecessors. 3G consumes more energy when it is in dedicated RRC state (CELL_DCH) as it draws a constant power rate for all throughput rates. LTE varies the power drain when in Continuous Reception state, depending on the throughput saving a tremendous amount of energy. Although LTE has got two tail states, Short DRX and Long DRX, which stay at a higher base power rate until it changes to Idle state. The study concludes, because of Long DRX tail, the over energy consumption of LTE is much greater.

2.4 Mobile Task Offloading

The poor battery, storage capacity and the computational power are the main constituents of mobile devices that insinuate the need of cloud technology for portable devices, due to the "portability" of hardware, which is the essence of mobile devices.

Currently, the main complication with mobile devices is the high energy consumption. So one of the advantages of outsourcing the processing power is that it may save energy of the mobile device, depending on other attributes such as wireless bandwidth, the degree of communication intensiveness of the application, processing power of the mobile device and the leased processing power in the cloud. Kumar *et al.* [1] show what affects the power consumption of mobile devices, and if offloading may save battery life of the mobile devices. Offloading should happen only when it is beneficial to the mobile application. Such that, the tasks should be offloaded to the cloud, only if the data transmission cost and the energy that cost on the mobile device while it stays idle when cloud executes

offloaded tasks are less than the cost when the tasks are executed locally.

In two previous studies on offloading algorithms, bandwidth was the only network parameters considered [1] [47], and does not estimate the latency of the wireless network as it affects the QoS of the mobile applications directly when offloading data to the cloud. Furthermore, the Kumar *et al.* [1] assume that the power consumption while sending and receiving data is same but [48] shows that wireless network interfaces have a complex range of behaviours. Hence when designing energy-aware compute offloading techniques or protocols, more factors has to be taken into consideration, such as packet size and the number of broadcasts and point-to-point traffic.

Yonggang Wen *et al.*[49] propose an Energy-Optimal mobile application execution method while in cooperating a cloud clone, a VM that resides in the Cloud. With numerical results, Yonggang Wen *et al.* investigate how the energy of the mobile device may be reduced by optimising the energy use of the application when executing on the mobile device and also when the tasks are offloaded to the clone when it is less efficient to execute it on the mobile device. The results show that the execution policy depends on the application model, which is the input data size of the application with the completion deadline, and the wireless transmission model. The energy of the mobile device is optimised by scheduling the clock frequency via "Dynamic Voltage Scaling"[50] as it has been observed that the clock frequency of the CPU approximately linearly proportional to the voltage supply. Then when the tasks are offloaded to the cloud, the data transmission energy is optimised by optimally changing the data transmission rate via a stochastic wireless channel. Moreover above optimisation methods are formulated as a constrained optimisation problem by introducing an application completion deadline. Although the wireless channel models that are used for the formulation, with only two channel states "good" and "bad", lacks granular knowledge of the wireless channel state.

Lin *et al.* proposed a context-aware algorithm that uses historical log records to determine whether to offload tasks or not [45]. The offloading algorithm considers the user location at a certain time of day when tasks are offloaded for remote execution. A task is offloaded if the energy consumption of the mobile device when the task was previously offloaded is lower than when it was executed locally for the same time of day and the same geographic location. This approach relies on historical records that might not be valid for present conditions and could lead to inaccurate offloading decisions.

Kovachev *et al.* addressed adaptive computation offloading as an optimisation problem using integer liner programming [51]. This approach considers available

memory and CPU and energy usage as the criteria for offloading. The algorithm dynamically chooses what services to offload by solving a new optimisation problem each time parameters such as the available bandwidth and memory change in the model. The offloading decision model of Wu *et al.* takes network unavailability into consideration [52]. The model uses an application partitioning algorithm, and an offloading decision module intelligently decides on whether to offload by considering the network availability for remote execution. The CRoSS algorithm also selects the best host for offloading according to the link cost [53]. The link cost includes both the link failure rate and the bidirectional transmission rate.

At times the user may have connectivity to more than one wireless network, it is possible that the mobile device is in reach of cloudlet at each network. On these occasions, the mobile device may decide whether to use both cloudlets to offload or to offload only to just one cloudlet. For example, there may be some cases where the connected cloudlet is not resourceful enough to finish the offloaded jobs/code while meeting the deadlines. So the mobile device may decide to offload code simultaneously to two different cloudlets but by using two different wireless networks. i.e., using WiFi-based cloudlet and cellular base station based cloudlet synchronously to offload different sets of code. There are some previous work[54] on multi-site offloading, but they only consider finding a solution to the partitioning problem as a *graph partitioning problem*.

2.5 Cloud Resource Management

In the proposed architecture, the computationally intensive tasks are offloaded from the UE to be executed by the computing cloud that is placed alongside the BBU pool. Once, the mobile cloud controller accepts the offloading requests, following a predefined scheduling algorithm/scheme, the task scheduler that resides in the mobile cloud controller may distribute the received tasks to corresponding clones in the cloud. Once a clone receives the scheduled offloaded tasks, it needs to execute the tasks while meeting Service Layer Agreement (SLA)/ Service level Objective requirements. Due to over time changing workload demands (e.g. Slashdot effect [55]), the cloud services should be able to adjust seamlessly to the exigencies exploiting *elasticity*, one of the key characteristics of cloud computing. Auto-scaling includes automated resource scaling techniques for spike workloads, since many web applications face unplanned large fluctuating loads, as elasticity in the cloud allows users to acquire and release resources dynamically in real time. How self-aware auto-scaling may benifit service providers has been

discussed in [56].

In the remaining part of this section, the author categorises existing auto-scaling approaches based on auto-scaling methodologies that have been used in existing literature. Such auto-scaling methodologies may utilise, thresholds, reinforcement learning techniques (e.g. control theory, neural networks) or time series analysis based approaches. Furthermore, the above may also be used by combining more than one method (e.g. thresholds for immediate changes and reinforcement learning for long term changes), as well as on their own, for both reactive and proactive approaches.

The threshold-based approach is classified as a reactive scaling approach; wich is one of widely used auto scaling category among cloud service providers such as Amazon EC2 [57] [58]. Threshold-based auto scaling is a very scalable and straightforward scaling techniquethat may easily manage the amount of resources assigned to an application, while performing auto scaling, by comparing the input demand with predefined scaling rules [59] [60] [61] [62]. When a threshold based scaling approach is used, the targetted VMs will be scaled according to a set of predefined rules. These rules may use one or more performance metric, e.g. CPU load, average number of requests and average response time. Moreover, the essential elements of threshold-based approach are the performance parameters that the rules use. Such performance parameters should be carefully set by the user, or multiple parameters, in cases where logical combinations of parameters are used.

In literature, the most commonly used parameters are, the *upper* and *lower* thresholds. The *upper* defines the highest limit, where when reached an action is a trigger for scaling the VM up. Similarly, the *lower* defines the lowest limit, when reached the VM will be downscaled. Furthermore, each threshold may also have a set time duration to define how long the condition must be met before an action on the targetted VM is triggered (e.g., scale up or scale down). Dutreilh *et al.* also propose that aforementioned thresholds should be correctly selected and carefully fine tuned to avoid such *oscillations* in cloud systems. Therefore, one may also have a cooldown period, inertia or calm down period; that is defined per each threshold, a time during which no scaling actions should be carried out, to avoid VM scaling *oscillations* in the system. The cooldown time prevents continuously allocating resources while the new VMs are created (horizontal scaling) or new resources are added to existing VMs (vertical scaling), or continuously reacting to fluctuating trends of the performance metrics depriving extra cloud resources and energy.

The most widely used performance metrics are CPU load of the VMs, the input request rate and the application response time. The threshold rules are typically

based on one performance metric (e.g. application response time), or at most two. Both Han *et al.* [61] and Dutreilh *et al.* [59] have used the average response time of the application when making auto-scaling decisions. Despite the convention, Hasan *et al.* have used three types of performance metrics from three different domains. Moreover, while other auto scaling techniques use each performance metric in isolation, Hasan *et al.* have considered a correlation of performance metrics in multiple domains. For example, instead of VMs are scaled based on CPU utilisation independently, the VMs are scaled, when the CPU load and the response time have increased above their thresholds.

Mao *et al.* [63] consider both user performance requirements and budget considers of the user when setting autoscaling rules. The auto scaling strategy dynamically allocates and deallocates VMs and scheduling tasks on the most cost-efficient instance. The solution chooses a scheduling plan which determines the instance type for each running task at a given time. Then a scaling scheme, in which it determines the number of instances of each instance type to be allocated for the task.

Although current work in literature may select one or more performance metric, often they choose two thresholds per performance metric at most. Contrasting from others work, Hasan *et al.* [62] have employed four thresholds per one performance metric for making scaling decisions, namely they are *ThrU*, *ThrbU*, *ThroL*, *ThrL*. *ThrU* is the highest upper bound, and *ThrbU* is the second upper bound that is below the *ThrU*. Likewise, *ThrL* is the lowest lower bound and *ThroL* is the second lower bound which is slightly above the *ThrL*. Independent and dissimilar time durations have been used for both *ThrU/ThrL* and *ThrbU/ThroL* threshold sets. The Hasan *et al.* have shown that the proposed four parameter approach is better when it comes to tracking trends of the selected performance metric. It has then proven to help make finer auto-scaling decisions, than when used only the common *upper* and *lower* thresholds.

As an extend to the conventional reactive rule-based approach to threshold based auto scaling policies, the auto-scaling algorithm of RightScale [64] introduces a democratic voting system, such that scale up and down actions are executed, only if the majority of the VMs vote to do so. Each VM make their decisions to scale up or down based on a set of predefined rules. Similarly to the conventional approach, once a scaling action has been carried out, a cool down period (recommended 15-minute time slot) will be spent, where no scaling actions are allowed.

Threshold-based auto scaling combined with democratic voting also adopted by [65] [66] [67] [61]. Moreover, Chieu *et al.* [67] use some active application

sessions as the performance metric, and threshold-based rules are created accordingly. In their later work, the former work has been extended by proposing a voting based auto scaling method. If the number of active sessions of all the instances is above the *upper* threshold, then a new instance is instantiated. If the number of active session of all the instances are below the *lower* threshold, then at least one instance that has no active sessions will be terminated.

Kupferman *et al.* [66] compare RightScales voting based auto scaling approach with other algorithms. It concludes that the RightScale's algorithm is sensitive to the variations/characteristics of the workload because the threshold values are set up manually by the users. This dependability also means, at a given time the values are set to conform only to one type of workload pattern and may underperform for workloads that have different characteristics other than for the one that they are set. This issue has been further studied in [65], and it proposes an approach for overcoming the issue. Simmons *et al.* use a strategy tree for switching between alternative strategies hierarchically over time, by evaluating the deployed policy set. Simmons *et al.* create "elasticity policies" to accustom to input workloads that have different characteristics. I.e., the strategy tree would switch between three custom policies based on the observed workload trends. All above-scaling approaches kill instances once the *lower* thresholds are reached, but note that most common cloud pricing models charge hourly, hence may experience cost inefficiencies, as when a VM is terminated early, the client will still be charged the full hour price. *Smart kill* [66] saves costs by halting instance terminating (killing) actions until it completes the charged hour, even though the VM load is low.

Another threshold based auto scaling scheme [68] has been designed for scaling resources of PaaS cloud services while meeting application performance needs and SLAs. The algorithm takes, CPU, memory and heap usage of applications when making scaling decisions. The implementation is designed for the IBM Bluemix PaaS cloud service.

All mentioned above threshold parameters are statically set, and the user interventions are required for tuning the parameter values to match the rapidly changing workload patterns. Lim *et al.* [69] dynamically allocates threshold parameters with the help of an integral controller. Beloglazov *et al.* [70] propose an adaptive threshold-based approach, where the threshold values are dynamically adjusted based on the CPU utilisation of each VM. The CPU utilisations of all VMs which are allocated to a host are collected. Then Beloglazov *et al.* determine the probability distribution of the host CPU utilisation (sum of CPU utilisations of all hosted VM), from which they calculate an interval of the CPU utilisation that is reached

with a low probability. Thereafter, the thresholds are set dynamically based on the calculated CPU utilisation interval.

Similarly, Liao *et al.* [71] focuse on changing the thresholds dynamically while increasing application response time and reducing VM running time and error rate. The thresholds are dynamically adjusted depending on loads of all other virtual machines serving the same application. If the workloads of over 50% of all VMs are raising, then the upper threshold is lowered proportionally by 0.5%. For fast resource dissemination to the application. Similarly, if the workloads of more than 50% of the VMs is high, while the overall workload still alleviated, the upper threshold is raised moderately to avoid large oscillations in the upper limit, resulting in lower cost. This strategy reduces the oscillations of the auto scaling system, albeit it allocates more resources to the application when the overall workload of the application is high, while it allocates less when the overall workload of the application is little.

Contrary to threshold based approach, Reinforcement Learning (RL) is another category of techniques that have been employed to solve the auto scaling problem in the cloud. It may also be used to make auto-scaling decisions in real-time by performing actions (e.g. scale up and scale down), depending on the current state of the environment (e.g. input workload and other performance metrics) while trying to maximise a reward (e.g. reduced energy, application response time etc.).

Dutreilh *et al.* [72] scale cloud resources horizontally based on the average application response time, while a realistic *upper* bound value has been chosen based on traces gathered by experimental results. Whereas, Jia *et al.* [73] consider the CPU and memory usage per each VM as the state of the environment when performing vertical scaling. One may define adding, removing or maintaining a set number of VMs, for horizontal scaling, or adding, eliminating the amount of allocated CPU and memory resources of a VM, when vertical scaling, as other possible actions that may be used for RL. The reward function may take, the resource cost, the cost of renting the VMs for the given period and the used network bandwidth, and/or the SLA/SLO violation costs into account [74] [72] [73].

Although, there have been many works on the use of RL in cloud auto-scaling, many issues of the approach have been reported. Even though learning seems a promising technique, it comes with a price. The price being a long time that it takes for the learning period, also known as the training period. To solve this issue with the initial bad performance, [72] updates the value for all states at each iteration, using an initial approximation for the Q-function. This technique has

shown to shorten some steps to convergence to an optimal solution. While [73] uses a policy that visits many states at each step, Barrett *et al.* [74] employ parallel learning agents, for reducing the initial training period. The parallel agents learn the values of non-visited states from neighbouring agents, without needing to visit each and every state and action. Moreover, alternate methods could deploy, where the RL models are trained offline on pre-collected data, for improving the poor inherited performances of online training procedures.

Neural networks, support vector machines, regression splines and trees, are some of many solutions to the *curse of dimensionality problem*, the large state-space problem, which is a known issue in RL. The existing work on neural networks [75] [73], takes state and action pairs as input, then outputs the approximated outcome, while also predicting the values for the states which are not visited. Rather than using lookup tables that are used in above solutions, nonlinear function approximators may be used, replacing lookup tables.

Queuing theory has been used to solve auto scaling problems in many instances in literature. A model [76] has been used for estimating the necessary resource required for a given input load and for the mean response time of requests, for a variable number of servers. Afterwards, similar types of information have been used to solve optimisation problems [77] or as a predictive controller [76]. Multiple queues have been used in number of occasions in literature, when formulating multi-server and single application (single application tier) scenarios [77] [75], due to limitations that other works have shown, which use single queuing models. However, some work have adopted queuing networks to formulate multi-tier applications [78] [79]. Furthermore, Tesauro *et al.* [75] considered both open and closed-loop models, by using Mean Value Analysis (MVA) [80] formulation when modeling mean response time of an application. By using a real trace of an e-commerce system, Villela *et al.* [77] model the arrival processes to a cloud application. The analysis of the trace has shown that it follows a Poisson process.

Queuing networks have also been used for multi-tier applications by considering queue per server, and queue per tier, in [78] and [79] respectively. In [78] Urgaonkar *et al.* predict the peak workload and the number of servers needed on each tier to satisfy the demand. Histograms have been used for determining prediction, and they further improve the solution by using reactive methods. Due to provisioning is done for peak loads, it results in under utilisation of resources. In [79] employs a closed system with a network of queues for a finite number of users, and the model is solved by using the MVA [80].

The previous work requires various information as input to the queuing model, such as the number of requests, transactions, service time etc. Such information

may be obtained by online monitoring [78] or by using estimating techniques such as regression based approximation [79].

Harold *et al.* [69] [81] does horizontal scaling by adjusting the number of active VMs, by taking the average CPU as the performance metric using an integral controller. Similarly, [82] uses a Proportional-Integral (PI) controller for controlling the resources based on the execution progress of batch jobs. In the formula of the PI controller, the parameter for the ratio of output response to the error in the previous cycle, and the throttling parameter that determines the accumulated errors in previous cycles may manually set based on trial and error [69] or using a theoretical model [82]. There have been many works on adaptive controllers [76] [83] [84] [85].

The Multiple Input Multiple Output (MIMO) adaptive controller technique [83] uses second-order combined moving average and autoregression techniques to model the non-linear and time-varying relationship of normalised resource allocation performance. Whereas, [76] combines proactive and adaptive controller approaches when carrying out scaling down actions with dynamic gain parameters based on input workload while using a reactive controller for scaling up. Kalyvianaki *et al.* [86] determine the CPU allocation of VMs by using a Single Input Single Output (SISO) and a MIMO controller that uses Kalman filters [87]. Bodik *et al.* [85] also use a smoothing technique when conducting auto scaling.

Authors have used Fuzzy models by providing workload values and required scaling values, then mapped into fuzzy sets, for mapping the right resource amount for a given workload (fuzzification). Although often such fuzzy sets are fixed and non-adaptive, [88] and [89] propose fuzzy controllers that repeatedly update their fuzzy model based on the online monitored information. The former has applied this method to estimate the required CPU load for the input load. The latter follows with the focus on the database tier of applications while they apply the same adaptive fuzzy controller technique. However, they ignore the possibilities of sudden fluctuations of the workload levels that could occur in future/next time slots. Wang *et al.* [90] propose a fuzzy predictive controller, combining fuzzy logic and control theory. Similarly, Lama *et al.* [91] combine neural networks and fuzzy controllers that may adapt its parameters through online learning.

There have been many works on auto scaling through time series analysis. Although some used a simple moving average on the workload to capture the trend when performing auto scaling [92], this technique has proven to be poor. Conversely, some works have used a moving average for removing noise from time series workload traces [82] [69]. Similarly, Huang *et al.* [93] employ double exponential smoothing in their resource prediction model, and compares it with

simple mean and Weighted Moving Average (WMA), showing the superiority of exponential smoothing in comparison to others, as it takes both the history and current data into account when making predictions. However, this is further proven by [94], by using quadratic exponential smoothing against ClarkNet [95] and World Cup 98, with a small amount of error.

Khan *et al.* [96] propose a model that is based on features and core entities of existing auto scaling techniques. A concise review of existing auto scaling systems has been carried out for identifying common components and operations, the parameters of which later used for designing the model. The gathered real time series data is used to calibrate the variables of the developed model for enabling proactive estimations of the responsiveness of the auto-scaling operations. The model has been evaluated by using the Google cluster trace [97].

Checn *et al.* [98] propose a hybrid auto-scaling algorithm which uses statistical methodologies such as AutoregressiveMoving-Average model (ARMA), the Autoregressive model (AR), the Exponential Smoothing (ES) model, the Trend-Adjusted Exponential Smoothing (TAES) model, the Moving Average (MA) model, and the Nave model. Checn *et al.* have used Mean Absolute Error (MAE), Mean Square Error (MSE) and Mean Absolute Percent Error (MAPE) when evaluating the prediction method. They also have shown that Autoregressivemoving-average model, the Autoregressive model and the Exponential Smoothing model have performed better than the others. Reactive rules are used to correct incorrect predictions that are made by the proactive model based on historical data.

An auto-scaling prediction system [99] for cloud resource provisioning that follows a hypothesis of, prediction accuracy may be increased by time-series prediction algorithms based on performance patterns. A comparison of accuracy levels of time-series prediction algorithms have been carried out for different performance patterns, to prove the hypothesis. An evaluation is conducted on Amazon EC2 and the results show that the accuracy of Support Vector Machine (SVM) and Neural Networks (NN) time-series prediction techniques depends on the incoming workload pattern. Nikravesh *et al.* propose a self-adaptive prediction system, which chooses the most suitable prediction method dynamically based on the workload profile.

A prediction-based proactive approach [100] that uses the data that is being produced by itself to make auto-scaling decisions. The autoscaling operations are triggered according to the changes in the public sentiments about soccer players that have identified to affect the cloud workload, which happens just before workload bursts. The proposed algorithm is able to predict peaks and prevent SLA violations.

Roy *et al.* [101] use second order Auto-Regressive Moving Average (ARMA) for predicting the workload levels. They first predict the value based on the last three observations; it is then used to estimate the response time. However, there have been number of work that have used autoregression for auto-scaling in literature [66] [102] [92] [103] [104]. When Kupferman *et al.* [66] applied first order auto regression to predict the request rate. Then they found that the performance of which was largely influenced by the user defined parameters. The history window (the monitoring window) resolved the degree of sensitivity of the algorithm to local trends. Conversely, the size of the adaptation window of the algorithm determined how far into the future the prediction extends. The aforementioned history window values have been then fed into a neural network by Sadeka *et al.* [105]. Similarly, in [66] [85] [105] history window values have been used inconjunction with multiple linear regression equations. Kupferman *et al.* [66] further explain the necessity of balancing the size of each sample in the window to avoid over-reactions, while maintaining a good level of sensitivity to workload fluctuations. However, using a mean of all predictions after regressing over windows of different sizes has proven to improve the performance. Also [105] was able to obtain better results when more than one past value for prediction has been used.

Although, most of above are proactive solutions, time series data may also be used with reactive approaches. A combined regression and reactive rules-based approach have been introduced for scaling up by Iqbal *et al.* [106]. For scaling down, they use a regression-based approach. Regression of degree two has been used to calculate the number of application-tier and database-tier instances, after a fixed number of intervals in which the response time is satisfactory. While time series forecasting has been the focus of a vast number of papers in literature, the interest of some work diverted towards identifying patterns in the input workloads [107] [108] [92] [109]. A simple approach is based on histograms; that consider the mean of the distribution [103], or the mean of the bin with highest frequency [92]. Gong *et al.* [92] have provided a comparison of such pattern recognition techniques, while showing that Fast Fourier Transform (FFT) based approach perform well when identifying patterns in workload traces of for CPU, I/O, memory and network resources. The approach proposed by Caron *et al.* [108] [107] has proven that the number of parameters in the algorithm, affect the performance of the algorithm and the time needed to iterate through the past traces.

Amazon EC2 provides an auction-like service to buy VMs while occupying idle capacity, i.e., spot instances. It is shown to be cost-effective to employ spot instances for fault-tolerant applications, they are not recommended to be used for time critical and fault-intolerant applications, as the VMs may be terminated by

the provider when the market price increase than the bid price. The proposed method by [110] uses a spot mode which utilises spot instances and on-demand mode which employ traditional on-demand VMs interchangeably.

A broker based approach has been proposed [111], for multi-cloud resource orchestration and allocation. Based on various parameters such as QoS, policies, regulations and cost, it allows workload provisioning and resource allocation over heterogeneous clouds service providers. It is also able to carry out auto scaling of the allocated resources based on the workload patterns. Biran *et al.* have developed a prototype to demonstrate the benefits of the proposed system. The implemented auto-scaling algorithm employs a threshold rule based scaling methodology.

Coninck *et al.* [112] investigate how to automate the dynamic provisioning of resources on private and public clouds to increase the number of successful requests that meet their deadlines. The jobs and the resources needed for the job to complete are scheduled over time on both private and public clouds. A knowledge model is created to accurately predict the behaviour of future jobs of the same type, by using past execution times of the jobs.

HAVEN system [113] approach the scaling problem by combining load balancing and cloud auto-scaling. Load balancing is carried out in the network Open Systems Interconnection model (OSI) layer 4, directing incoming TCP connections to a pool of cloud servers while balancing the load. HAVEN follows a threshold based auto scaling approach that uses a calculated score based on the bandwidth, CPU and memory utilisations of the virtual servers, as the performance metric. Poddar *et al.* highlight the importance of designing auto scaling system in conjunction with the load balancer, as when horizontal scaling, the load balancer needs to be notified, to add the new VM to the load balancer when scaling up, and to remove a VM from the load balancer when scaling down.

A proactive auto scaling scheme has been proposed [114], and that predicts the number of web requests, for which an optimal cloud resource demand is calculated with cost-latency trade-off in mind. Depending on the estimated resource demand, the scheme makes a resource scaling decision. The algorithm includes an operation, called No Operation (NOP) when decided no operations are carried out. The algorithm has been evaluated on Amazon cloud platform using real web log traces. The proposed scheme proven to achieve resource scaling with optimal cost-latency trade-off, with low SLA violations.

Campos *et al.* [115] present a Markov chain model for parametric sensitivity analysis. The analysis is to help system administrators set auto scaling configurations efficiently in private clouds while prioritising certain parameters.

The security issues of auto-scaling techniques have been studied by Mor *et al.* [116], in which they demonstrate the attacks and their effects by emulating an Economic Denial of Sustainability (EDOS) [117] attack, on Amazon EC2 virtual machines. Mor *et al.* have further stressed that such attacks impact the cost of service and the response time of standard users.

Ahn *et al.* [118] carry out the first study that approaches auto scaling problem for real-time healthcare applications, that process and compress gathered signals in the cloud, once received from sensors of a health care system. However, the focuses on existing work on auto scaling algorithms have been only for delay tolerant, and best effort applications. Moreover, existing work does not take the VM creation time delay into consideration in their auto-scaling algorithms. Ahn *et al.* incorporate a prediction mechanisms to predict future workloads and create a hierarchy of child VMs, prior to workload fluctuations. Once new child VMs are created, then extra workloads are shifted to new VMs. The system predicts if the workloads may be served while meeting the task deadlines since in real-time systems meeting the task deadlines is more important than saving computing resources. The proposed mechanism has been evaluated using Amazon EC2 and proven to be effective.

All previous work on auto scaling techniques is designed keeping only the best effort cloud applications in mind. In contrary, with the recent advancements Ahn *et al.* [118] utilises cloud computing for mobile Real-Time Applications (RTA). To handle time sensitive and mission critical nature of mobile medical systems, Ahn *et al.* propose a hierarchical horizontal auto-scaling architecture with a proactive algorithm based on future medical sensor workloads, for achieving seamless auto scaling while meeting critical hard deadlines. This work is the first to address the auto scaling problem for sensor-based RTAs that also processes deadline-critical real-time data generated by medical devices. Likewise, later on other works have adressed auto scaling issues in RTAs [119] [120] [121] [122] [123] providing solutions for horizontal auto-scaling using various reactive and proactive techniques.

Chapter 3

Architecture and Protocol

3.1 Introduction

The past few years have witnessed a rapid shift in computing from the desktop to the cloud. To keep pace with advances in both wireless network technologies and mobile smartphones, there is an increasing need for the provision of cloud services to mobile users via mobile wireless networks. This new research field is called mobile cloud computing (MCC) [124] [23] [5] [7] [125] [126]. As mobile devices, even modern smartphones, are constrained in size and weight, their resources for computation and communication are limited compared to their desktop counterparts [127]. Therefore, it is beneficial to offload heavy mobile applications to more powerful machines in the cloud. Computing and service delivery are possible because of the advanced sensors built into most mobile phones currently on the market; these sensors include accelerometers, magnetometers, GPS chips, gyroscopes, and pressure sensors. The more sensors a device has, the more data need to be analysed in various domains at the same time, which accentuates the need for more computational power. One critical issue in MCC is how battery power for mobile devices may be spared [128]. One effective approach is to offload some tasks from the mobile device to a remote cloud server for execution. This may also potentially reduce the task execution time because of the power of cloud servers. Kumar and Lu investigated the power consumption of mobile devices, including whether offloading may increase battery life [1]. Offloading should only occur when it is beneficial to the mobile application. Thus, tasks should only be offloaded to the cloud if the sum of the data transmission cost and the energy cost is smaller than when the tasks are executed locally on the mobile device.

Providing a reliable infrastructure for delivering network services to the end users has been the primary purpose of the Mobile Network Operators (MNOs) today. Can the operators help to improve the user experience and the services that are delivered to the users, except making the content delivery faster and reliable without making many changes to the existing infrastructure? Furthermore, operators are interested providing customers with additional services and performance enhancements other than a faster mobile network connection for attracting more customers. Mobile cloud computing enables the network operator to go beyond just a pipe provider. They may have a platform that could allow other service providers to build up services on top of it.

In this chapter, the author designs a novel system architecture for mobile clouds with a local controller that manages newly introduced computing resources in mobile operators network, as shown in Figure 3.3. The controller takes a similar role to previously proposed SDN like controllers [16] [17], although the southbound interfaces interact with the mobile cloud and the C-RAN wireless infrastructures that are co-located. This enables the resource management algorithms to manage both computing and communication resources cooperatively/jointly. Moreover, this chapter presents a novel design of an intra-cloud protocol for computing offloading and for allocating/managing resources in mobile clouds.

3.2 System Architecture

The author introduces a mobile cloud in to future radio access networks as depicted in Figure 3.1, specifically for the C-RAN deployment approach. The mobile cloud is placed next to the pool of Base Band Units (BBU). One of the design goals of such approach is that to enable joint and cooperative resource management between the BBU pool and the mobile cloud. Therefore, a centralised controller for making resource management decisions based on both computing and communication resources has been introduced. The Section 3.3 describes the components of the mobile cloud controller in detail. The author also proposes that all inbound and outbound traffic would be routed through the mobile cloud so that the users may better benefit from resources that the mobile cloud has. The dotted lines in Figure 3.1 show the connectivity from the mobile cloud controller to the BBU pool and the mobile cloud.

In the conventional cloud service scenarios, one VM may serve many users i.e., different VMs may host various services. However, the notion of Clone that is introduced in [5] [24] [26], where one VM is dedicated only for one user, has

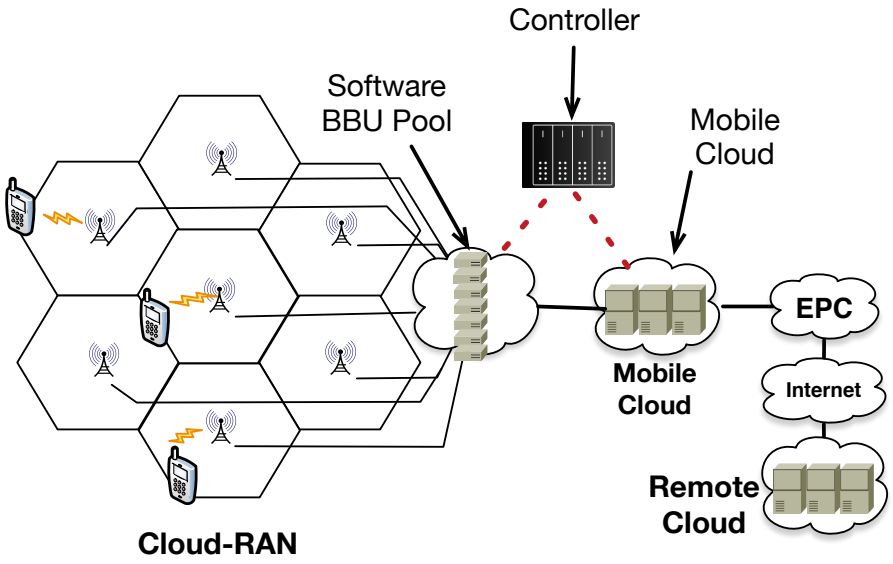

Figure 3.1: Overview of the architecture, showing the interaction between, Mobile device and mobile cloud and C-RAN.

been extended in this work. Therefore, the user data and application data may be stored locally on each user's clone. Moreover, in Section 3.4 multi-user clone to clone communication scenarios have been introduced. In the rest of the thesis, the author refers to user's dedicated VM in the mobile cloud as "Clone".

3.2.1 Benefits

Benefits for the User

There are multiple aspects of the proposed architecture that benefit the end-user. Due to centralised resource management and newly introduced computing resources in the operator's network, the users do not have to request or subscribe to multiple services at the same time. For example, a mobile network operator may provide computational resources with its mobile network service for the subscribers.

The users may observe that their mobile devices may perform computational tasks much faster when it is connected, than when it was not (when computationally intensive tasks are offloaded as described in Section 3.5). Moreover, the mobile applications and the operating systems may become faster and efficient

while also increasing the Quality of Experience (QoE). The application developers will take network parameters into account when developing applications for devices in mobile cloud computing environments. For executing applications in a mobile computing environment, network information will be important due to the unavoidable code transferring and receiving time. Henceforth, the applications will be aware of the network parameters such as bandwidth, delay, signal strength, the location in the network that computational resources are placed in, and the size of the computing resource.

As a case in point, one may imagine that there are two mobile network operators, which have the same network bandwidth and latency when accessing the internet, called A and B. The only difference between the two is, A is a conventional MNO and does not have proposed architecture deployed, but B has deployed the proposed architecture. If there is a mobile user that may connect to both networks, when the user connects to A, he may get access to the internet just like any other network that exists in present day. Although, when the user connects to B, he is not only able to surf the internet faster, but in the background the computing capacity of his mobile device has also elevated, consequently improving the user experience. Such increase of computing capacity of the connected devices will not only help perform networking operations even faster but most importantly it will also improve the performance of the mobile operating systems and other installed applications, as a result of task offloading and centralised resource management.

Benefits for the Service Provider

Network and computational resources will be jointly and cooperatively managed to optimise the performance of networks and computing services. As an instance, one approach is to jointly allocate computation and network resources, while exploiting computing and network transmission trade-off shown in Equation 3.1, when computational tasks are offloaded to the network [1] . Where, T is the time takes to execute a task, while $D - S$ amount of data is transferred with a r bitrate to the cloud, for executing F number of instructions in f speed. The set of data D is required to execute the given task, while some parts of that data S might already exist in the network (on the clone). The objective of the resource allocation algorithm may vary; more computational resources f may be allocated for the benefit of the network or more network resources r for the benefit of the computational resources in the network.

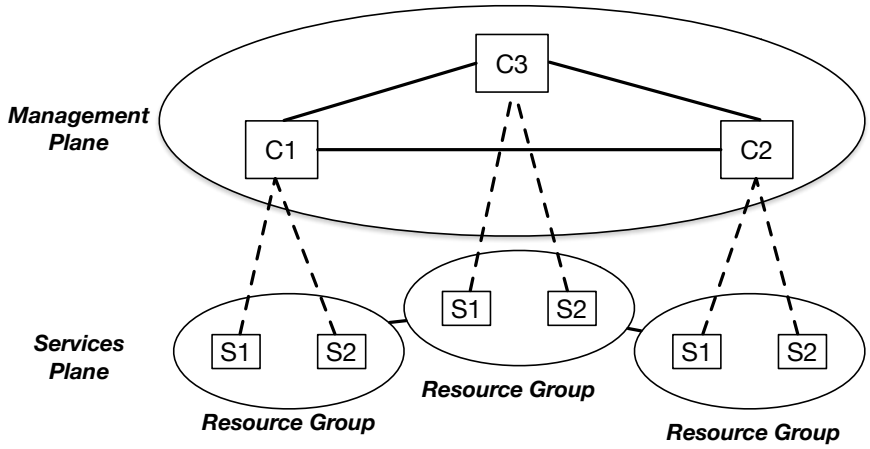

Figure 3.2: Management and service planes

$$T = \frac{D - S}{r} + \frac{F}{f} \tag{3.1}$$

Accordingly, the subscribers' QoE will be increased, as a result of utilised computing resources made available by the network. Subscribers will notice a significant improvement, in application performance of their devices locally, while they are connected to the operator's network. Online gaming is one application that could benefit from the proposed architecture.

The newly introduced computing resources may introduce new types of services. Such improved services and new types of services may lead to new price plans for the users. There, also will be a change in the current business model of the network service providers. Instead of the subscribers having to subscribe for computing services from different service providers, the computational resources that are built into the network will allow the network service providers to offer computing services to the end-users. Those above will lead to better price plans that increase revenue for the service providers while also increasing the user experience for the end users that may attract more customers.

3.3 Mobile Cloud Controller

Mobile cloud service providers may offer one or more services for the end users, which will primarily be computing and communication services. The services

and resources will be managed and tailored to match subscription plans of the users, while also managing resources efficiently among all the system-wide users. As shown in Figure 3.1, for scenarios where computing resources are distributed across multiple regions in future networks, the controllers will manage services and resources that are meant to be served for a particular group of users, whether if its users within a specific geographical are or a specific organisation.

As shown in the Figure 3.2, each controllers C1 through C3, manages resource groups (S1, S2) that each controller governs. For example, controller C1 manages two resource types S1 and S2. In the mobile cloud computing scenario, they could be computing and communication resources. The management plane consists of connected neighbouring controllers for supporting mobility, and the connectivity between other controllers via the backhaul networks, providing scalability (access to larger computing resources in core cloud). The service plane consists of networks, of various types of resources that interconnect, for cooperatively or/and jointly managing and sharing resources. Furthermore, the author envisions that the mobile cloud, base station and the controller will be integrated into one entity in future. For vertical interactions between the local controller C1 and its resources and services (i.e. S1, S2) intra-cloud protocols has to be designed. Likewise, horizontal communication scenarios are two-fold. In the services plane, for intercommunications of the neighbour mobile clouds, and for intercommunications between the computing resources at the edge of the network and computing clouds in core networks or in the Internet (the scenario in Chapter 4), needs Inter-cloud protocols. In the management plane, for intercommunications of the neighbouring local controllers, and for intercommunications of the local controllers and central controllers, inter-controller protocols have to be designed.

The BBU pool is connected to the mobile cloud with a high bandwidth, low latency transport network. The mobile cloud consists of multiple Physical Machines (PMs) that host clones. The PMs are also interconnected with a high bandwidth, low latency networks. The newly introduced computing resources at the edge of the wireless access networks require global knowledge of both communication and computing resources for efficient utilisation of the network and communication resources. A mobile cloud controller is introduced for continuous resource monitoring and management of resources within one base station. The controllers of neighbouring cells/base stations will be interconnected for efficient inter-cell and inter-cloud resource management. In specific scenarios, the mobile cloud controller may also be connected to a central cloud where resources will be leased when the mobile cloud computing resources are not sufficient (i.e. the scenario in Chapter 4), although designing such iter-cloud protocols is not within the scope

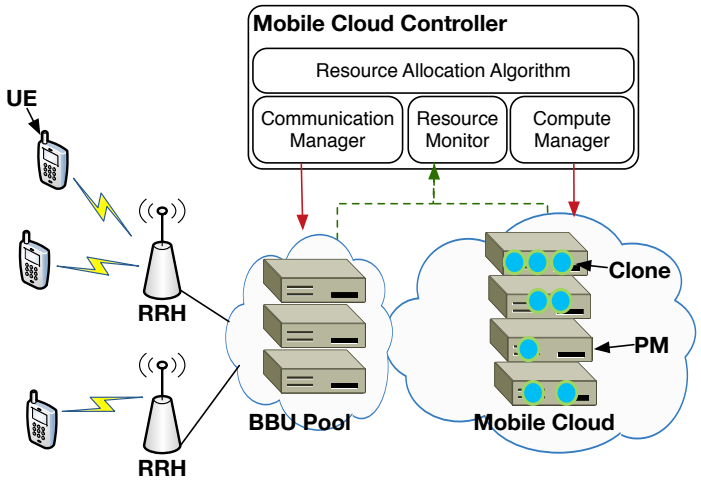

Figure 3.3: Mobile Cloud Controller

of this chapter.

The mobile cloud controller is used for combined communication and computing service provisioning, SLA management, service lifecycle management and monitoring in both communication and computing sides. In Figure 3.3 the dashed lines represent monitoring data flows, and the resource allocation control streams are shown in colour red. The resource monitoring and analytical modules in the controller receive the monitoring information of both radio and clone resources. The communication manager performs radio resource management tasks on the mobile network, while the compute manager manages the mobile cloud. However, the work on this thesis primarily focus on the Compute Manager as seen in Chapter 5 and the Resource Monitor as may be seen in Chapter 6. Therefore, the proposed protocol also has been designed predominantly around the compute manager, and additional functional designs and implementations will be carried out in future work.

3.3.1 Mobile Device

The UE is the initiator of the offloading sessions and acts upon control messages that it gets from the controller and the mobile cloud during offloading sessions. One offloading session may include more than one offloading tasks. An offloading session is identified by the offload start request that is sent by the Offload Initiator, and once the UE finishes offloading it sends another control message to the

controller indicating that it has finished offloading. Such messages will be sent by the offloading framework that resides in the UE after it has decided to offload to the mobile cloud. Once, the controller has created a clone for the UE, the UE connects to the offloading framework in the clone for offloading the code. The clone receives code from the server side of the offloading framework, executes them in the clone and sends the results back. If the code is not offloaded to the mobile cloud, it will be executed locally on the mobile device by the the local mobile code executer.

3.3.2 BBU Pool

It is assumed that the controller is able to acquire and send real-time monitoring information from the base station via a dedicated high bandwidth network. The BBUs and the mobile cloud accept control messages that it receives from its controller for allocating resources, then subsequent error or acknowledgement messages will be sent back.

3.3.3 Mobile Cloud

The mobile cloud is composed of powerful computing servers that are managed by a cloud computing framework. It is able to manage and dynamically deploy virtual and physical resources within the cloud. Virtual machines will be created as they are demanded by the controller, virtual networks for interconnecting virtual machines and for making them available to their assigned UEs and to the EPC (Evolved Packet Core) via the E-UTRAN. All aforementioned resource allocation tasks are carried out by the resource managers in the controller, by utilising Application Programming Interfaces (API). The Resource Monitor is able to acquire real-time utilisation information of the cloud.

3.4 The clone and communication offloading

Device to device communication enables direct communication between two devices in the same vicinity, using existing cellular spectrum. However, with a clone in place all communications does not have to happen directly from one device to the other. Since, all users have their own clones, when one has to send data to another, the senders clone may transfer the data to the receivers clone assuming that

sender has already got the sending data in the clone. This is turning D2D communication to D2C (Device to Clone). An example of this could be mobile social networks where users (i.e., UE) store their user-generated content (UGC) on their clones rather than on a centralised server (like Facebook). Current social networks are centralised which has data privacy concerns. Mobile social networks enabled by clones store content on content owners clone. When it comes to sharing the stored data between users using D2D communication, the clones may come into play and convert D2D into a D2C communication. The author considers three scenarios below.

1. One sender - One receiver.

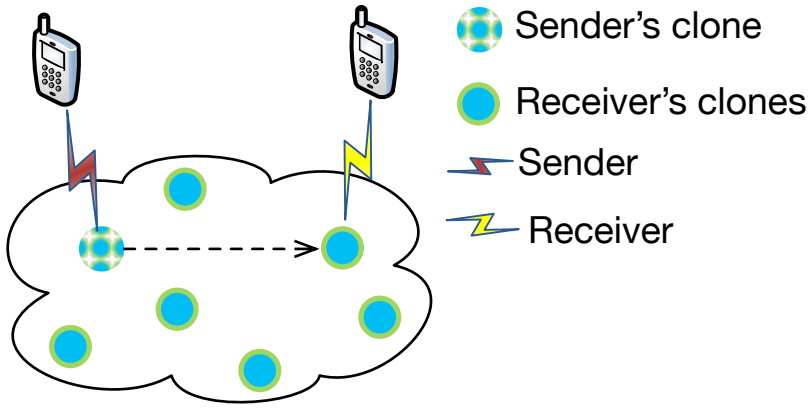

Figure 3.4: One sender - One receiver

In this scenario, user A (sender) may upload its some data (content) that is intended to be shared with others in the future to it's dedicated clone. User B (receiver) is interested in what user A has got, so he goes ahead and requests user A's content. Once a connection has been established between the user A and B for sharing requested data, user A sends the requested content to user B's clone. Finally, the user B receives the requested content from its clone. When considering this scenario, there are not many benefits for the network operators nor the users. The reason being is that similarly to the D2D communication scenario wireless spectrum has been used ones to upload the data and also again to download the data at the receiving end. It is only when the sender needs to send the same content more than one time the cellular spectrum may be saved. Because the receiver's clone may get

the content from the sender's clone without the sender having to re-upload content to his clone, assuming that the sending clone has still got all content still stored by the time the second receiving user requests for content as shown in Figure 3.4.

2. One sender - many receives same content to their own clone at the same time.

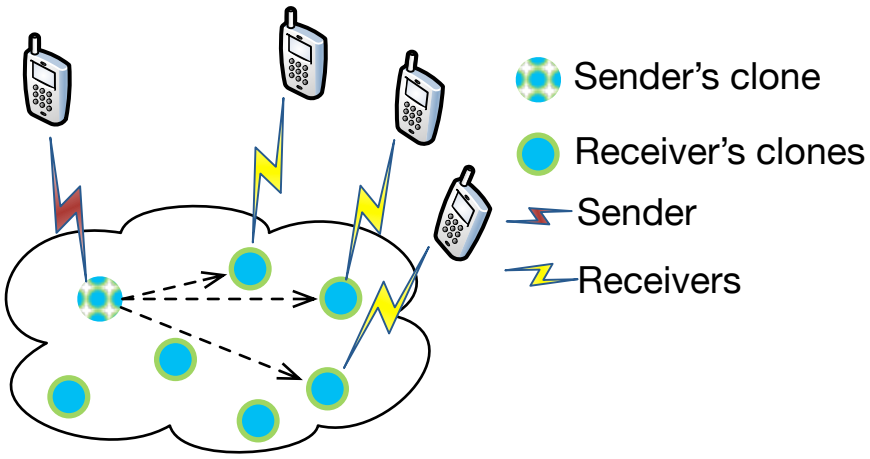

Figure 3.5: One sender - many receives same content to their own clone at the same time.

The user A (sender) A has got something that many other users are interested in. All interested users request for the same content from the user A. Once the receivers have established connections with user A to receive data. The sender uploads the content to his clone. Then the sending clone sends content to all the receiving user's clones. In this scenario, the wireless spectrum may be saved since the sender only uploads the content ones when sharing the same content with many users at the same time. Whereas, in a conventional D2D system, the sender may upload n times to send data to n receivers. Aforementioned scenario is depicted in Figure 3.5.

3. One sender - many receives same content from the sender's clone directly.

   This scenario that is shown in Figure 3.6 is very similar to the previous scenario in Figure 3.5, but the only difference is that the sender does not have to send the content to receivers clones. The sender's clone allows all other receivers to get content directly from his clone. This scenario elements

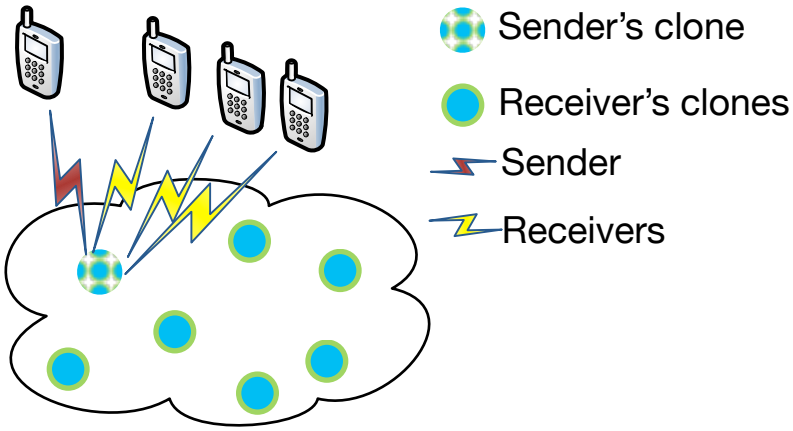

Figure 3.6: One sender - many receives same content from the sender's clone directly.

the second Clone to Clone data transfer step. The benefit of this elimination is that by reducing clone to clone communication, network traffic may be reduced in the backhaul network in C-RAN.

3.5 Mobile Task Offloading Architecture

As described above mobile cloud computing resources could be hosted at the edge of the network as well as on the internet in some deployment scenarios. In Chapter 4, an offloading algorithm for a scenario where the offloading resources made available from both next to wifi access points (cloudlets) as well as in the public cloud (clones), has been proposed. Therefore, the existing offloading frameworks have to be altered to support multiple offloading locations. For this purpose, new updates that need to be made to the existing Thinkair [7] offloading software framework architecture to complement the algorithm that is proposed in Chapter 4 have been proposed. The proposed task offloading architecture is built on the Thinkair framework [7] and is shown in Figure 3.7. However, the author's aim is only to propose a conceptual model of the architecture. Its implementation is still ongoing and will be finalised in future work.

A mobile clone is a VM hosted by a public cloud service provider with an application offloading server (AOS). Every mobile device is assigned a clone in the cloud for offloading and caching purposes. Clones may communicate with

other clones and with cloudlets. A cloudlet is a VM hosted by a resource-rich machine placed next to an AP or a cellular base station. Cloudlets also hold a type of AOS. They are interconnected using a separate network for transferring user data.

3.5.1 Features of the proposed task offloading architecture

1. *Dynamic adaptation to a changing environment* In the proposed architecture, the offloading framework [7] implements code migration such that the code is offloaded to be executed remotely at the most suitable time. Two offloading destinations are available to the offloading framework, clones and cloudlets, so the framework may decide *Where* to offload, depending on the user's environment. Users offloaded data are always backed up to their dedicated clones at all times to prevent data loss.

2. *Performance improvement via cloudlets and clones* Offloading of code to more resource-rich servers in cloud clones may improve the performance of mobile devices. With the increasing bandwidth available, users expect a faster mobile experience. The bottleneck for this is the mobile network status, specifically the response time and available bandwidth when performing online operations. In the proposed architecture, VM-based cloudlets [14] are deployed closer to the mobile device (at a one-hop distance) as temporary points for code execution because the distance and the number of hops directly affect the response time and energy consumption [129]. Users may access cloudlets via high-bandwidth WiFi networks. This may be perceived as bringing the cloud closer to mobile devices. However, it is considered that the clones as permanent execution points. Because it is assumed that every mobile device has a dedicated clone for offloading.

3. *Faster code execution via caching and data localisation* Long-term caches of remote code may be stored in the user-specific clone so that the mobile device will not have to send the code to the clone when offloading and cloudlets may download cached code from the clone to reduce mobile data traffic.

4. *Communication and Energy improvement through communication offloading* One of the main factors for high energy consumption is sending data through low bandwidth networks. So that users will be able to offload communication to the cloud, such that mobile user will only send control mes-

sages to the cloud, but data will be shared among each user's cloud clones by establishing secured Peer to Peer network connection. Also if there is the desired set of data for executing a process on, that may be downloaded from the from the internet, the clone may download it to the clone/cloudlet without the user having to send the data all the way from the mobile device.

5. *Fail-safe synchronised cloudclones/cloudlets and intelligent session handovers*

   In the cases where the user has gone out of the range of the connected cloudlet while in the process of offloading code, the user will not loose the offloaded code or the data that it executed the code on, it will be able to transfer all of the data to the mobile user's clone without mobile user's involvement, or it will send data directly to the next cloudlet that the user has connected to, if it's more efficient to do so through the cloudlet WAN. Also, the architecture supports multiple cloudlets per mobile device to offload code at the same time so that depending on the cost and available resources the mobile device may offload code to minimise energy and also the service cost. Assuming there will more than one cloudlet in reach for mobile users in the future. This allows cloudlet service providers to compete with each other with competitive service costs much similar to cloud service providers.

3.5.2 Components

This section describes component functions and their interconnections, which are denoted by arrows in Figure 3.7. The device–clone link is shown by the dotted red line and the device–cloudlet link by green arrows. A discussion of the modules inherited from the Thinkair architecture is beyond the scope of this chapter. Thus, only the new components are explained.

Cloudlet handler

When a user is in within range of a cloudlet, the cloudlet handler connects the users mobile device to the nearest cloudlet. Cloudlets may be placed next to WiFi APs and to cellular base stations, so the mobile device may save energy by only turning on the WiFi interface when it is within reach of a cloudlet under the control of the cloudlet handler. The handler queries the nearest mobile base station for

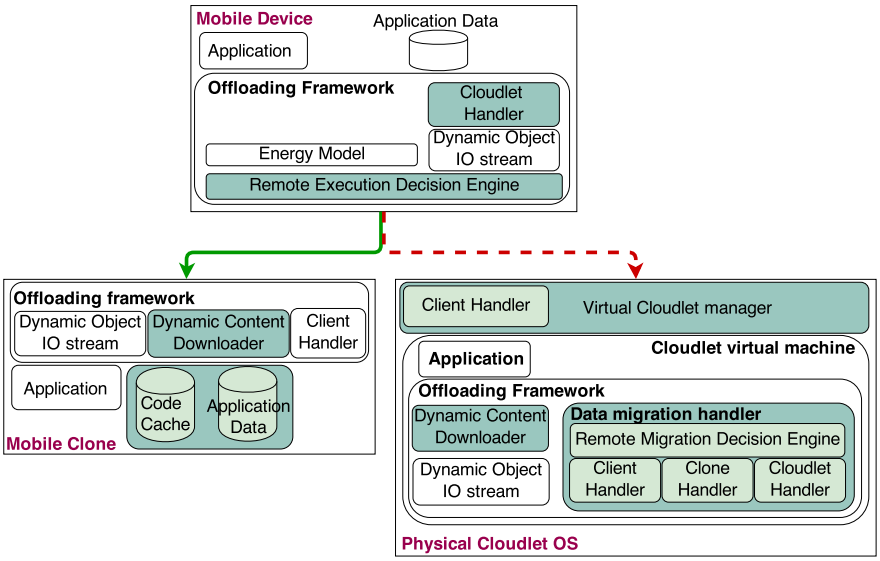

Figure 3.7: Proposed architecture for MCC offloading.

the nearest cloudlets. If a cloudlet is available for connection, the cloudlet handler turns on the WiFi interface and connects to the cloudlet using a well-known public IP.

Remote execution decision engine

The remote execution decision engine is the intelligent component of the mobile offloading framework. It decides on *Whether* and *Where* to offload data. In other words, it decides if an offloadable block of code (method) annotated by the programmer should be offloaded (*Whether*) to a clone or cloudlet (*Where*). The embedded algorithm used to make this decision is presented in Chapter 4.

Dynamic content downloader

Dynamic offloader downloads are placed in the clone and the cloudlet. The downloader downloads input data for tasks when a mobile device offloads computing tasks through the internet so that the mobile device does not have to transmit input data with the code, which saves energy. The input data are downloaded using the fixed network to which the clone or cloudlet is connected, as shown by black

arrows in Figure 3.7. This module is equipped with a utility that may be used to download content by providing a URL, such as GNU Wget [130].

Data migration handler

The data migration handler manages data transmitted between a mobile device and its clone. Similar to Thinkair [7], the offloading framework on the mobile device connects to the client handler on the cloudlet, and the dynamic migration handler connects to the dynamic object input/output stream on the clone and the mobile device. The clone handler in the cloudlet may initiate a session to connect to the client handler in the clone. The client handler in the cloudlet waits and accepts new connections from mobile devices. Only one mobile device that may be connected to one cloudlet at any instant. If WiFi coverage fluctuates, the connection to a cloudlet may drop when the user is moving. The remote data migration decision engine decides on whether to migrate unfinished jobs with their data to the clone or not.

When a mobile device connects to a cloudlet, the URL, clone authentication details, and details for the previous cloudlet to which it was connected (if applicable) are transferred to the cloudlet. Then the remote migration decision engine connects to the clone and automatically downloads the cached data and code to the cloudlet to speed up the offloading process. When the mobile device connects to the next nearest cloudlet, the handler of the new cloudlet connects to the handler of the previous cloudlet to retrieve the code/data. If the user is not too late, the new cloudlet may receive the requested data. If the old cloudlet does not receive any requests for user data after a short period, it transfers the data back to the user's dedicated clone to prevent data loss. This is ideal if many cloudlets are available in user's vicinity when the user is mobile. Since cloudlets may provide a seamless service to the moving users.

Code cache and application data

Our approach utilises any available clone storage by allowing the offloading framework to cache frequently offloaded codes with application input data in the clone. A record of the cached code and data is also kept so that mobile devices may discover whether the code and/or input data already exist on clone so that they do not have to be offloaded. Also, when a user loses connection to a cloudlet, the cloudlet storage system may transmit unfinished/finished jobs back to the user's dedicated cloudlet so that the clone may execute the code and/or the user may

access the data later.

Virtual cloudlet manager

The offloading framework in the cloudlet is placed in a VM (cloudlet VM) hosted by a virtualization platform such as KVM [131] and managed using APIs such as Libvirt [132].

3.5.3 Scenarios

Static

- A mobile device user may offload code or data to a dedicated cloud clone directly. The mobile device communicates with the clone using a cellular network via the Internet. Because it uses networks with low bandwidth and high latency, this conventional offloading approach may be improved.
- Two different Communication offloading methods are possible. Communication offloading to cloudlet or to cloudclone.
- Communication offloading to cloudlet is desired when two entities that communicate are within the same cloudlet cloud.
- In communication offloading scenario, if the communicating mobile devices are connected to different cloudlets, the most efficient way to communicate has to be calculated. The most efficient method will be chosen from following two methods. Through the interconnected cloudlet WAN networks or by communicating with the help of cloud clones.

Dynamic

- Because of mobility, device users may lose connectivity to a cloudlet, but might reconnect to the same cloudlet later or might connect to a different cloudlet. A user may also lose connectivity to a cloudlet but might not reconnect to it for a long time. In all these cases, the offloaded code and data should not be lost during provision of a seamless service.
- Although the user is moving, he/she may still be within the same cloudlet cloud. To provide a seamless cloud service, the unfinished offloaded code/-

data has to be migrated to the next cloudlet that the mobile device connects to.

- The data has to be migrated the most efficient way as possible. If the user moves within the same cloudlet cloud, the data may be migrated via the same cloudlet WAN.

- If the user has moved from its cloudlet cloud to a different cloudlet cloud, one of the following two ways will be chosen to migrate code/data. It may be migrated using the interconnected cloudlet WANs if it's more efficient to move data to the new cloudlet directly from the previous cloudlet. If not data may be transferred to the user's dedicated cloud clone, and then it will be transferred back to the new cloudlet.

3.6 Unified Protocol

Cloud Radio Access Networks allows the cellular networks to process baseband tasks of multiple cells/RRHs jointly as well as to allocate cellular resource to subscribers jointly. In conventional mobile task offloading systems, the offloading framework and the offloading network are not aware of the offloading process. The network treats the offloading data as any other data, and the offloading destination resides outside the network operator's network. The offloading process could have been more efficient if the offloading network resources and cloud resources may be dynamically allocated to fit offloading requirements.

First, the author designs a protocol for communicating between the controller, BBU and the Mobile Cloud for resource allocation. The author assumes that the UE has already discovered its BBU and its Clone. The author uses a uniform PDU (Protocol Data Unit) format for both resource allocation and for task offloading as the name suggests. The proposed Uniform Offloading Protocol (UOP) operates in the Application Layer of the Internet Protocol Suite (Layer 7). Mobile Cloud Controller holds the management and the main point of contact roles for both User (service consumer), and for services in the service providers side, for the offloading protocol. We assume that the mobile cloud controllers are discoverable throughout the network for offloading using web service discovery protocols (e.g. Universal Description, Discovery and Integration). Once a mobile device discovers the most suitable mobile cloud to offload, it will then directly connect to the corresponding controller using the offloading protocol.

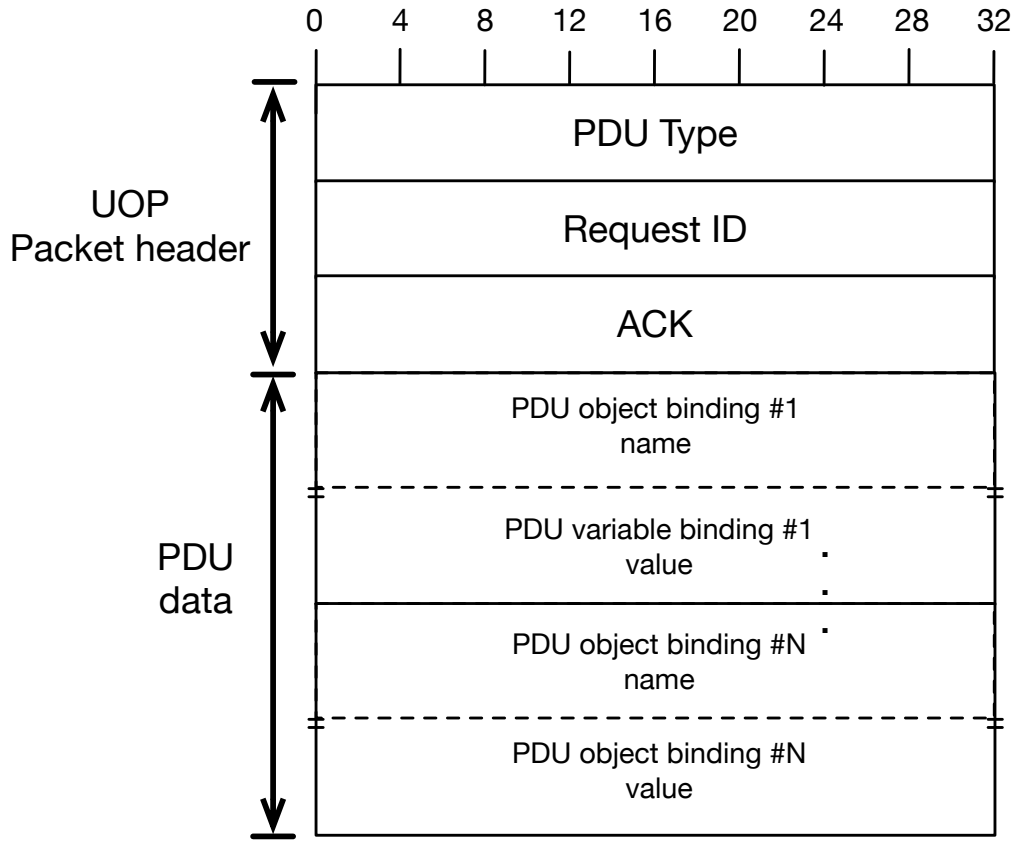

Figure 3.8: Unified Offloading Protocol PDU. (The ticks represent number of bits)

3.6.1 Protocol Data Unit format

The PDUs (Protocol Data Unit) format of both offloading and resource allocation protocol is shown in Figure 3.8. The offloading PDU is then passed down to the TCP layer and may be encapsulated with the TCP packet header. The offloading packet contains application control fields and a payload. The payload may contain offloading code, user data and application errors if any. Moreover, the definitions of the fields are shown in Table 3.1, the PDU types in Table 3.2 and the definitions of object binding name-value pair in Table 3.3

The author has avoided basing the offloading protocol on top of other web application protocols such as HTTP and SOAP due to their complexity and large overhead [133]. Henceforth, it operates on raw Transport Layer Sockets. There-

Table 3.1: UOP attribute definition

Field Name	Value Type	Size (Bytes)	Description
PDU type	Unsigned Integer	4	An integer value that indicates the type of PDU. Refer to Table 3.2 for the list of PDU types.
Request ID	Unsigned Integer	4	An identifier to match requests with replies. The mobile device sets the Request ID in the request PDU and then is copied by the controller and the clone in the response PDU when offloading. The controller sets this attribute when sending resource management messages.
ACK	Unsigned Integer	4	0 if response packet, 1 if Acknowledgement packet and, request rejection packet if the value >1
Object Binding	Object	Variable	A set of name-value pairs identifying application objects to execute, user data and error messages with their corresponding object references. Refer to Table 3.3 for object definition.

Table 3.2: PDU Types (some examples)

PDU Type value	PDU type
0000	Offload_Req
0001	Offload_Accept
0002	Offload_Denied
0003	Offload_Start
0004	App_Register
0005	App_Request
0006	App_Data
0007	App_Response
0008	Offload_FIN
0009	Manage_Compute
0010	Manage_BBU

Table 3.3: Definitions of Object Binding name-value pair

Subfield Name	Value Type	Size (Bytes)	Description
Object_Name	Sequence of Integer	Variable	Object identifier (code, user, data, application error)
Object_Value	Object	Variable	Contains the values of the specified object type.

fore, depending on the Transport Layer protocol that the offloading protocol (TCP/UDP) implements, the Client Handler in the controller and the offloading framework in the clone will listen on a known port for messages that are sent from the offloading mobile device. In our case, the offloading protocol is designed on top of TCP.

- Time-outs takes an important role in the protocol for assuring the timeliness of individual transactions. Such timeouts are conventionally implemented within applications. Instead, the offloading protocol handles timeouts and indicates the applications if the protocol has timed out waiting for service responses.

- Each transaction (offloading or resource management procedure) must succeed, but in a case of a failure, the system should roll back to its previous state. The acknowledgement messages assure the completeness of transactions. If a task or a set of tasks fail, then an error message will be sent back instead of acknowledgement. The protocol assumes an offloading task as one transaction. Within this main transaction, there are multiple individual sub-transactions. Namely, they are communication resource allocation task, compute resource allocation task, remote code execution tasks. If any of aforementioned sub-transactions fail, the main transaction is also considered failed. The communication and computing resources that were allocated will be unallocated and put back into the pool of globally available resources. Finally, the code will have to be executed locally by the mobile device, if the delay restrictions do not allow it to resend an offloading request. From start to the end of an offloading transaction, mobile cloud controller keeps track of all protocol and application states.

- Implementing the protocol on top of TCP, automatically inherits TCP reliable delivery, error correction and ability to add optional Transport Layer Security (TLS/SSL) layers.

- Keeping the protocol and the packet format as simple as possible for reducing overhead and processing complexity.
- Separate management functions from services for increasing scalability and centralising management of services.
- Integrated application error reporting to the offloading protocol.
- The offloading protocol is independent of the mobile operating system and the offloading framework.
- Integrated resource allocation to the offloading protocol.
- The payload of the offloading protocol packet may carry more than one offloading task.

3.6.2 Working Procedure

The Figure 3.9 shows an instance where the protocol successfully instructs the UE to offload computationally intensive tasks to the mobile cloud. Once the UE receives the "$Offload\_Start$" message with corresponding information about the offloading location (the clone), it successfully carries out offloading tasks. The Figure 3.10 illustrates an instance where the controller fails to allocate BBU resources for the user. Therefore the offloading request gets rejected by the controller, by finally sending a "$Offload\_Denied$" message back to the UE. In both above figures, the acknowledgement messages are shown by appending "_ACK" at the end of corresponding originating message name, to indicate the ACK bits have set to 1 in the packet header. Resource monitoring is out of the scope of this protocol, and it is assumed that the controller uses existing monitoring protocols for monitoring C-RAN and mobile cloud resources. Moreover, resource allocation, estimation and prediction algorithms are out of scope of this chapter.

3.7 Summary

In this chapter, the author proposed a novel system that includes, a controller which cooperatively/jointly manage computing and communication resources, a unified protocol that is used for offloading and managing mobile cloud resources and C-RAN wireless resources, and a task offloading architecture for offloading computationally intensive tasks.

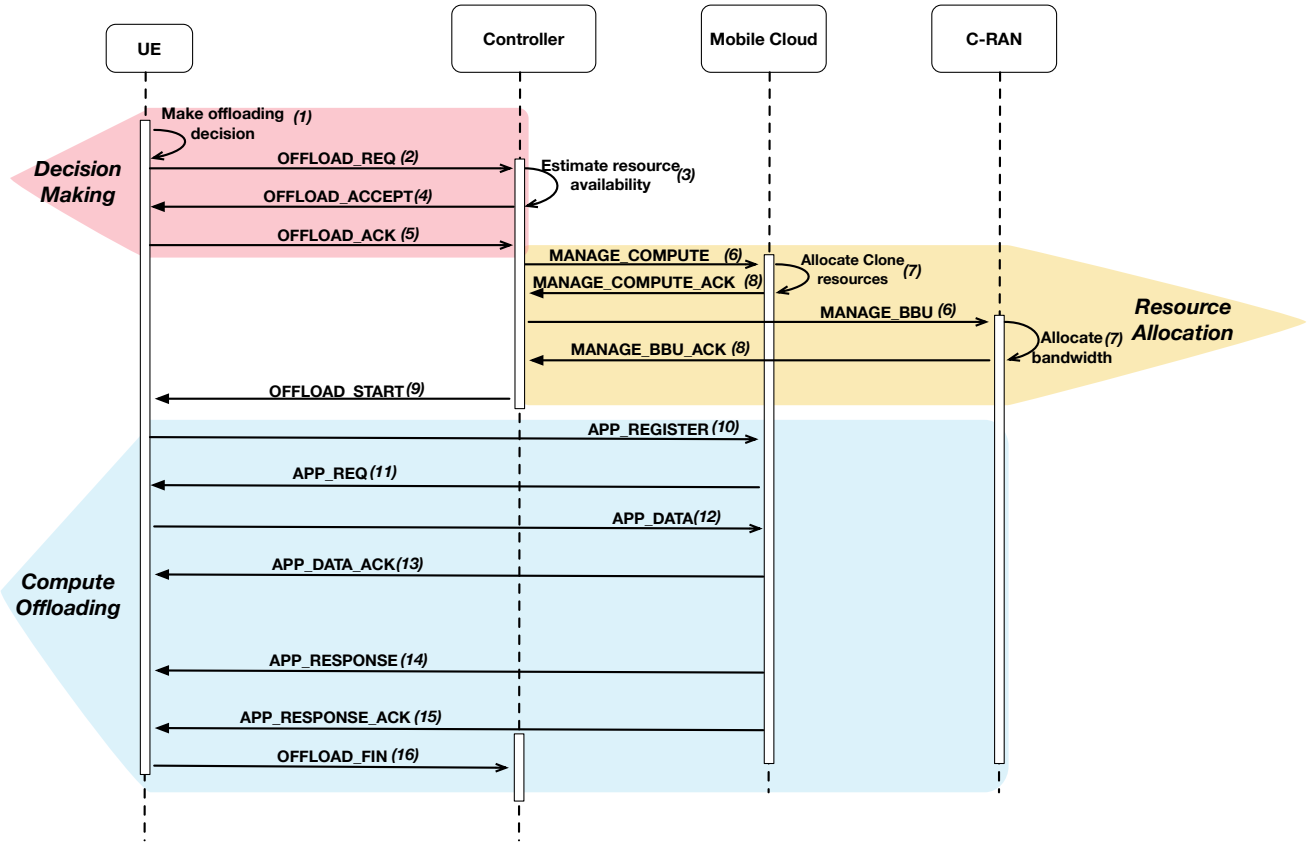

Figure 3.9: Unified Offloading Protocol: Procedure when successfully allocates resources

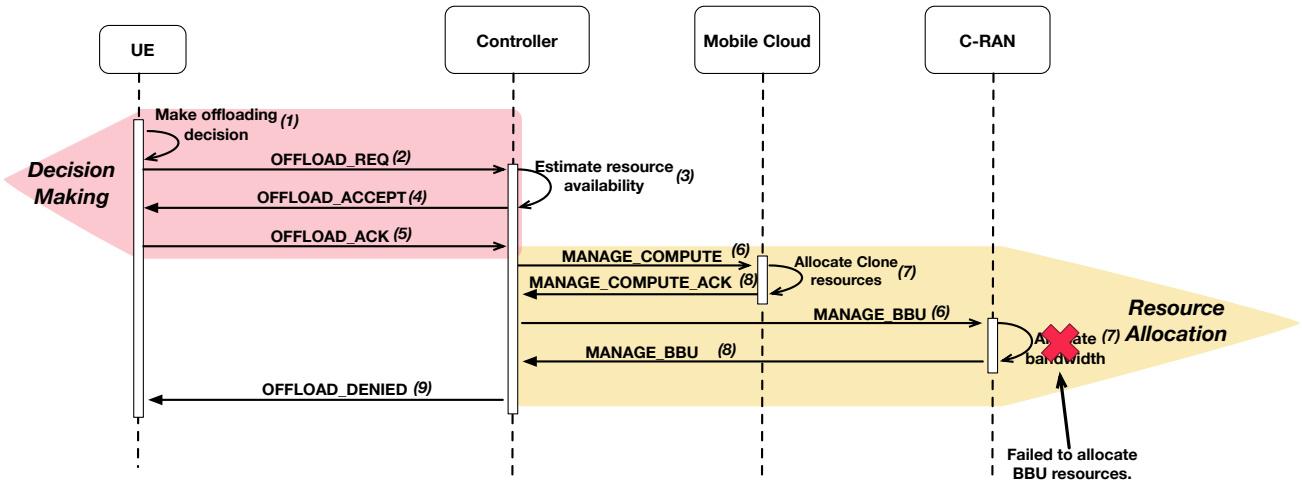

Figure 3.10: Unified Offloading Protocol: Procedure when resource allocation failed

Chapter 4

Network-Aware Offloading Algorithm

4.1 Introduction

The author contributes to the exciting MCC research field by investigating a key problem for mobile application offloading. There is much work being carried out in this area, which largely falls into two categories: (1) task partitioning, which involves dividing an application into offloadable partition(s) and a local partition [23]; and (2) virtual machine (VM) selection [134], which involves choosing an appropriate VM to which to offload a partition. VMs are the key component of a cloud, and they provide virtual resources such as CPU, memory, storage, and network interfaces in the same way as physical resources do. One common element of this research is the assumption that there are a perfect network connection and sufficient bandwidth between a mobile application on a mobile device and the remote cloud VM. This may not be an unrealistic assumption for wired networks, for which network bandwidth is usually abundant or at least not scarce. However, this is not the case for wireless networks[135], for which network bandwidth is not as abundant and a network connection may sometimes not even be available; this is more of a problem for mobile cellular networks such as 3G. To address this issue, the author uses a new network-awareness perspective: in addition to considering network status parameters such as bandwidth and delays, the network types have also taken into account. In this study, two types of wireless network have been considered: IEEE 802.11 (i.e., WiFi) and mobile cellular networks such as 3G. These are the two mainstream wireless technologies that people interact with

on a daily basis. Considering one without the other is not realistic and leads to a situation in which a mobile cloud system is not as efficient as it should be.

The author introduces two types of cloud resources. Persistent and transient cloud resources. Persistent cloud resources continue to exist and serve their dedicated mobile devices. They do not only contain computing cloud resources but also provides storage services to the mobile users where they may store data such as user-generated content, cached data and other data that belongs to their dedicated mobile users. Hence, it is of type persistent. On the contrary, as the name suggests transient cloud resources are temporary compute only cloud resources that are placed at the edge of the networks. At the point in time, this chapter is written, virtual machines have been used as the primary technology for building cloud service infrastructures. However, recently there is other new virtual machine like technologies have been emerging to the market for, such as Docker [136]. Accordingly, throughout the chapters the author calls the persistent virtual machines ”Clones” and transient virtual machines ”Cloudlets”. It is assumed that all subscribers of the mobile operator do have their dedicated clone that comes with their mobile subscription. Although a cloudlet is a temporary code execution location, it may maintain a peer-to-peer connection with the user's dedicated Clone for receiving user specific data and cached code.

The Cloudlets should be used and placed at the locations where there is no enough infrastructure to host Clones. It is required that the clones may respond to user's requests at all times. Hence, unlike cloudlets, the clones have to exist in the cloud at all times. Whereas, a Cloudlet virtual machine is created temporarily when a user initiates a connection to the cloudlet network for offloading compute tasks. Once the offloading procedure is over, the created cloudlet will be destroyed or cleaned and reused by another user. Furthermore, a cloud where the clones reside (clone cloud), may also maintain user subscription information, such that the clone cloud may associate the clones with their corresponding users and also able to tailor the cloning service according to the subscription price plan. Moreover, the clone size and the hosted services in clone may vary depending on the type of mobile subscription. Hence, one obvious location to host the mobile clone cloud could be a data centre that is owned by the mobile operator. In this case, the mobile operator also becomes the mobile cloud service provider. On the other hand, cloudlets are easier to be deployed due to its fewer demand on the infrastructure.

Aforementioned considerations are reflected in the proposed MCC system design, in which a middle layer is introduced between mobile devices and their corresponding cloud clones. This middle layer, called the cloudlet layer, is de-

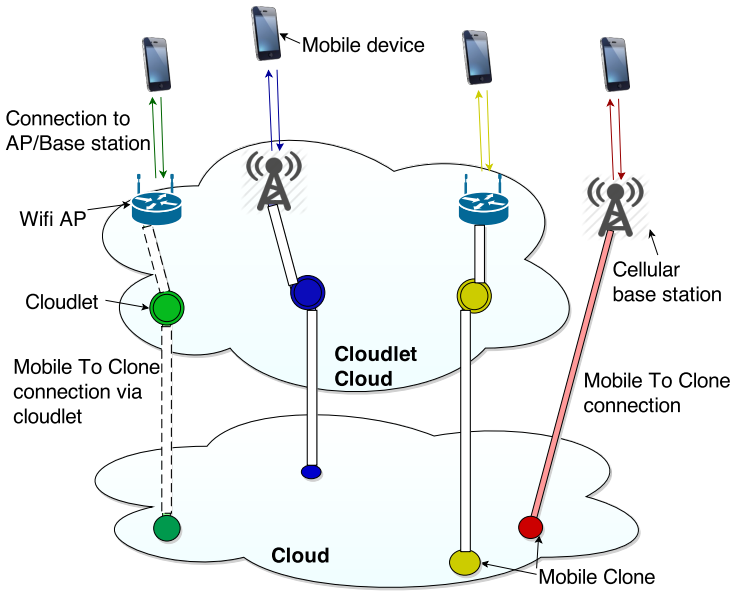

Figure 4.1: Illustration of task offloading with clones and cloudlets.

ployed next to WiFi access points (APs) in the proximity of mobile devices. The aim is to run clone equivalents named cloudlets in this layer, as illustrated in Figure 4.1. The benefits of this approach are twofold. First, it may take advantage of the higher bandwidth of WiFi and switch the network connection from cellular to WiFi. Second, temporary data caching may be carried out on cloudlets to some extent for applications that download data from the Internet. One deployment method would be MCC service providers to install their offloading software on the home wireless gateways or servers of end users. If the home WIFi access point also belongs to the mobile operator, then providing an offloading platform with it for the WFi users becomes much easier.

Furthermore, our proposed offloading algorithm in Chapter 4.2 is aware of the offloading location (clone or cloudlet) when offloading a mobile application. In contrast, most of the current literature on offloading algorithms decides *whether* to offload a mobile application or not. Cloudlets may also be placed next to cellular base stations, as illustrated (connecting a mobile device, cloudlet, and clone) in Figure 4.1. Also, clones may be placed next to the cellular base stations and WiFi access points as shown in Figure 4.1.

Table 4.1: Notation conventions

Symbol	Unit	Description
i^j	MIPS	Number of instructions for completion of task j
T	seconds	Task hard deadline
t^j_{mob}	seconds	Execution time for task j on a mobile device
t^j_{cloud}	seconds	Execution time for task j in a cloud (cloudlet or clone)
E_{local}	J	Energy consumption by a mobile device for task execution
E_{cloud}	J	Energy consumption by a mobile device when a task is offloaded to a cloud
E_{nic}	J	Energy consumed by a network interface (E^{wifi}_{nic}, E^{3g}_{nic}, E^{lte}_{nic})
μ_{cpu}	MIPS	Mobile CPU speed
μ_{cloud}	MIPS	Cloud speed
B	Kbps	Transmission bandwidth
d_j	KB	Data that need to be transmitted for task j
P_{trans}	Watt	Power consumption of the network interface
P_{basic}	Watt	Baseline power consumption when the mobile device is idle
P_{comp}	Watt	Power consumption of mobile CPU
κ_ν	%	Proportional reduction in execution time compares to mobile execution
γ_ν	%	Proportional reduction in energy compared to mobile execution

4.2 Algorithm

The offloading algorithm embedded in the *remote execution decision engine* decides on *Whether* and *Where* to offload data. The novelty of this algorithm is that it considers more than one offloading location as a parameter when deciding *Where* to offload. By contrast, conventional offloading algorithms support just one offloading location [2]. In the remainder of the section, it is assumed that cloudlets are only located next to WiFi APs and mobile devices access clones using their cellular network via the Internet, as shown by dashed and filled bars respectively in Figure 4.1.

Before application execution, the decision-making process seeks to reduce the total energy consumed by the mobile device before execution [49]. A novel energy and network aware offloading algorithm that takes two offloading locations into account, has been proposed in Figure 4.1. The energy model considers ramp, tail, and maintenance measures when estimating energy, unlike energy models used by existing offloading frameworks.

First, the task execution time and the device energy consumption is estimated.

Then describes the algorithm that uses these estimates to make offloading decisions. The notations used are listed in Table 4.1.

4.2.1 Estimating the response time and energy for local execution

The execution time for task j is estimated as

$$t^j_{mob} = \frac{i^j}{\mu_{cpu}}. \tag{4.1}$$

The energy consumption for local execution is given by

$$E_{local} = (P_{comp} \times t^j_{mob}). \tag{4.2}$$

4.2.2 Estimating the response time and energy for offloading

The response time for offloading of tasks to a remote cloud server is calculated as

$$t^j_{cloud} = \frac{d_j}{B} + \frac{i^j}{\mu_{cloud}}. \tag{4.3}$$

It is assumed that a mobile device accesses its cloud clone or cloudlet using a WiFi and/or cellular network, so the calculation of the energy consumption for operations performed on these interfaces is crucial. Zhang *et al.* [35] carried out a detailed study of the power consumption of mobile device components using PowerTutor, an energy estimation tool that considers CPU, LCD, GPS, WiFi, audio, and cellular interfaces. A similar model for WiFi and cellular networks has been used in this work, although in their work the power consumed when a task is executed on a cloud and when data are transferred are the only parameters that are considered. The energy consumption is estimated according to;

$$E_{cloud} = P_{basic} \times t^j_{cloud} + E_{nic}, \tag{4.4}$$

where P_{basic} is the baseline power consumption of the mobile device when it is idle, $t^j cloud$ is the time for task execution in the cloud, and E_{nic} is the energy consumed by the network interface when offloading to the cloud. If $E_{nic} = E^{wifi}_{nic}$, then the equation yields the energy consumption for offloading to a cloudlet.

As empirically observed by Zhang *et al.* [35], the packet rate, not the bit rate, determines the power state of the WiFi interface. In other words, the packet size

does not influence power consumption given a fixed channel state and packet rate. However, Zhang did not take the energy for connection establishment into account [42]. In author's work in this chapter, a similar model has been used, but it also takes the connection establishment energy E_e into account. The author expresses the system energy cost for establishing and transferring n bytes as

$$E_{nic}^{wifi} = E_e + n \cdot E_{trans}. \tag{4.5}$$

It is assumed that there is no significant difference in power consumption between receiving and transmitting packets, so the same values have been used to represent transmitted and received energy.

Models of energy consumption for cellular networks consist of three components [36]: (1) *ramp* energy e_{ramp}, (2) *transmission* energy e_{trans}, and (3) *tail* energy t_{tail}. Unlike Zhang *et al.*. [35], the author considers radio resource control states. The power amplifier of a mobile cellular interface switches to a higher power mode to counter the drop in signal strength [137] when transferring and receiving data. Cellular base stations use feedback received from mobile devices some 800–1600 times per second and choose an appropriate modulation scheme and data rate. Hence, a strong signal allows for high data rates and shorter data transfer times. It may be concluded that when the signal is weak, data transfer take longer to complete, and the radio interface draws higher power. Considering these factors, the energy model for cellular data transmission is given by

$$E_{nic}^{3g} = e_{ramp} + e_{trans} + e_{tail} \times t_{tail}. \tag{4.6}$$

The author uses a simple mobility model similar to that of Huang *et al.* [138]. The author assumes that the availability of wireless networks and their data transmission rates vary according to the current location of the mobile device. In contrast to Huang *et al.* [138], It is assumed that the availability of networks for the mobile user may change during the remote execution time for one application.

4.2.3 Decision-making

At runtime, the offloading framework dynamically requests the cloud server speed of the cloudlet μ_{cloud}^{ct} from the connected cloudlet; it is assumed that the speed of the dedicated clone, μ_{cloud}^{cl}, is already known by the mobile device. The framework then calculates the proportional reduction in execution time for offloading the task to a clone or cloudlet as

$$\kappa_\nu = \frac{t_{mob} - t^{\nu}_{cloud}}{t_{mob}}, \tag{4.7}$$

where the subscript ν may be either *ct* or *cl*, denoting cloudlet and clone, respectively. If Eq. (4.7) yields a negative value, offloading will not be time-efficient. A hard deadline T_j is defined for task j. This deadline must be met and it is preferable to choose the offloading location that executes the task in the shortest time, even though energy consumption has a higher priority than execution time in the offloading decision engine.

The proportional reduction in energy consumption for offloading a task to a clone or a cloudlet is given by

$$\gamma_\nu = \frac{E_{local} - E^{\nu}_{cloud}}{E_{local}}. \tag{4.8}$$

If Eq. (4.8) yields a negative value, offloading will not be energy-efficient. It is obvious that if $\gamma_{mob} = 0$ and $\kappa_{mob} = 0$, then tasks are executed locally. If $\gamma_{cl} > 0$ and $\kappa_{cl} > 0$, then it is energy and time-efficient to offload the task to a dedicated clone. If $\gamma_{ct} > 0$ and $\kappa_{ct} > 0$, then it is energy and time-efficient to offload the task to a connected cloudlet or clone.

The proposed offloading approach is presented in Algorithm 1, which favours the execution time over energy consumption as the criterion for task offloading. Moreover, this is because the application will fail to reach quality of service (QoS)/quality of experience (QoE) requirements if the task deadline is not met. By contrast, the execution time has a direct affect on energy consumption as defined in Section 4.2.2. If offloading of a selected task to either a clone or a connected cloudlet may not improve the execution time, the task will not be offloaded (line 1). If the cloudlet may execute the task faster than the clone may, and the energy cost is less for offloading to the cloudlet or the clone while meeting the deadline, the task will be offloaded to the cloudlet, and vice versa (lines 5–9). Next, if the mobile device may save more energy by offloading to the cloudlet than to clone while meeting the deadline, the task will be offloaded to the cloudlet. Otherwise, the clone will be selected as the offloading destination (lines 10–15). If none of the above conditions is met, the task is executed locally (line 16).

Algorithm 1 Energy-efficient and network-aware decision algorithm

1: $\forall \nu$ calculate γ_ν & κ_ν
2: **if** $\kappa_{cl} \leq 0$ & $\kappa_{ct} \leq 0$ **then**
3: *return* Do not offload
4: **else**
5: **if** $\kappa_{ct} > \kappa_{cl}$ & $\gamma_{ct} > \gamma_{cl}$ & $t^j_{ct} \geq T$ **then**
6: *return* Offload to Cloudlet
7: **else if** $\kappa_{cl} > \kappa_{ct}$ & $\gamma_{cl} > \gamma_{ct}$ & $t^j_{cl} \geq T$ **then**
8: *return* Offload to Clone
9: **end if**
10: **if** $\gamma_{ct} > \gamma_{cl}$ & $t^j_{ct} \geq T$ **then**
11: *return* Offload to Cloudlet
12: **else if** $\gamma_{cl} > \gamma_{ct}$ & $t^j_{cl} \geq T$ **then**
13: *return* Offload to Clone
14: **end if**
15: **end if**
16: Do not offload

4.3 Performance evaluation and analysis

The name Cloudlet+Clone is used hereafter to refer to our proposed architecture. The terms CloudletClone and CloneOnly in figures refer to Cloudlet+Clone and a conventional architecture, respectively.

4.3.1 Simulation set-up

The author has conducted several simulations to study the performance of the proposed MCC system architecture and the offloading algorithm. The algorithm is implemented using CloudSim 2.0, which is an extensible toolkit for modelling and simulating cloud computing environments that support VM nodes in data centres. [139, 140]. The effectiveness and efficiency of our approach were evaluated mainly concerning the average service response time and the average energy consumption by a mobile device. Comparisons were made among the following three scenarios: mobile only, clone only, and Cloudlet+Clone.

The simulation environment was configured as follows. It is assumed that the network state was constant for every offloading instance, for example, that the bandwidth and link latency for the uplink and downlink were constant throughout

the execution process for one task. The CPU intensity of a task is represented in million instructions per second (MIPS). The proposed architecture has two types of network available for offloading. When offloading, the time taken to execute a task in a particular environment may be taken into consideration because of task deadline values to achieve the promised QoE level. Hence, the author measured differences in response time between Cloudlet+Clone and other popular architectures for task execution.

For the evaluation, it is assumed that the CPU power (MIPS) of the remote server is double that of the mobile device and that the size is fixed throughout because the resources are much greater for VMs in the cloud than for mobile devices and may be resized on demand. Hence, the computational power of a cloud clone or cloudlet may exceed double the power of a mobile device. The author chooses 500 MIPS for the clone and 1000 MIPS for the cloudlet.

The author assumed that the maximum bandwidth of the wireless network for Cloudlet+Clone is 300 Mbit/s since the cloudlets are located next to WiFi APs, although the bandwidth may be even higher for wireless 802.11n devices. For offloading frameworks that only support clones, the bandwidth is 15 Mbit/s and the latency is 0.020s when accessing a clone hosted by a cloud service provider. The average latency was measured experimentally using a real mobile device with a 3G or WiFi network enabled in various geographic locations.

4.3.2 Response time

The aim of these simulations was to demonstrate the Cloudlet+Clone efficiency in executing offloaded code. The number of instructions W required to execute a task for a given data size of L is $W = XL$, where X may vary depending on the type of application [141, 142]. Figure 4.3 depicts a scenario in which the input size increases by 300 bytes as the instructions increase by 50 000 MIPS. The above is a typical scenario in which the instructions to be executed increase exponentially in all cases when the input data size is increased for one task. For evaluating the proposed architecture, it is assumed that the number of instructions increases by 166 MIPS per byte. Cloudlet+Clone performs best in this case because it is equipped with a network with a larger bandwidth compared to clone-only conventional architectures. The response time increases exponentially with the input data size in all cases. Depending on the task deadline, this response time might not be good enough because it directly affects the QoE of mobile applications.

The author also considered a case in which the number of instructions for execution may vary depending on the task, regardless of the size of the input data.

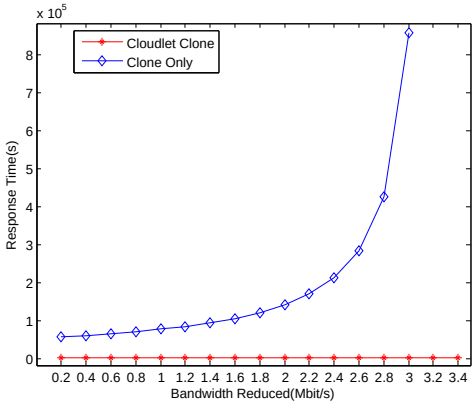

Figure 4.2: Response time for a conventional architecture and Cloudlet+Clone when the bandwidth is reduced by a fixed amount.

Figure 4.4 shows how the response time increases with the number of instructions. It is apparent that it is much quicker to offload the code to a remote server (clone or Cloudlet+Clone) because of the lack of computational resources in mobile devices. In particular, Cloudlet+Clone is even quicker when offloading because data transmission is much quicker than via traditional offloading frameworks owing to the smaller network latency and larger bandwidth. It is clear that the bottleneck for offloading computation is the transmission time. Two feature of the proposed architecture overcome: (1) the high bandwidth of the offloading framework and (2) the use of data caching.

The proposed architecture supports local data caching on a clone and a cloudlet. This means that when offloading, a mobile device does not have to transfer input data with the code if the data may be downloaded via the Internet. The input data do not have to be sent with the code if they are already stored in the clone. In such cases, if the code is offloaded to a cloudlet, the cloudlet will retrieve these data from the user's dedicated clone using the fixed network. Current popular offloading frameworks do not support this functionality. As shown in Figure 4.5, applications benefit from having an offloading framework with data caching enabled in comparison to traditional offloading frameworks (clone without data caching) and clones with data caching enabled. Figure 4.6 compares the response time when offloading for traditional clones and Cloudlet+Clone with data caching enabled.

Every offloading framework considers the state of its available network to de-

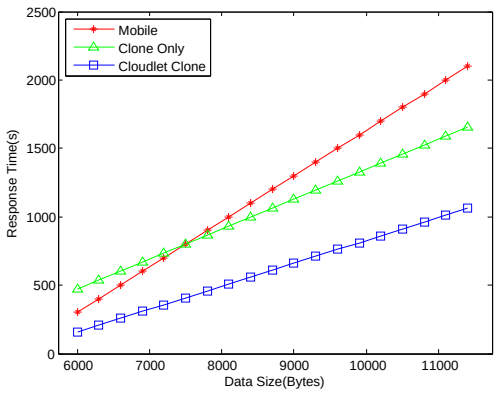

Figure 4.3: Task response time as a function of input data size.

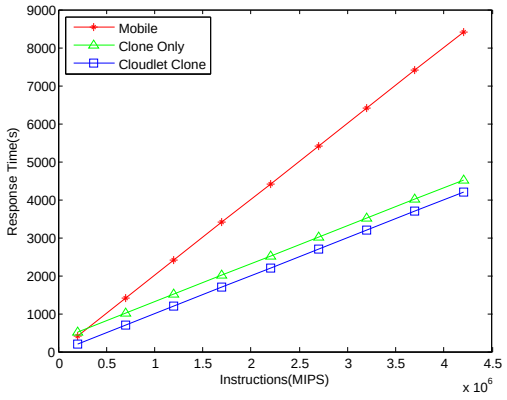

Figure 4.4: Task response time as a function of the number of instructions.

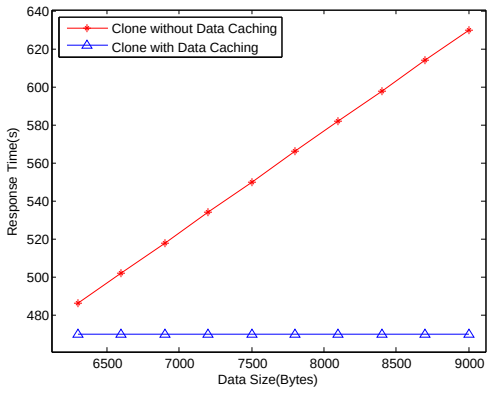

Figure 4.5: Response time as a function of input data size with data caching (proposed architecture without Cloudlet) and without data caching in a clone (conventional architecture).

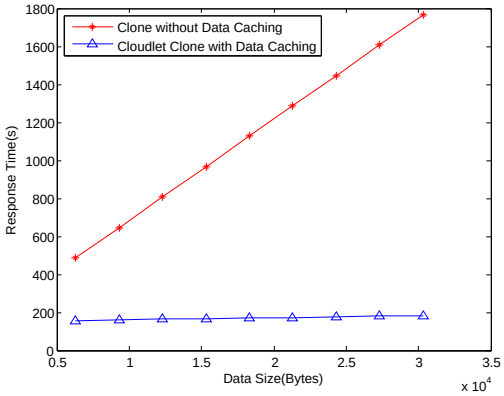

Figure 4.6: Response time as a function of input data size without data caching on a clone (conventional architecture) and with data caching in Cloudlet+Clone (proposed architecture).

	3G	WiFi
Transfer energy	0.025(x)+3.5	0.007(x)+5.9
Maintenance	0.02 J/s	0.05 J/s
Tail energy	0.62 J/s	NA
Tail-time	12.5 s	NA

Table 4.2: Wireless energy model for downloading x bytes of data over 3G and WiFi networks

cide whether it is worth offloading. The bandwidth of the network is a function of the network state. Hence, it is not appropriate to assume that the network state remains constant until a user finishes using an offloadable mobile application. Bandwidth reduction may negatively affect the performance of the offloading framework. Because of the previous statement, it is important to analyse the effect when the available network bandwidth decreases. The results are shown in Figure 4.2 for a fixed reduction in bandwidth when offloading the same task with the same number of input data. It is evident that the decrease in response time when offloading is far greater owing to the small bandwidth of traditional offloading frameworks, whereas the proposed architecture supplies more bandwidth to the offloading framework by adding WiFi-based cloudlet support.

4.3.3 Energy consumption

The author simulated the energy consumption of the CPU and the network interface of a mobile device. The aim was to give an indication of the energy consumption of the mobile device rather than to report precise measurements. When calculating the energy consumption of a mobile device, the author considers the CPU energy consumption for local execution of a task, and the energy consumption of the network interface when offloading. For a CPU with approximately 500 MIPS, the energy consumption is 0.9 W [1]. The parameters listed in Table 4.2 were used to estimate the energy per byte [36].

Figure 4.7 compares the energy consumption for a task executed on a mobile device only, a clone only, and Cloudlet+Clone as the input data size for the same task increases, which also increases the number of instructions to complete. The results show that the Cloudlet+Clone architecture saves a significant amount of energy owing to its high bandwidth and low-latency networks. Figures 4.8 and 4.9 compare energy consumption with data caching enabled in a clone only and

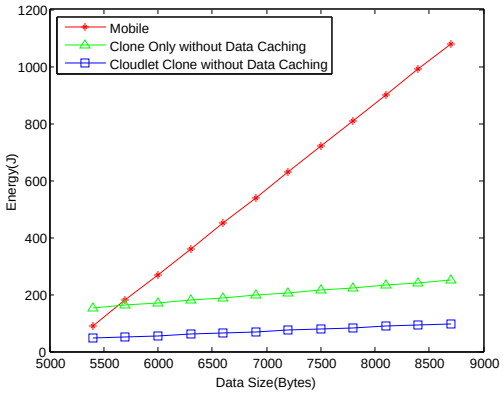

Figure 4.7: Energy consumption of a mobile device with a clone only and Cloudlet+Clone when increasing the input data size without data caching.

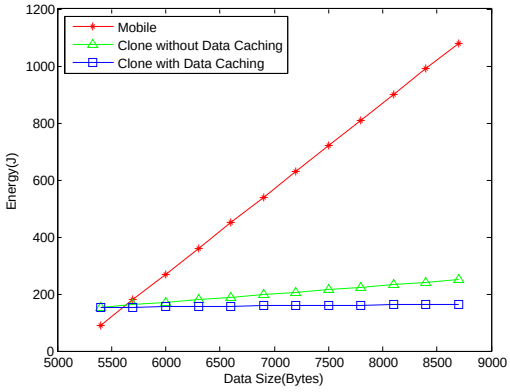

Figure 4.8: Energy consumption of a mobile device when offloading without data caching in a clone (conventional architecture) and with data caching enabled in a clone (proposed architecture without Cloudlet).

Cloudlet+Clone, respectively, with energy consumed when the task is executed on a mobile device and a clone without data caching. In all cases, Cloudlet+Clone performs better than the comparative architecture because of the reduction in transmitted data.

4.4 Summary

In this chapter, the author proposed a network-aware and energy-efficient offloading algorithm that recides on task offloading mobile devices. It decides whether to use the cellular network to offload to a clone in the internet, or to use the Wi-Fi network to offload to a cloudlet at one hop distance. The results illustrated the benefits of offloading to cloudlets. It has evidently shown, that it is beneficial to use high-bandwidth low-latency networks for mobile task offloading, confirming Kumar *et al.* [1]'s observations.

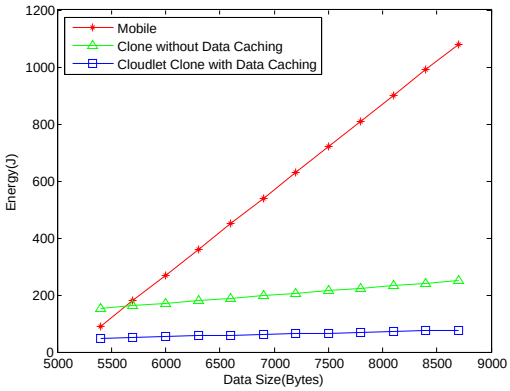

Figure 4.9: Energy consumption of a mobile device when offloading without data caching in a clone (conventional architecture) and with data caching enabled in Cloudlet+Clone.

Chapter 5

A Performance Analysis of Cloud Vertical Scaling for Delay Constrained Applications

5.1 Introduction

Auto-scaling in IaaS clouds has been studied extensively in literature covering various aspects, for allocating the right amount of resources to the users, while meeting Service Level Agreement(SLA) requirements and keeping the overall cost to a minimum. Although there have been many techniques proposed in literature, in the past few years, for automating cloud resource scaling procedures using predictive and reactive methodologies, there is still room for improvements. Especially in fundamental aspects of underlying technologies and algorithms that affect the performance of auto scaling. Some known aspects of cloud computing that require improvements are [20] [18] [143] [118] [119], real-time hypervisors, real-time operating systems (OS), OS support for Cloudification (e.g. native cloud orchestration support in OSs), improved host resource sharing among guests (e.g. reduced latency), user friendly cost models and auto scaling techniques for real-time applications. Specifically, when adopting cloud technologies into mobile systems, issues mentioned above are imperative due to high QoS/QoE requirements and dynamic nature of wireless networks and mobile applications [118] [119]. However, above matters are out of scope for this chapter. The author's focus has been drawn to providing an extensive performance analysis for improving auto scaling algorithms for real-time task offloading on today's existing cloud

environments by addressing issues that are generic to current cloud platforms and cloud service providers, by providing solutions which do not directly operate on underlying virtualisation technologies.

Current literature have proposed various reactive and proactive techniques to achieve horizontal auto scaling (e.g. [85] [74] [76] [65] [59] [74] [75] etc.) by adding or removing VMs to/from a cloud application, or to achieve vertical scaling (e.g. [61] [73] [82] [109]) by adding or removing resources to/from existing VMs. All of which are reviewed extensively in Chapter 2 by highlighting the techniques (proactive and reactive) they have employed. Although throughout literature threshold rules based techniques are the most highly valued techniques, it is evident that reinforcement learning, queuing theory, control theory, time series analysis have also been widely used. However, horizontal scaling has been used more often, in comparison to vertical scaling in literature and by cloud service providers due to some reasons. For both horizontal and vertical scaling, the resource provisioning introduces a delay. Therefore the desired effect may arrive when it is too late. Therefore, current literature stress the need for future work on auto scaling focusing on reducing the time required to provision new VMs (or resize VMs when vertically scaling) [20].

Vertical scaling is known to have a lower range when it comes to scaling up and down [19], and for the changed resources to take effect often the VMs have to be restarted [20]. Even if the underlying virtualization technology (e.g. Xen CPU hot-plug) allows you to scale VMs without restarting, it may take up to 5-10 minutes for the changes to take effect and be stabilised [73] [144]. Moreover, this is mostly because most conventional operating systems do not allow real-time dynamic configurations on the VMs without rebooting [20], and [73] reports this may also be partially due to the backlog of requests in prior intervals. However, many works have proposed a number of vertical scaling algorithms, and they assume that vertical scaling actions may be performed timely, or implement the auto-scaling algorithm using resource hot-plugging [86] [89] [88] [83] [82] [144] [73] [61]. However, the latter is not possible for most of other popular hypervisors (e.g. Kernel Virtual Machine), cloud platforms (e.g. OpenStack) and cloud service providers (e.g. Amazon EC2 [71]) [20]. For reasons as mentioned above, horizontal scaling has been more attractive to the community and has been adopted by existing cloud service providers instead of vertical scaling [61] [71]. Elastic Application Container (EAP) proposes an alternative provisioning mechanism to heavy VM provisioning [145], but requires you to alter the underlying infrastructure. Furthermore some papers have proposed horizontal-vertical hybrid scaling systems where both approaches are used simultaneously for achieving a

better performance by utilising benefits of both methodologies [58] [61] [62] [104] [19].

It is considered wasteful to use a VM as the smallest resource unit when allocating computing resources to a cloud application, as the new VMs that are assigned to the applications not used immediately after, and each VM consumes resources that are not directly utilised by the applications [61]. Which also meant that as the number of VMs increase, the total number of resources that are consumed for just hosting and for keeping the VMs alive also increase. Furthermore, current literature have emphasised that adding or removing whole VMs to applications is not always required for many real world scenarios [61], but subtle changes such as adding or removing available resources are sufficient. Dutta *et al.* [19] further clarifies and claims while horizontal scaling allows the application to achieve higher throughput levels per each addition, the deployment cost is greater than vertical scaling. Therefore, this lead us to investigate further into using vertical scaling for clones in the mobile cloud to handle task offloading workloads.

When the reactive autoscaling approach is used, the system reacts to changes in *ingress* requests when the monitoring system detects such changes. Often threshold-based rules are predefined by the user, and the actions that are assigned to these rules are triggered once changes in workload are detected from the last read monitoring values of the system. Proactive algorithms anticipate and predicts future workload demands and allocates resources accordingly to applications ahead of time. The drawbacks of reactive approach are, it is unable to cope with high sudden bursts of workloads and difficulty to compensate to VM resize and VM bootup delays. In contrary, proactive approaches are not able to react to sudden, unpredictable changes in workloads, and forecasted future workloads may sometimes be inaccurate [61] [20]. Due to all above reasons, some researchers have proposed a reactive-proactive hybrid approach where both techniques have been used; proactive techniques for predicting future workloads and reactive techniques for correcting prediction errors and react when predictions are incorrect [64] [78] [146] [76]. Despite the drawbacks, due to its simplicity, existing commercial cloud service providers offer mostly reactive threshold based auto-scaling services [20].

The above reasoning lead the author to investigate into how vertical scaling may be used to handle task offloading workloads. It is evident that the long resize time when vertical scaling is not suitable for real-time applications such as task offloading, where tasks need to be completed before a given deadline. The author investigates into vertical auto scaling delay times. Moreover, from the lessons learned from this analysis, future researchers may design auto scaling al-

gorithms which may minimise the downtime (allowing the VMs to reboot) and reduce the amount of SLA violations, in existing popular cloud frameworks without resource hot-plugging, and without needing to change the underlying cloud infrastructures. Furthermore, such algorithms may consider the penalty cost (SLA violations) when scaling, and service downtime when adding and removing resources. Moreover, the author acknowledges that VM creation and resize delays are significant parameters when designing auto scaling systems and needs further investigations and analysis. Therefore, he conducts an empirical analysis on an IaaS cloud platform in Section 5.2 to understand how VM start and resize time delays vary depending on various aspects of cloud platforms.

5.2 Cloud Performance Analysis

Many previous work propose various auto-scaling algorithms and systems for vertical scaling. However, most of such work has been carried out assuming that the underlying cloud infrastructure may change configurations of VMs (rescale). However, the author has discovered that this is not a valid assumption, as if the resources of a VM are added or removed in real-time they are not affected until the VM is restarted [71] [20]. For above reasons, existing cloud service providers do not provide vertical scaling functions. The Chapter 6 describes the Mobile Cloud RAN (Radio Access Network) testbed that the author has developed. The prototype has been left with out of the box configurations of most software components (e.g. OpenStack) for making the evaluation results as generic as possible to all other existing platforms and cloud service providers. The following analysis on VM creation and resizing times has been carried out by repeating the experiments 60 times for reducing noise due to other unavoidable influences within the environment.

There are two basic functions that one may perform on VMs when scaling either horizontally or vertically. One may instantiate and terminate new VMs behind a load balancer, or one would use a resizing function that is provided by the underlying cloud platform (hypervisor). Moreover, In the author's case, the Kernel Virtual Machine (KVM) environment allows the user to instantiate and terminate VMs. In Figure 5.1 the author depicts how the VM instantiation time has been influenced by all other VM created within the same cloud environment. Author has applied a moving average function on the data for smoothing out the gathered results to highlight the increasing trend in data. The above graph clarifies the observations made by [147], where auto-scaling actions (horizontal) typically

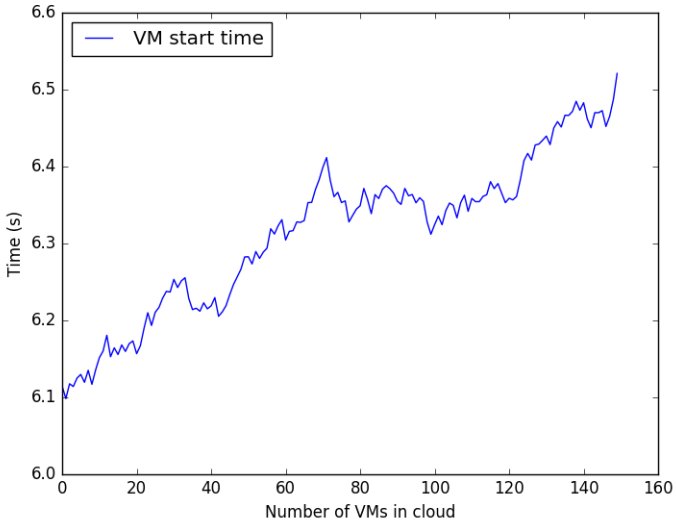

Figure 5.1: The VM start time as the number of instances increases in the cloud globally

get delays in orders of minutes on public cloud service providers (Amazon EC2, Azure and Rackspace). The above is mainly due to the increased request backlog and due to the time that it takes to assign a physical server to deploy a VM, then to move the VM image to it and get it fully operational. This further shows that instance start-time changes depending on the number of active VMs in the cloud at a given time. Moreover, when predicting the VM start times, knowledge on historical start delay times may help. The author further stresses that although the trend in data may be similar, the exact numbers at given points may change depending on the cloud environment. Therefore, one may expect future cloud service providers to provide such data to be used by users when auto-scaling.

The author categorises auto-scaling into two scenarios. 1) Resizing continuously; the resources are added or removed by one at each iteration, 2) the resources are added or removed by amounts other than 1 (e.g. +1. +2, +3, +4). For the sake of the discussion, the author calls the latter ”non-continuous”. The former method may be suitable for algorithms that decide *when* to scale (and later scales by adding/removing only one resource), and the latter method may be appropriate for algorithms that decide the *amount* of resources to be added or removed at a given time.

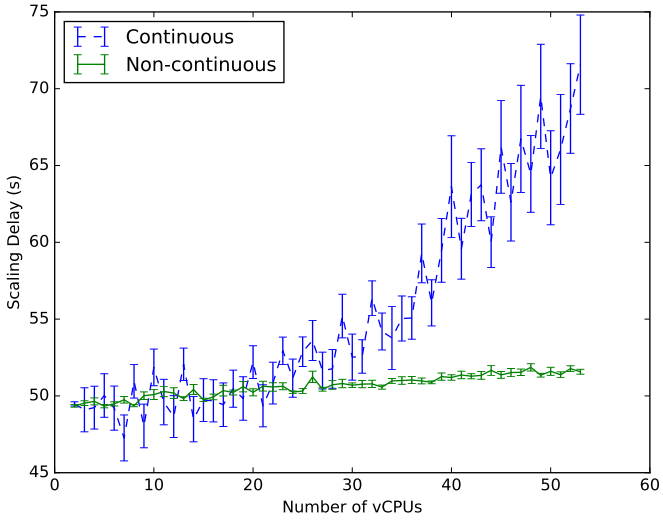

Figure 5.2: Mean CPU upscale delay as the size of the base VM increases. The standard error shows variations in results

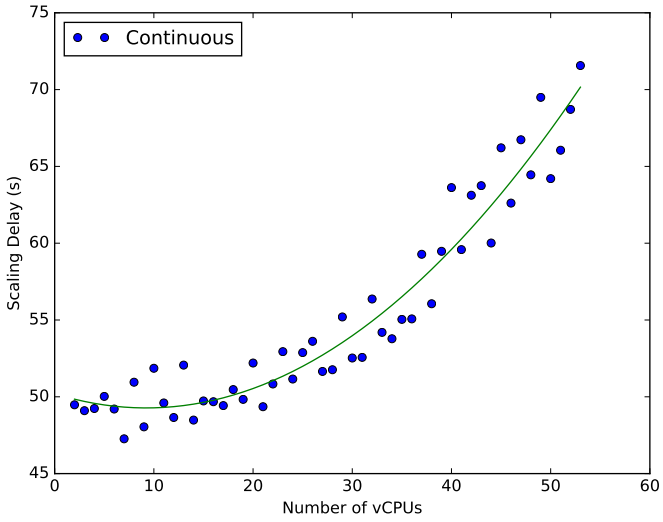

Figure 5.3: Second order polynomial function of mean CPU upscale time in continuous scenario

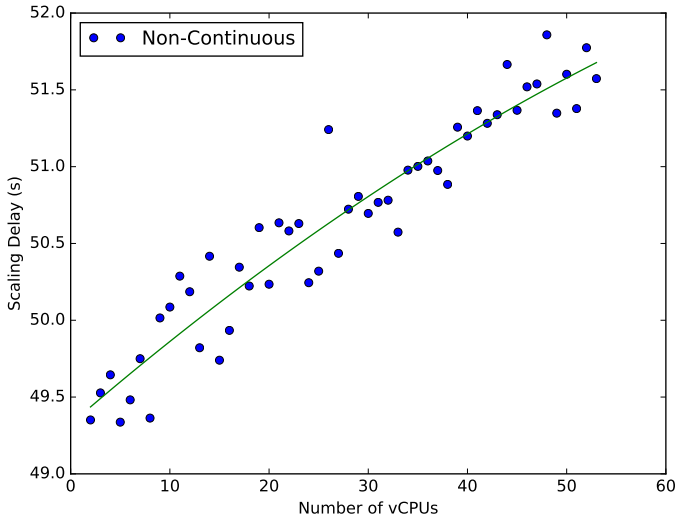

Figure 5.4: Second order polynomial function of mean CPU upscale time in non-continuous scenario

The author applies polynomial curve fitting [148] on the empirical data to further demonstrate how up and down scaling delay times vary across different types of resource and when different types of scaling are employed (continuously vs. non-continuously). Specifically, the author uses a function of the form in equation 5.1 for fitting the data, where M denotes the order of the polynomial function, given a training data set comprising N observations of x, where $x \equiv (x_1, \ldots, x_N)$ and corresponding observations $t \equiv (t_1, \ldots, t_N)$. The vector W contains the polynomial coefficients $w_0, \ldots, w_M$.

$$y(x, W) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j \tag{5.1}$$

The coefficients are calculated by fitting the polynomial to the provided scaling delay data by minimising the squared error $E(W)$, as shown in equation 5.2.

$$E(W) = \sum_{n=0}^{N} |y(x_n, W) - t_n|^2 \tag{5.2}$$

where t_n denotes the corresponding target values for x_n. It measures the misfit

Table 5.1: Polynomial coefficients of second order polynomial function for the scaling scenarios

	Coefficients		
Scaling Scenario	w_2	w_1	w_0
Figure 5.3	0.0109	0.2013	50.2
Figure 5.4	-0.0002161	0.05584	49.32
Figure 5.6	0.01358	0.3637	51.61
Figure 5.7	0.0003889	0.04552	66.85
Figure 5.9	-1.159e-05	0.1038	46.92
Figure 5.10	2.837e-05	0.008834	47.05
Figure 5.12	-0.002184	0.04266	49.07
Figure 5.13	0.007889	0.1701	50.31
Figure 5.15	0.1402	2.375	56.91
Figure 5.16	0.03394	0.6107	53.26

between the function $y(x, W)$, for any given value of W and scaling delay data points. The author has calculated the second order polynomial function using the least square polynomial fit [148] for the data, only for illustrating trends in data and to show differences in values, as shown in second order polynomial equation 5.3. Such a model may be adapted to the architecture proposed in Chapter 3 where the polynomial function is used for predicting resize delays in the mobile cloud controller (when making auto-scaling decisions), the order of the polynomial function should be chosen to fit the data best. For providing further insight into the analysis, the author provides the polynomial coefficients in Table 5.1 that may be used with the equation 5.3 for evaluating vertical scaling algorithms in future. In equation 5.3 the w_2, w_1, w_0 coefficients may be looked up from the Table 5.1, while x denotes the amount of resources to evaluate on.

$$y(x) = w_2 x^2 + w_1 x + w_0 \tag{5.3}$$

CPU resizing has been more often used than resizing other resources. The Figure 5.2 shows that the time varies when adding resources, depending on the vCPU amount it increases from (i.e. the number of vCPUs that the VM has before resizing), in continuous auto scaling scenario. Another observation that we may gather from the graph is that the VM resizing time increases as the size of the VM increases. One may observe that the change in results in the non-continuous scenario is much smaller as the scale of the VM increases, albeit there is still an

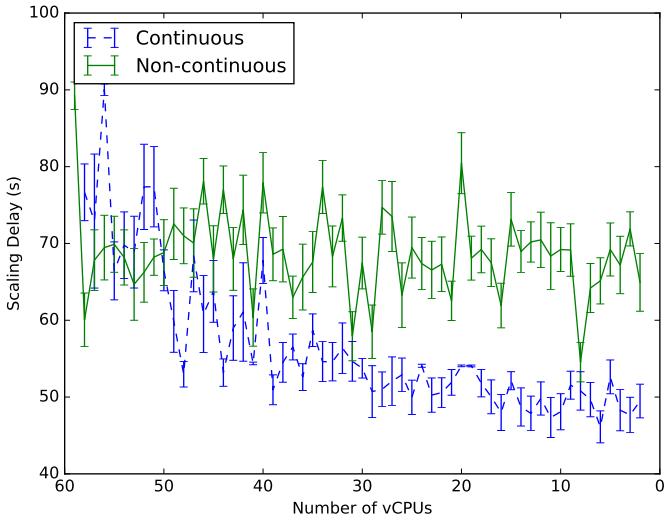

Figure 5.5: Mean CPU downscale delay as the size of the base VM increases. The standard error shows variations in results

increasing trend. The aforementioned statements are further clarified in Figure 5.3 and Figure 5.4 respectively. The both 5.3 and 5.4 figures show second order regression functions of results for continuous and non-continuous scenarios respectively, depicting that time delay increases as the VM size increases. The graph also shows the standard error of each point to show how results varied in the results.

Similarly to VM CPU upscaling, Figure 5.5, Figure 5.6 and Figure 5.7 shows continuous and non-continuous cloud performances of CPU downscaling. Specifically, the Figure 5.5 depicts a comparison of delay times when virtual CPUs are scaled down continuously and non-continuously. The error bars denote the standard error at each point to show the variations of gathered results. One may observe that this has an opposite trend (decreasing trend) to VM CPU upscaling delay as shown in Figure 5.2, where the delay time decreases as more CPUs are removed from the VM. One may conclude from the analysis that the resize time is greatly influenced by the size of the VM when scaling computing resources (i.e. the number of CPUs the VM has before resizing). Moreover, the Figure 5.6 and the Figure 5.7 shows second order polynomial function of both continuous and non-continuous mean CPU downscaling delay in cloud respectively. Further-

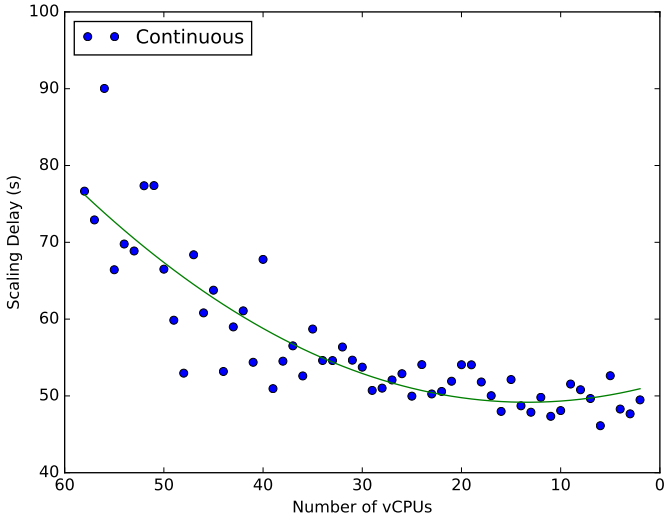

Figure 5.6: Second order polynomial function of mean CPU downscale time in continuous scenario

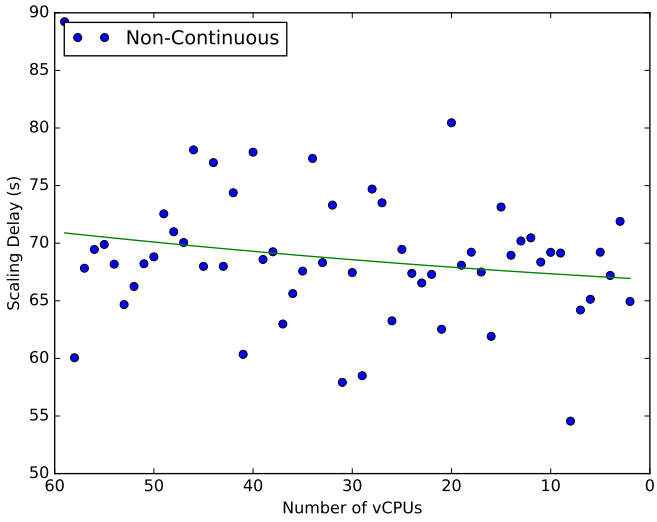

Figure 5.7: Second order polynomial function of mean CPU downscale time in non-continuous scenario

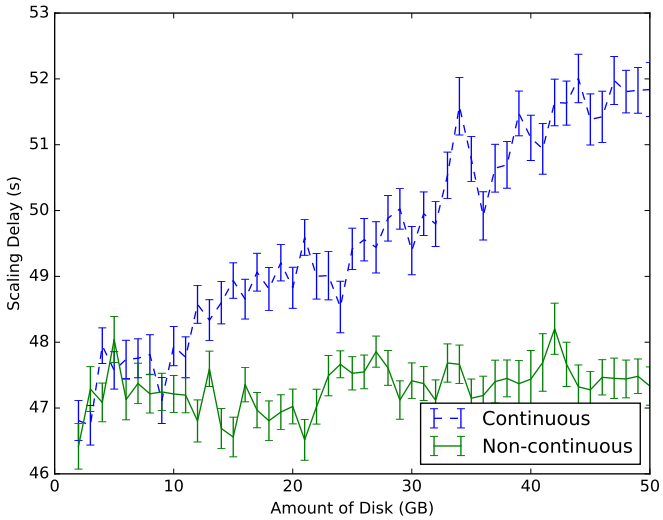

Figure 5.8: Mean disk upscale delay as the size of the base VM increases. The standard error shows variations (error) in data

more, the observations suggest that when designing auto-scaling algorithms that scale CPU resources, one must consider the time differences when scaling continuously and scaling non-continuously.

According to the author's experiments, the second most influential resource on the resize time delay is the VM disk size. However, in the used test environment, disk downscaling is not supported by the cloud framework (OpenStack with KVM as shown in Chapter 6), therefore only the upscaling results have presented. The Figure 5.8 shows an opposite trend to the CPU resize time delays in Figure 5.2. In this experiment, the non-continuous resizing time delays appear to higher than when resizing continuously. Moreover, the resize time does not show any significant increases or decreases as the amount of disk space increase in the non-continuous scenario. A second order polynomial regression analysis has been carried out in Figure 5.10 further clarifying the above observation.

It is clear that the resize delay time increases as the VM's disk size increases in the continuous VM scaling scenario as shown in Figure 5.9. Such an observation is expected, as when resizing some hypervisors (e.g. KVM) takes a snapshot of the running VM, then a new resized VM is created from the snapshot with changed configurations (often on a different compute node), finally deleting the

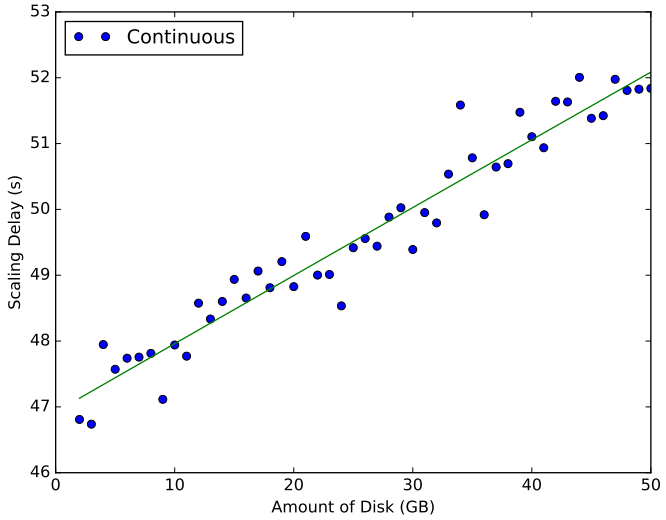

Figure 5.9: Second order polynomial function of mean disk upscale time in continuous scenario

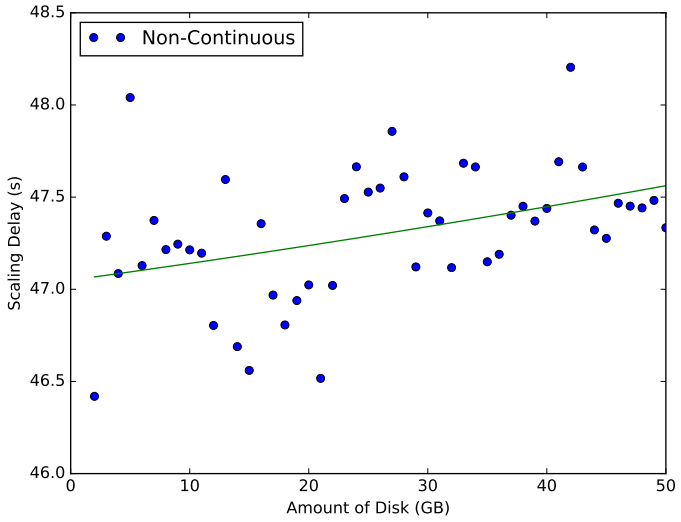

Figure 5.10: Second order polynomial function of mean disk upscale time in non-continuous scenario

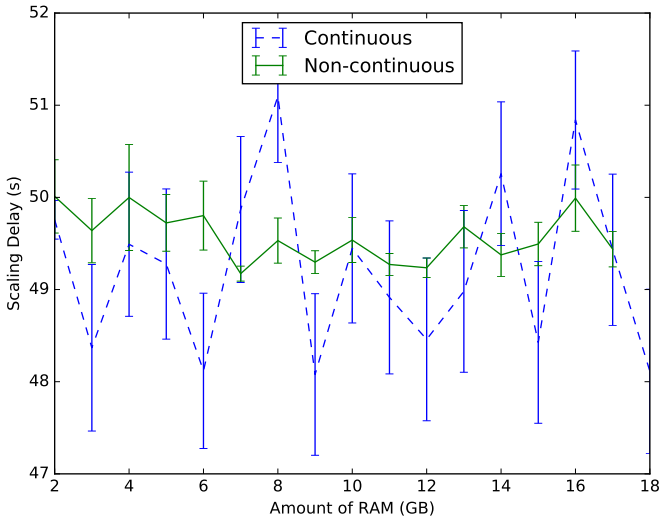

Figure 5.11: Mean RAM upscale delay as the size of the base VM increases. The standard error shows variations (error) in data

old VM. Whereas, when the author conducted the non-continuous experiment, he chose 1GB as the base VM disk size, and one may expect somewhat a similar time delay at each step when resizing as the disk size stays constant, although the VM is resized to a different disk size.

Despite all above clear trends in results, when analysing resizing time delays when resizing RAM resource on VMs, no distinctive trends were found in both continuous and non-continuous scenarios, as shown in Figure 5.11 for upscaling and 5.14 for RAM downscaling. However, this could be due to the fact that the test-bed's maximum memory resource limit is 18GB, and the sample amount is not large enough to see clear trends. The second order polynomial regression analyses show decreasing trends on both continuous and non-continuous scenarios, in Figure 5.12 and in 5.13, in continuous and non-continuous upscaling scenarios respectively. Moreover Figure 5.15 and Figure 5.16 shows a second order polynomial regression analysis of both continuous and non-continuous downscaling of RAM resources.

From the lessons learned from above empirical VM resize performance analysis, the author proved that careful attention should be paid to the time delays when resizing CPU and disk resources of VMs for real-time applications. The

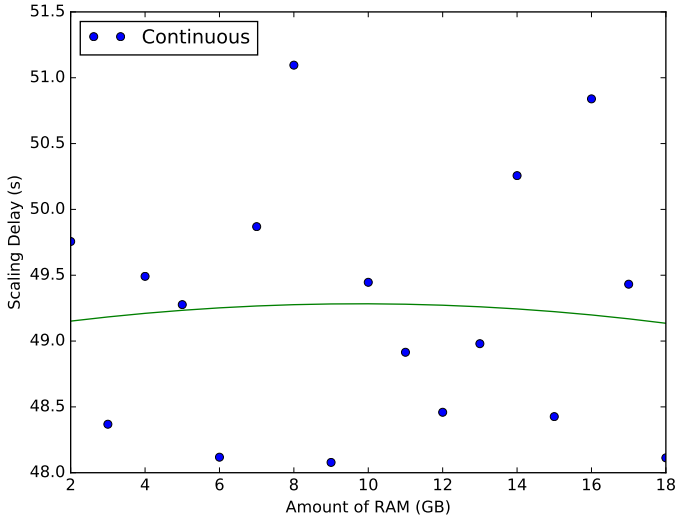

Figure 5.12: Second order polynomial function of mean RAM upscale time in continuous scenario

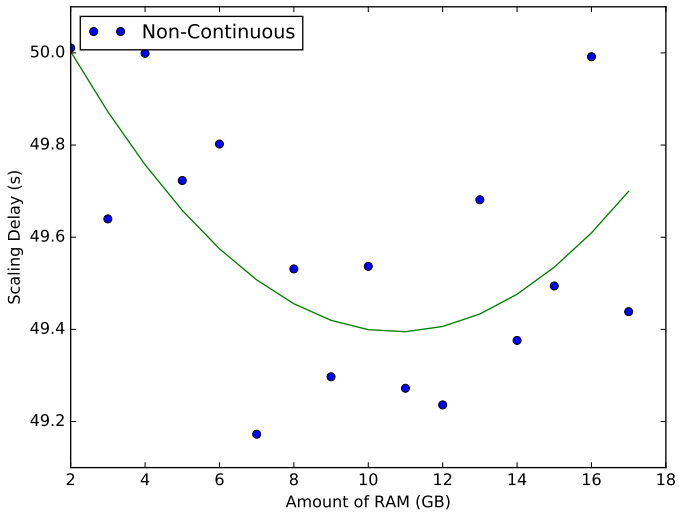

Figure 5.13: Second order polynomial function of mean RAM upscale time in non-continuous scenario

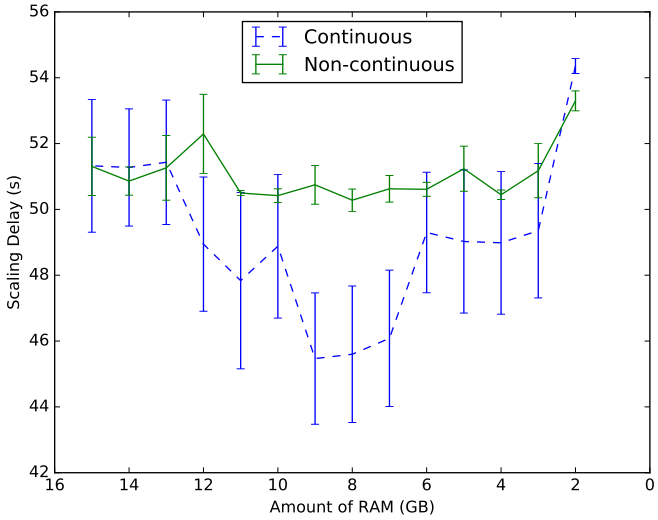

Figure 5.14: Mean RAM downscale delay as the size of the base VM increases. The standard error shows variations (error) in data

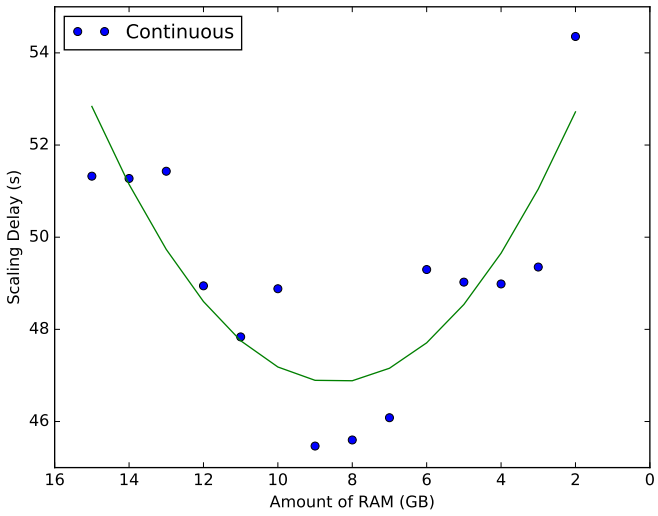

Figure 5.15: Second order polynomial function of mean RAM downscale time in continuous scenario

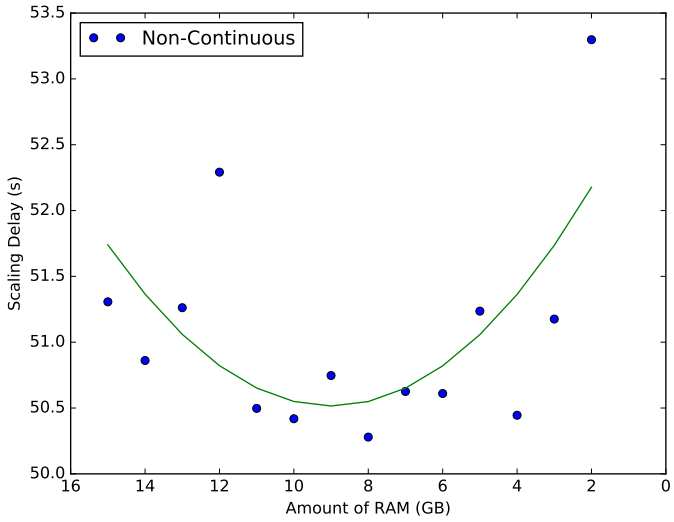

Figure 5.16: Second order polynomial function of mean RAM downscale time in non-continuous scenario

auto-scale delay trends may vary depending on the underlying technologies (e.g. virtualization and physical resource management) that are deployed. The analysis brings awareness to the community that auto-scaling delays depend on the auto-scaling types (upscaling, downscaling, continuous and non-continuous). The author stresses that especially when designing auto-scaling algorithms, the changes in time delays when resizing CPU and disk resources in various conditions of the VMs itself (VM size) and of the environment (total VM count in the cloud) that they are hosted in, should be considered. The author has discovered that it is important to use both the knowledge of the VM itself and of the cloud environment together at the same time in future vertical scaling algorithms.

5.3 Summary

This chapter presented an extensive performance analysis of cloud vertical auto-scaling. The analysis has been performed on performance data that have been gathered from the cloud prototype implementation presented in Chapter 6. The study has demonstrated that VM start time increases as the size (global number of VMs) of the cloud itself increase. The author conducted the auto-scaling perfor-

mance analysis for CPU, memory and disk resources. Significant changes in CPU and disk scaling time trends have been observed. Finally, the author discussed that vertically scaling CPU and disk resources may not be suitable to be performed in real-time due to the large delay, for delay-constrained applications.

Chapter 6

Prototype

Cloud Radio Access Networks (C-RAN) [149][150] is a novel deployment concept for future radio access networks (RAN) which takes the powerful cloud technologies to use. C-RAN is a centralised RAN architecture which processes the signals of all base stations on a shared pool of compute resources. The base stations of the existing architecture have been replaced by Remote Radio Heads (RRHs) and digital processing components called Base Band Unit (BBU), which have been moved into a central location. Whereas, the conventional RAN's baseband processing units are located locally in each cell. The centralisation of the computational resources have significant benefits; improved load balancing increased energy consumption by exploiting the load variations, resource sharing, joint and cooperative resource scheduling, saving the operating expenses due to centralised maintenance. Researchers have done extensive work on RAN and MCC independently in the literature. However, there has been very limited amount of work done on the joint study of C-RAN and Mobile Cloud Computing (MCC) for next generation cellular networks.

MCC brings rich and powerful properties of cloud computing to mobile devices. Compute offloading is one of the techniques that was created to achieve the energy efficiency of mobile devices, by offloading compute-intensive tasks to a remote cloud to be executed. Excising compute offloading systems take coarse network information and assume the remote computing resource is a public cloud or a cloudlet that is in close vicinity to the mobile user.

The author has set up a test environment for evaluating the proposed architecture, use cases and its applications. Figure 6.1 depicts the design of the testbed. There are two USRP N210 and one X300 have been set up as RRHs of Amarisoft LTE 100 [151] and OpenAirInterface (OAI) [152] software base stations respec-

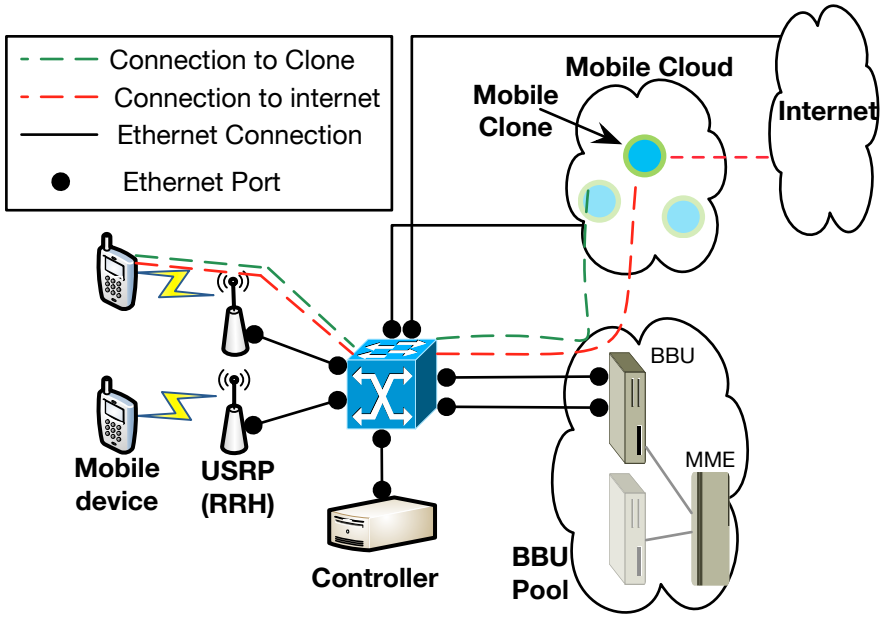

Figure 6.1: C-RAN with Mobile Cloud testbed

tively. All nodes in the network are connected via a Gigabit Ethernet switch. The soft base stations are deployed on a Dell PowerEdge R210 rack server. OpenStack with Kernel Virtual Machine (KVM) has been deployed as the cloud framework that hosts the mobile clones running Android-x86 operating system. In the current setting, the wireless bandwidth of the base stations has been set to 5 MHz. The BBUs are connected to its Mobility Management Entity (MME) via its S1-MME links using S1 Application Protocol (S1AP) that provides signalling between E-UTRAN and the evolved packet core (EPC). Host sFlow and sFlow-RT [153], monitoring and analytical tools, has been deployed on the controller as a part of the Resource Monitor for monitoring resources in the mobile cloud. The author has developed a monitoring module and a dashboard for monitoring wireless resources in C-RAN as described in Section 6.2.1.

6.1 User Equipment Side

The author uses Thinkair [7] offloading framework for offloading computationally intensive tasks from the connected UEs to the Clones in the Mobile Cloud.

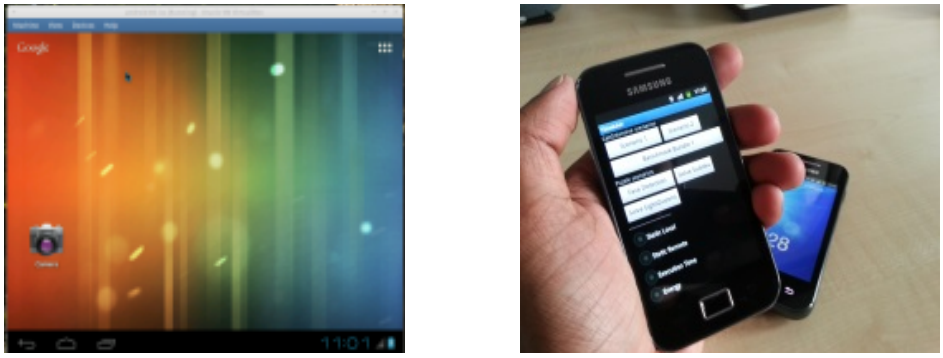

Figure 6.2: Thinkair offloading framework deployed on Android UEs (right side) and on the Clone (left side)

The Thinkair server side is installed on the cloudified Android x86 clone (as further discussed in Section 6.2.2), while the client side is installed on all Android offloading UEs. The subfigure in the right side of the figure 6.2 depicts a mobile device with Thinkair client installed, and the sub figure on the left side shows a screenshot of the Thinkair installed clone on the OpenStack-based mobile cloud.

6.2 Infrastructure Side

The operator's network infrastructure side consists of two parts as described in Chapter 3, the wireless infrastructure (i.e. C-RAN) and the mobile cloud. The Section 6.2.1 highlights the author's contributions to the wireless infrastructure and Section 6.2.2 includes the contribution that relate to the mobile cloud.

6.2.1 C-RAN

The wireless environment of the infrastructure has been deployed using Amarisoft LTE 100 and OAI software solutions that also comes with wireless core network components (i.e. EPC) included. Therefore, these two platforms may be used to emulate a fully fledged mobile operators network. Amarisoft LTE 100 is a commercial solution, and OAI is a fully open-source project. Although both solutions offer basic required functionalities of the operator's network, they follow independent development roadmaps and have implemented their extra functionalities (e.g. Amarisoft LTE 100 API).

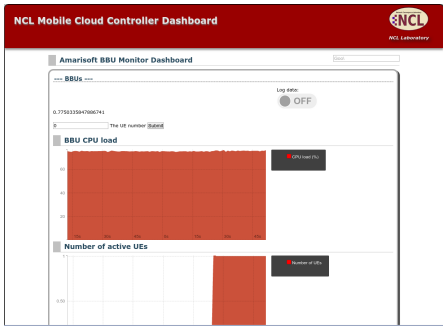

Figure 6.3: A screenshot of Amarisoft wireless resource monitor dashboard running on a web browser, showing CPU utilisation and the number of connected UEs

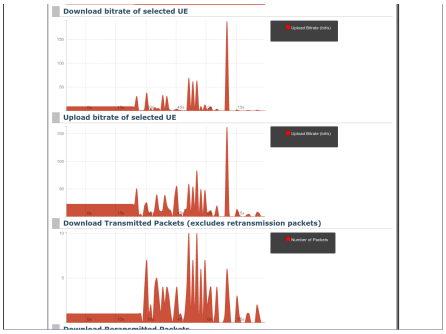

Figure 6.4: A screenshot of Amarisoft wireless resource monitor dashboard running on a web browser, showing the download bitrate, upload bitrate and the number of transmitted packets in downlink

Although OAI provides a very basic monitoring interface, it is not useful for logging or observing trends in monitoring information. Amarisoft does not provide any monitoring functionality out of the box. Therefore both solutions lack inbuilt fully functioning monitoring functionalities. Specifically, they both lack detailed and user-friendly monitoring dashboards for observing monitoring information of the changing parameters when carrying out experiments. Therefore, the author has developed his own monitoring module that includes a useful dashboard that displays changes in various parameters at real-time as a part of the Resource Monitor (as described in Chapter 3) in the proposed controller. Currently, the dashboard only designed to work with Amarisoft LTE 100 software as shown in figure 6.3 and figure 6.4. The web page is developed using HTML, and the graphs are drawn using JavaScript based Rickshaw [154] interactive time-series graph, which is based on D3 [155] document manipulating library. The dashboard is hosted on the Apache web server on the controller. The Resource Monitor on controller gets monitoring information from the connected BBU(s) every second and passes through to the dashboard.

Although OAI may not have all required functionalities, one may develop extra functionalities or alter existing code to adapt it to their requirements. The author has implemented a light monitoring functionality into OAI that sends monitoring information periodically (every second) to the Resource Monitor on the controller for further analysis and decision making, using the protocol described

in Chapter 3. Moreover, the Resource Monitor receives monitoring information from the MAC scheduler of the OAI [156] [157]. For demonstrating the functions of the implemented MAC scheduler, the author has gathered observational results from OAI, received from the MAC scheduler. The implemented MAC layer of OAI calculates the required amount of resource blocks for the user before resource allocation based on MAC layer parameters such as the buffer size [156]. However, the MAC layer may or may not be able to allocate the required amount of resources based on the available resources [156] [157].

The conducted experiment contains two UEs connected to the Mobile Cloud over the OAI wireless network. The User0 streams a video from the mobile cloud while User1 accesses Facebook servers on the internet using it's Android application. To make the analysis fair, the author has considered only the allocation iterations that allocate resources to both UEs at the same time on the downlink. This analysis shows probabilities of getting a specific number of resource blocks at a given allocation round, comparing the required and the allocated number of resource blocks. The figure 6.5 shows the probability mass function of allocated resource block amounts. The figure indicates that User0 gets larger amounts of resource blocks more often than User1 and that User1 often get a smaller amount of resources allocated more often than the User0. This is because User0 transfers a larger amount of data continuously when streaming video, but User1 only transfers small web requests and responds from Facebook servers. This claim may be further clarified by the Gaussian Kernel Density Estimation (KDE) that is conducted in figure 6.7 to further demonstrate the trend. The figure 6.6 shows that User0 gets a larger amount of resources allocated overall when both UEs are transferring data at the same time.

Figure 6.8 and figure 6.10 shows that User0 requires larger amounts of resource blocks at a given time for the same above reason, in comparison to the required resource blocks of User1. It is evident in figure 6.9 that User0 requires much more resources than User1 due to the larger number of data that is transferred over the network.

Although one may require a certain number of resource blocks as mentioned above, the MAC scheduler may or may not be able to allocate the required amount at all times. Figure 6.11 and figure 6.14 shows that for both User0 and User1, the required and allocated resource amount probabilities vary. Moreover, this claim may be further clarified by the Gaussian Kernel Density Estimation (KDE) that is conducted in figure 6.13 and figure 6.16 to further demonstrate the trends. However, figure 6.12 and figure 6.15 depict that the MAC scheduler has a higher probability that it may not be able to allocate the required amount of resources

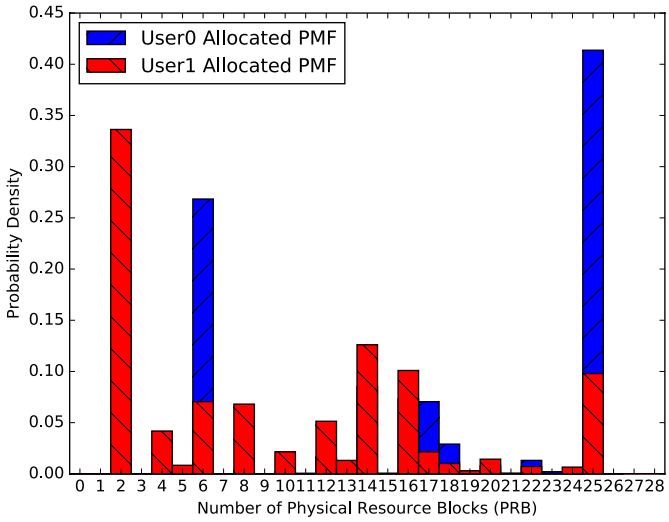

Figure 6.5: A comparison of probabilities of both User 1 and User 2 getting a specific number of PRBs allocated by the MAC schedular, when both users are transferring at the same time.

sometimes, as the resources are shared with the other user.

6.2.2 Mobile Cloud

Openstack has been deployed as the cloud framework for orchestrating virtual resources in the cloud. A modified version of the Android OS that runs on X86 processor architecture has been installed in the Clone that the server side of Thinkair runs on. Android-x86 is an open source project that has an active community that releases x86 versions of new Android versions. However, the original version of above OS could not be directly installed on OpenStack virtual machines due to lack of modules and applications that are required for cloudification. Cloudification involves adding kernel modules to the OS for utilising virtual resources that are presented by the VM, to automate and for dynamically tailoring the VM at runtime. With the help of the Android-x86 project community, Android-x86 OS was modified to work with OpenStack virtual machines.

A hypervisor is a software and at times is a firmware that is used to create virtual machines. The computer on which the hypervisor runs on is defined as the host device. Hypervisors virtualize physical hardware resources and present

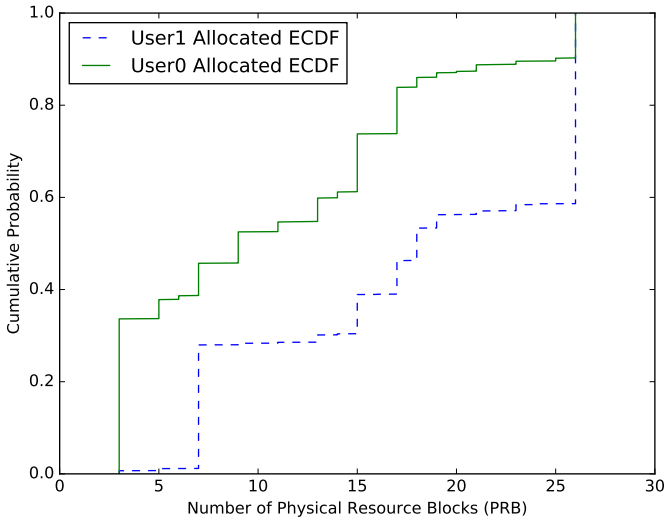

Figure 6.6: A comparison of the cumulative probabilities of User 1 and User 0 getting a specific number of PRBs allocated by the MAC schedular, when are transferring at the same time.

them to the guest operating system that is running in the virtual machine. Such guest operating systems running on a virtualised system need to communicate and cooperate with the underlying hypervisors for using virtualized resources. Kernel-based Virtual Machine (KVM) [131] is the hypervisor that the author uses that in the prototype. KVM turns the Linux Kernel into a hypervisor. To be able to run KVM, it requires a processor with hardware virtualization extension, which enables full efficient virtualisation for the guest OS. KVM also falls into the Type 2 hypervisors, which is the category of hypervisors that run on a conventional operating system rather than on bare-metal.

To be able to create Clones on OpenStack to offload tasks, a virtual machine that runs android-x86 has to be created. At the time that the experiments were done by the author, he was not able to create OpenStack virtual machines with android-x86 images that came out of the box, due to its lack of modules which enable paravirtualization, in the Linux kernel of the android-x86 OS. Virtio [158] is a set of standards for disk and network virtualisation that are required to be installed on OpenStack VM instance that runs on OpenStack, which enables paravirtualization for the VMs. Also, except for the kernel modules, the source code

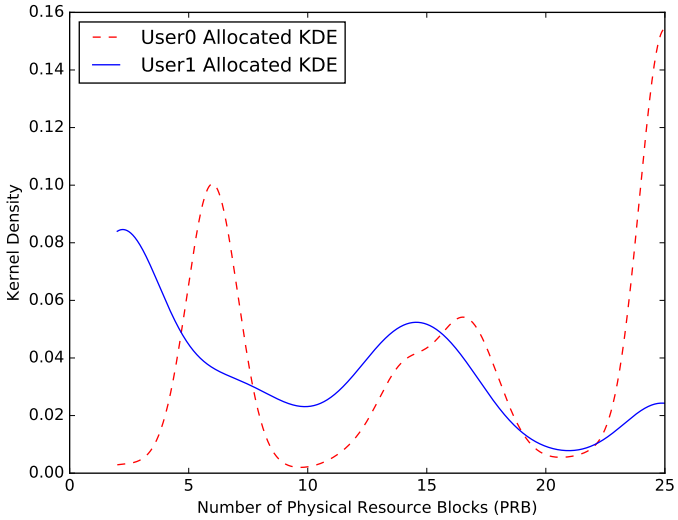

Figure 6.7: A comparison of the gaussian Kernel Density Estimations (KDE) showing the trends of User 1 and User 0 getting a specific number of PRBs allocated by the MAC schedular, when both are transferring at the same time.

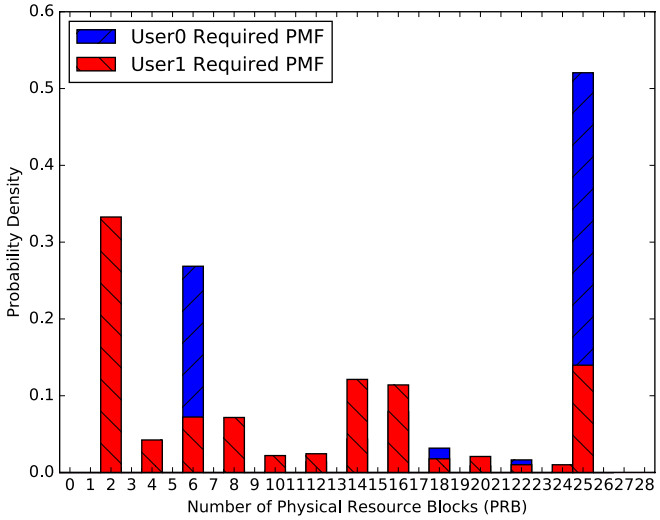

Figure 6.8: A comparison of probabilities of both User 1 and User 2 requiring a specific number of PRBs in the MAC schedular, when both users are transferring at the same time.

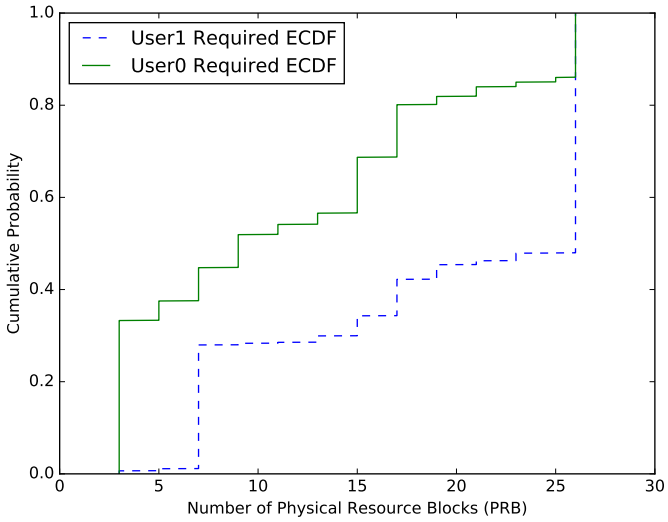

Figure 6.9: A comparison of the cumulative probabilities of User 1 and User 0 requiring a specific number of PRBs in the MAC schedular, when are transferring at the same time.

of the Androidx86 OS has been modified to get the Clone fully functioning in Cloud. The modifications that have been made to the Android x86 OS, during the process of cloudification, are presented in Appendix A. However, the modifications are suggested only for Android version 4.4 and have not been tested with other versions of the operating system.

A ready-made OpenStack VM image with all above changes has been made available to the general public by the author [1]. Nonetheless, it has been brought to authors attention that modification mentioned above has now been adapted and included into the main branch of the Android-x86 project, hence now one may run the Android-x86 OS on OpenStack VMs without performing most of above modifications.

Video Streaming with Clone

Video streaming over RTMP has been widely used today for delivering live video streams as well as video on demand (VOD) services. In a conventional system,

[1] An Androidx86 VM for OpenStack was developed during the study; https://sourceforge.net/projects/androidx86-openstack (over 4600 downloads, *by 14.02.2017*)

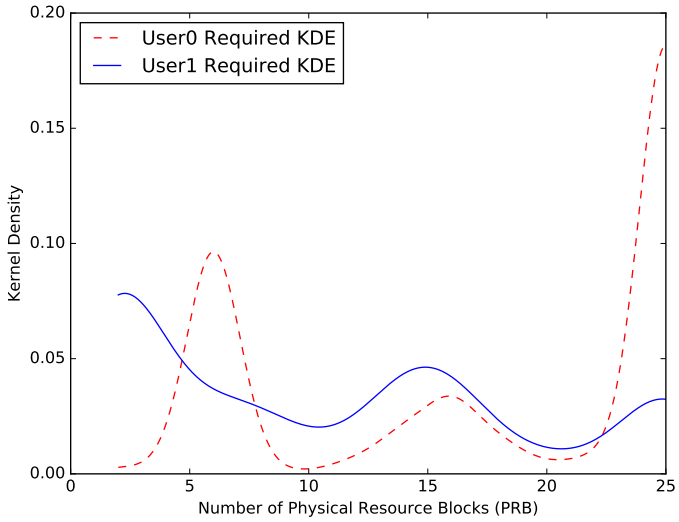

Figure 6.10: A comparison of the gaussian Kernel Density Estimations (KDE) showing the trends of User 1 and User 0 requiring a specific number of PRBs allocated in the MAC schedular, when both are transferring at the same time.

a video stream is streamed from a remote server to the client/mobile device directly. Wireless networks in comparison to wired networks are unpredictable and unstable. The stability of wireless networks depends on parameters such as the interference, the distance from the mobile device to the connected base station and congestion. Due to instabilities in the wireless network packet loss and packet retransmissions occur; in turn, it leads to increase in traffic in the backhaul network.

In the proposed architecture, a mobile cloud has been placed next to the BBU pool in C-RAN, as a part of the wireless network infrastructure. Similarly to the scenarios discussed above, one clone per each mobile subscriber will be assigned. The mobile clone acts as a local buffer point such that, the mobile device gets the video stream through its clone. When issues in the wireless network such as packet loss occur, the mobile clone handles retransmissions instead of the remote video source. In occasions where there is packet loss, RTMP packet retransmission is done by the clone instead of the remote video service provider. The above therefore reduces the retransmission packet delay and reduces traffic in the backhaul network. We may conclude that it separates wireless related issues from the backhaul networks. As shown in Figure 6.17, the user uses RTMP streaming for

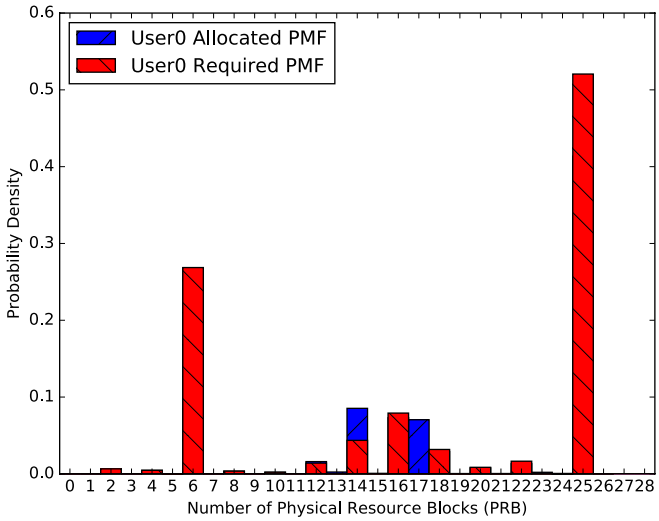

Figure 6.11: A comparison of the probabilities of User 0 requiring a specific number of PRBs vs the probability of getting a specific number of PRBs allocated by the MAC schedular, when User 1 and User 0 are transferring at the same time.

getting video content.

It is assumed that the mobile device requests a video from the video service provider through its clone. The video source responds by initiating an RTMP stream to the RTMP server running in the clone. Once the clone receives the stream from the remote video service provider, it buffers and forwards the stream back to the mobile device (possibly after some transcoding if needed).

When a video has been requested the remote video server start a video encoder, which might encode the video to the requested video quality or copy the frames as it is to the RTMP application that is setup on Nginx server. Then, it forwards the stream to the user's clone. Ones the clone gets the stream, the mobile RTMP client on the mobile device may connect to it and start streaming the video.

For performing experiments, the author has used a PC with a Quad-core Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz processing power and 4G RAM as the remote video source and the encoder. The encoder and the video server are hosted on the same machine. The clone is of OpenStack type flavour tiny 1, which consists of 1 CPU and 512MB of RAM. The video that was being used for streaming is a 480p video with 1508 kb/s bitrate. To emulate packet loss that may occur

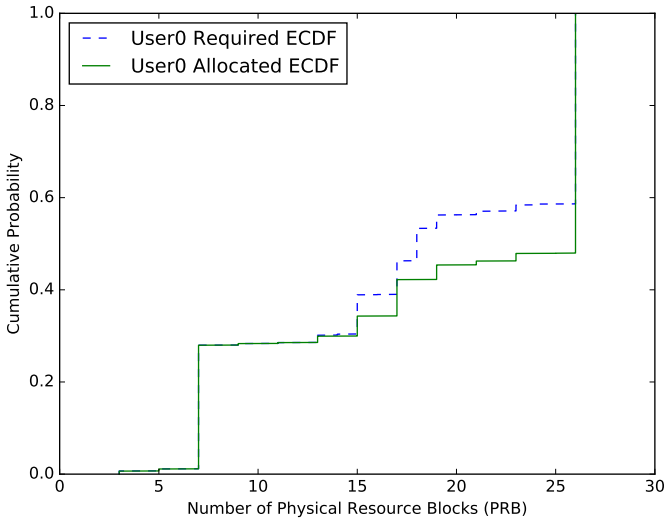

Figure 6.12: A comparison of the cumulative probabilities of User 0 requiring a specific number of PRBs vs the probability of getting a specific number of PRBs allocated by the MAC schedular, when User 1 and User 0 are transferring at the same time.

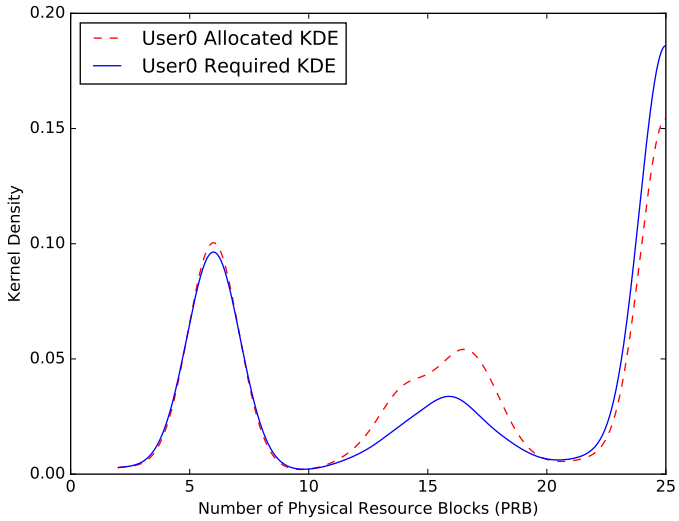

Figure 6.13: A comparison of the gaussian Kernel Density Estimations (KDE) showing the trends of User 0 requiring a specific number of PRBs vs getting a specific number of PRBs allocated by the MAC schedular, when User 1 and User 0 are transferring at the same time.

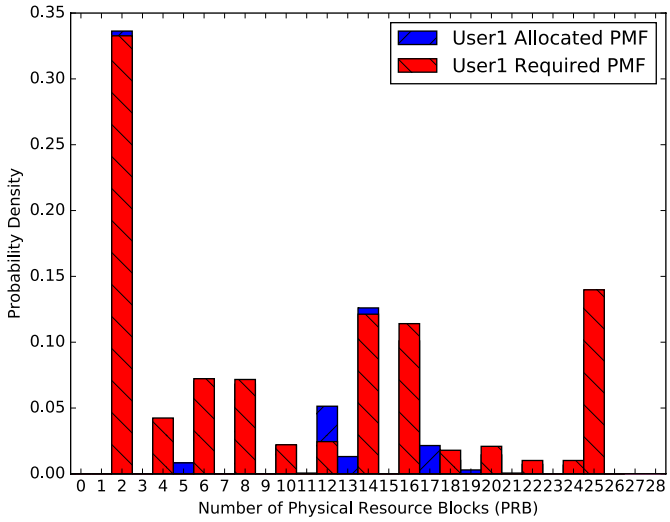

Figure 6.14: A comparison of the probabilities of User 1 requesting a specific number of PRBs vs the probability of getting a specific number of PRBs allocated by the MAC schedular, when User 1 and User 0 are transferring at the same time.

due to wireless related issues such as congestion or interference in the wireless network, netem [159] a network emulation tool has been used to introduce packet loss into the wireless network in the BBU.

The figure 6.18 shows how the clone has reduced TCP retransmission traffic in the C-RAN backhaul network. In the conventional video-streaming scenario without a clone in place, one may observe that as the packet loss rate has been increased the retransmission traffic has also been increased as a result of TCP reliable stream delivery service. However with the proposed architecture, the clone that sits next to the BBU instead of the remote video server does retransmission of TCP packets, so there is almost no retransmission traffic in the backhaul network. Thence, the bandwidth of the backhaul network has been saved, as extra data other than the original video stream has not been transferred.

6.3 Summary

This chapter introduced the prototype implementation of the system that has been proposed in Chapter 3. The prototype integrates the mobile cloud and the C-RAN

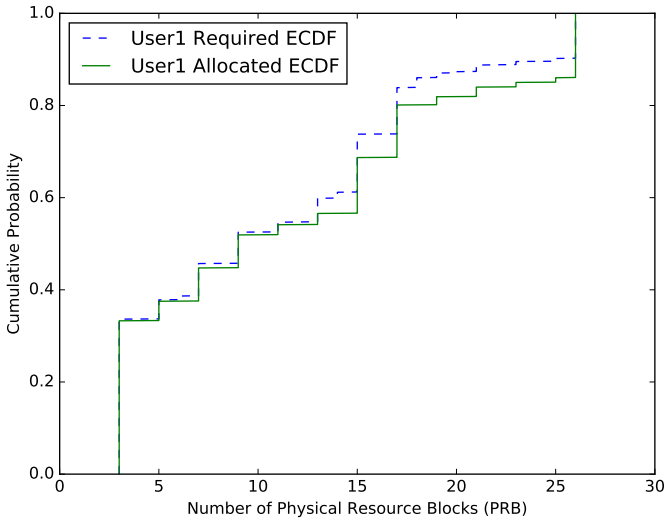

Figure 6.15: A comparison of the cumulative probabilities of User 1 requiring a specific number of PRBs vs the probability of getting a specific number of PRBs allocated by the MAC schedular, when User 1 and User 0 are transferring at the same time.

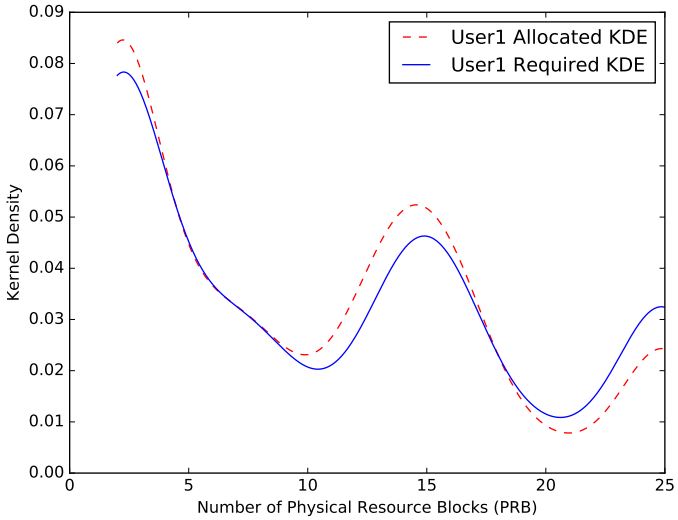

Figure 6.16: A comparison of the gaussian Kernel Density Estimations (KDE) showing the trends of User 1 requiring a specific number of PRBs vs getting a specific number of PRBs allocated by the MAC schedular, when User 1 and User 0 are transferring at the same time.

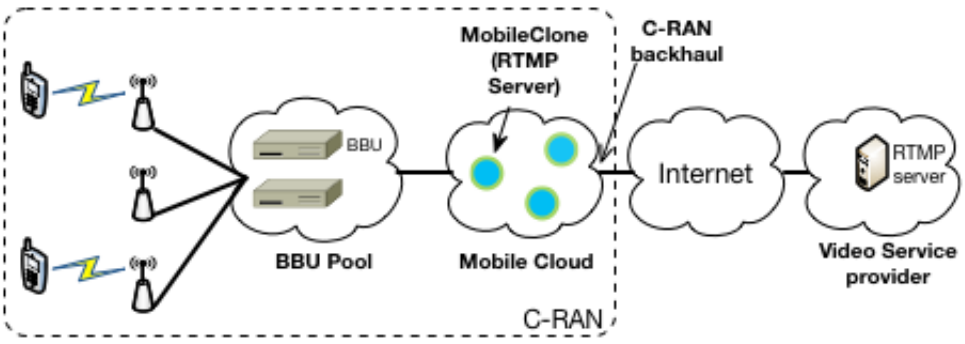

Figure 6.17: C-RAN testbed architecture for video streaming with RTMP

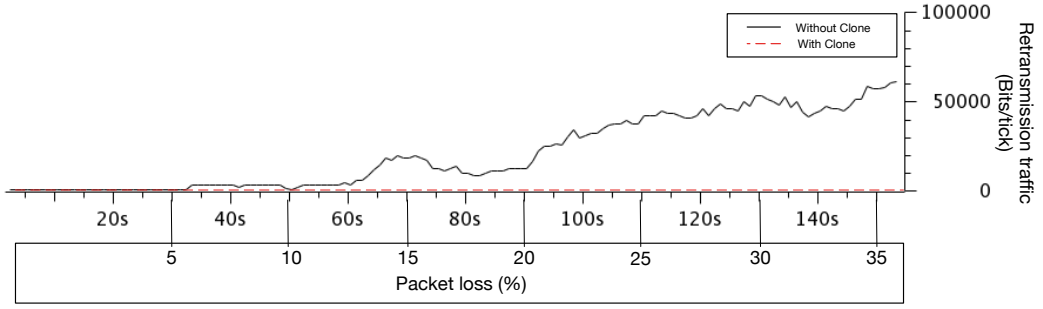

Figure 6.18: Video retransmission traffic comparison in C-RAN backhaul. With and without Clone

through a centralised controller. The controller manages both computing, i.e., mobile cloud, and communication, i.e., C-RAN, resources cooperatively and/or jointly. Thinkair framework has been adopted as the task offloading framework, while Amarisoft and OpenAirInterface have been used to emulate the wireless edge and core networks. OpenStack cloud platform hosts the modified Android VMs.

Chapter 7

Conclusion

The resource-constrained UEs (CPU, GPU, memory, storage capacity, and battery lifetime) have become widely popular among users, mobile software and hardware developers. The increased number of computing intensive mobile applications have driven a dramatic surge in developing new paradigms to handle computation intensive tasks [1]. MCC [2] is an exciting new paradigm that offers task offloading capabilities to mobile devices by allowing the devices to send computationally intensive mobile tasks to a remote resourceful location to be executed. Various MCC offloading frameworks have been developed to tackle above issue, such as Thinkair [7]. The mobile offloading workloads generate data intensive network traffic due to transferred user data and mobile code via the wireless network. As predicted [10], the mobile traffic will increase exponentially by the year 2020, and mobile offloading traffic may become one of the influential factors of such growth. Therefore, the future cellular infrastructures must be flexible and reconfigurable to support simplified deployment and management of network resources.

The BBU pool in C-RAN [11] may be realised by the VMs in data centres, and the centralised baseband processing enables BBU to be dynamically configured and shared on demand [13]. Therefore, C-RAN becomes an ideal wireless network architecture for supporting mobile task offloading in future cellular networks. Due to the delay constrained nature of offloading tasks, it is beneficial to bring computing resources as close to the user as possible [14]. Consequently, the above reduces offloading latency when computationally intensive tasks are offloaded within the radio access network. Due to the reconfigurability, flexibility and the ease of deploying services in C-RAN, the computing resource (mobile cloud) may be introduced to the C-RAN network for reducing offloading delay,

and for introducing a new kind of service (mobile cloud on top of the cellular network services) for future mobile network operators.

The newly introduced computing resources need to be managed accordingly for utilising both network and computing resources efficiently in the operator's network. Therefore an architecture and a protocol for managing both computing and communication resources for mobile task offloading need to be developed. Although managing both computing and communication resources collectively is an interesting area to investigate into, in this thesis the author focuses on managing resources in the mobile cloud for delay constrained workloads such as mobile compute task offloading.

Cloud auto-scaling [18] [20], a technique that allows dynamic reconfigurations of cloud resources (CPU, RAM, disk), for efficient cloud resource management per application demand. Cloud vertical scaling adds and removes resources from existing VMs [19]. Cloud computing systems are renowned for being very complex application systems consisting of various sub systems (hypervisors, host OS, guest OS, virtual networks) [18]. Moreover, due to its complexity and other dependent sub-modules, it is challenging to predict its performance for managing mobile cloud resources efficiently. Especially when cloud resources are used for delay-constrained applications, it is important to understand the trends in cloud auto-scaling performances and its behaviours under various scaling scenarios for managing resources timely.

The architecture that is proposed in Chapter 3, have introduced a new kind of service into mobile network operator's network. Computationally intensive mobile tasks are offloaded to a resource-rich computing resource that is placed alongside the BBU pool in C-RAN, for the benefit of both mobile subscribers/UEs (e.g. energy efficiency) and the mobile network operator (e.g. new premium price plans for new computing services). A new logical controller that receives instantaneous monitoring information from both computing and communication sides and makes efficient resource management decisions on both C-RAN and mobile cloud for mobile task offloading. Such management controlling signals are transferred using a high bandwidth and low latency wired networks. A protocol that uses a simple unified packet header for both resource management and task offloading has been proposed; that utilises resource monitoring and management components in the controller. Two examples showing the working procedures has been demonstrated to illustrate the functions of the protocol further. In Chapter 4 the author proposed an energy-efficient and network-aware offloading algorithm that considers an offloading scenario where mobile cloud resources are placed both at the edge (next to Wifi access points (cloudlets)) of the network as well

as in a public cloud (clones). The architectural changes that need to be done to existing mobile task offloading frameworks to support above offloading scenario have also been discussed in Chapter 3.

The author considered an MCC scenario called Cloudlet+Clone in Chapter 3, containing a middle layer called a cloudlet layer. This cloudlet layer sits between mobile devices and their traditional cloud infrastructure or clones. Cloudlets are deployed next to WiFi APs and serve as a localised service point in close proximity to mobile devices to improve the performance of mobile cloud services regarding response time. An offloading algorithm for deciding *Whether* and *Where* to offload has been employed on top of the new architecture. The decision-making takes into account the availability of two types of wireless networks, WiFi and cellular, with the aim of saving battery life for mobile devices while satisfying the response time constraints of applications. The author demonstrated the efficiency of the proposed architecture by comparing it with conventional offloading architectures in simulations.

When mobile computing tasks are offloaded to the mobile cloud, the mobile cloud resources must be allocated efficiently while meeting delay constraints. Due to the lack of studies on vertical scaling and scaling delay times in literature [20], In Chapter 5 the author carried out an empirical analysis of cloud vertical auto-scaling performance. The author conducted an analysis on scaling up and scaling down performances. The analysis showed that auto-scaling performances of all disk, CPU and RAM resources vary when scaling vertically. The results also revealed that scaling time delay depends on the amount of resources that are added or removed from the VM at each step. Furthermore, the author has approximated a second order polynomial function for the performance results gathered, and its coefficients have been provided for giving an insight into vertical scaling performances. The above may help develop and evaluate future cloud vertical scaling algorithms. One may conclude from the auto-scaling delay analysis that it is necessary to analyse auto-scaling performances of each cloud platform due to complex nature of today's cloud systems. Moreover, knowing scaling delay time trends help to make effective auto-scaling decisions.

The author proposed a new architecture for mobile cloud in Chapter 3, a prototype of which has been developed in Chapter 6. The implementation of the C-RAN has been carried out using both Amarisoft LTE 100 [151] commercial software base station and the OpenAirInterface [152] open-source software base station. The mobile cloud side consists of an OpenStack-based cloud running Androidx86 mobile VMs (clones) [22] developed by the author. The implementation of the mobile cloud controller includes a resource monitor module that gets instan-

taneous monitoring information from both C-RAN and the mobile cloud. Due to lack of a user-friendly monitoring dashboard to observe changes in parameters in real-time, a monitoring dashboard has been developed. A compute manager has also been implemented that may perform cloud auto-scaling operations on the mobile cloud depending on the workload patterns detected by the resource monitor modules.

The author's aim for future work is to integrate the proposed coarse-grained offloading algorithm with a VM auto-scaling algorithm from the lessons learned from the vertical auto-scaling empirical analysis. The used energy model will be refined, and the overhead for collecting energy consumption information will also be evaluated. For allocating mobile cloud resources for offloading workloads, an auto-scaling algorithm will be designed, which takes the scaling delays into consideration. The designed algorithm will be implemented and evaluated on the prototype in Chapter 6. Once the auto-scaling algorithm allocates resources for computing resources, the offloading framework (the offloading algorithm) and the resource allocation modules will be integrated together, through the implemented mobile cloud controller. Then a system integration analysis and a task offloading performance analysis will be carried out to demonstrate how the mobile cloud allocates computing resources to the incoming workloads while also meeting delay constraints (e.g. SLA/SLO). The proposed design of the unified protocol will be refined accordingly and comprehensively implemented, then will be used when performing aforementioned mobile task offloading and resource allocation procedures. The monitoring information that the resource monitor receives from the software base stations will be improved, and missing monitoring functionalities will be implemented (e.g. in OpenAirInterface). A further investigation into wireless resource management (e.g. BBU allocation, BBU life-cycle management) will be carried out later for cooperative and dynamic resource management in C-RAN with the mobile cloud.

Bibliography

[1] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?," *Computer*, vol. 43, pp. 51 –56, april 2010.

[2] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

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

[4] "Amazon elastic compute cloud (amazon ec2)." https://aws.amazon.com/ec2. Online, accessed: 2016-05-09.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, EuroSys '11, (New York, NY, USA), pp. 301–314, ACM, 2011.

[6] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, (New York, NY, USA), pp. 49–62, ACM, 2010.

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

[8] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *Mobile Computing, Applications, and Services*, pp. 59–79, Springer, 2012.

[9] E. Ahmed, A. Gani, M. K. Khan, R. Buyya, and S. U. Khan, "Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges," *Journal of Network and Computer Applications*, vol. 52, pp. 154 – 172, 2015.

[10] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang, "What will 5g be?," *IEEE Journal on Selected Areas in Communications*, vol. 32, pp. 1065–1082, June 2014.

[11] C. Mobile, "C-ran: the road towards green ran," *White Paper, ver*, vol. 2, 2011.

[12] J. Wu, "Green wireless communications: from concept to reality [industry perspectives]," *IEEE Wireless Communications*, vol. 19, pp. 4–5, August 2012.

[13] J. Tang, W. P. Tay, and T. Q. S. Quek, "Cross-layer resource allocation with elastic service scaling in cloud radio access network," *IEEE Transactions on Wireless Communications*, vol. 14, pp. 5068–5081, Sept 2015.

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

[15] W. H. Chin, Z. Fan, and R. Haines, "Emerging technologies and research challenges for 5g wireless networks," *IEEE Wireless Communications*, vol. 21, pp. 106–112, April 2014.

[16] C. J. Bernardos, A. de la Oliva, P. Serrano, A. Banchs, L. M. Contreras, H. Jin, and J. C. Zuniga, "An architecture for software defined wireless networking," *IEEE Wireless Communications*, vol. 21, pp. 52–61, June 2014.

[17] H. Ali-Ahmad, C. Cicconetti, A. de la Oliva, V. Mancuso, M. R. Sama, P. Seite, and S. Shanmugalingam, "An sdn-based network architecture for extremely dense wireless networks," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pp. 1–7, Nov 2013.

[18] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surv.*, vol. 47, pp. 63:1–63:33, July 2015.

[19] S. Dutta, S. Gera, A. Verma, and B. Viswanathan, "Smartscale: Automatic application scaling in enterprise clouds," in *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, (Washington, DC, USA), pp. 221–228, IEEE Computer Society, 2012.

[20] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *J. Grid Comput.*, vol. 12, pp. 559–592, Dec. 2014.

[21] "H2020 icirrus project - d4.3 mobile cloud networking and virtual mobile." http://www.icirrus-5gnet.eu. Online, accessed: 2016-09-01.

[22] "androidx86-openstack." https://sourceforge.net/projects/androidx86-openstack. Online, accessed: 2016-09-01.

[23] L. Yang, J. Cao, S. Tang, T. Li, and A. T. S. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing.," in *IEEE CLOUD* (R. Chang, ed.), pp. 794–802, IEEE, 2012.

[24] S. Kosta, C. Perta, J. Stefa, P. Hui, and A. Mei, "Clone2clone (c2c): Enable peer-to-peer networking of smartphones on the cloud," *T-Labs, Deutsche Telekom, Tech. Rep. TR-SK032012AM*, 2012.

[25] K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow, "Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing," in *Proceedings of the 17th annual international conference on Mobile computing and networking*, MobiCom '11, (New York, NY, USA), pp. 73–84, ACM, 2011.

[26] S. Kosta, V. C. Perta, J. Stefa, P. Hui, and A. Mei, "Clonedoc: Exploiting the cloud to leverage secure group collaboration mechanisms for smartphones," in *Proc. of IEEE INFOCOM*, vol. 2013, 2013.

[27] M. Barbera, S. Kosta, J. Stefa, P. Hui, and A. Mei, "Cloudshield: Efficient anti-malware smartphone patching with a p2p network on the cloud," in

Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on, pp. 50–56, 2012.

[28] M. V. Barbera, S. Kosta, A. Mei, V. C. Perta, and J. Stefa, "Cdroid: Towards a cloud-integrated mobile operating system," in *Proc. of IEEE INFOCOM*, vol. 2013, 2013.

[29] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang, "Toward cloud-based vehicular networks with efficient resource management," *Network, IEEE*, vol. 27, pp. 48–55, September 2013.

[30] D. T. Hoang, D. Niyato, and P. Wang, "Optimal admission control policy for mobile cloud computing hotspot with cloudlet," in *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, pp. 3145–3149, April 2012.

[31] D. Niyato, P. Wang, E. Hossain, W. Saad, and Z. Han, "Game theoretic modeling of cooperation among service providers in mobile cloud computing environments," in *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, pp. 3128 –3133, april 2012.

[32] A.-D. Nguyen, P. Senac, and V. Ramiro, "How mobility increases mobile cloud computing processing capacity," in *Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on*, pp. 50 –55, nov. 2011.

[33] Y. Cai, F. Yu, and S. Bu, "Cloud radio access networks (c-ran) in mobile cloud computing systems," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pp. 369–374, April 2014.

[34] G. P. Perrucci, F. H. P. Fitzek, and J. Widmer, "Survey on energy consumption entities on the smartphone platform," in *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, pp. 1–6, 2011.

[35] L. Zhang, B. Tiwana, R. Dick, Z. Qian, Z. Mao, Z. Wang, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pp. 105–114, 2010.

[36] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, (New York, NY, USA), pp. 280–293, ACM, 2009.

[37] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "Characterizing radio resource allocation for 3g networks," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, (New York, NY, USA), pp. 137–150, ACM, 2010.

[38] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Profiling resource usage for mobile applications: A cross-layer approach," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, (New York, NY, USA), pp. 321–334, ACM, 2011.

[39] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan, "Bartendr: A practical approach to energy-aware cellular data scheduling," in *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, MobiCom '10, (New York, NY, USA), pp. 85–96, ACM, 2010.

[40] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, (New York, NY, USA), pp. 317–328, ACM, 2012.

[41] J. Manweiler and R. Choudhury, "Avoiding the rush hours: Wifi energy management via traffic isolation," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 5, pp. 739–752, 2012.

[42] A. Rahmati and L. Zhong, "Context-based network estimation for energy-efficient ubiquitous wireless connectivity," *Mobile Computing, IEEE Transactions on*, vol. 10, no. 1, pp. 54–66, 2011.

[43] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, (New York, NY, USA), pp. 335–348, ACM, 2011.

[44] V. Namboodiri and T. Ghose, "To cloud or not to cloud: A mobile device perspective on energy consumption of applications," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, pp. 1–9, 2012.

[45] T.-Y. Lin, T.-A. Lin, C.-H. Hsu, and C.-T. King, "Context-aware decision engine for mobile cloud offloading," in *Wireless Communications and Networking Conference Workshops (WCNCW), 2013 IEEE*, pp. 111–116, 2013.

[46] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g lte networks," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, (New York, NY, USA), pp. 225–238, ACM, 2012.

[47] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi, "Using bandwidth data to make computation offloading decisions," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8, 2008.

[48] L. Feeney and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1548–1557 vol.3, 2001.

[49] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *INFOCOM, 2012 Proceedings IEEE*, pp. 2716–2720, 2012.

[50] M. Jan, M. Rabaey, and B. Nikolic, "Digital integrated circuits," 1996.

[51] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud," in *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pp. 784–791, July 2012.

[52] H. Wu, D. Huang, and S. Bouzefrane, "Making offloading decisions resistant to network unavailability for mobile cloud collaboration," in *Collab-*

orative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on, pp. 168–177, Oct 2013.

[53] S. Ou, K. Yang, and L. Hu, "Cross: A combined routing and surrogate selection algorithm for pervasive service offloading in mobile ad hoc environments," in *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pp. 720–725, Nov 2007.

[54] K. Sinha and M. Kulkarni, "Techniques for fine-grained, multi-site computation offloading," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pp. 184–194, 2011.

[55] S. Adler, "The slashdot effect: an analysis of three internet publications," *Linux Gazette*, vol. 38, p. 2, 1999.

[56] T. Chen and R. Bahsoon, "Toward a smarter cloud: Self-aware autoscaling of cloud configurations and resources," 2015.

[57] "Amazon ec2 auto scaling." https://aws.amazon.com/autoscaling. Online, accessed: 2016-07-29.

[58] F. L. Ferraris, D. Franceschelli, M. P. Gioiosa, D. Lucia, D. Ardagna, E. D. Nitto, and T. Sharif, "Evaluating the auto scaling performance of flexiscale and amazon ec2 clouds," in *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*, pp. 423–429, Sept 2012.

[59] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck, "From data center resource allocation to control theory and back," in *2010 IEEE 3rd International Conference on Cloud Computing*, pp. 410–417, July 2010.

[60] M. Maurer, I. Brandic, and R. Sakellariou, *Enacting SLAs in Clouds Using Rules*, pp. 455–466. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[61] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pp. 644–651, May 2012.

[62] M. Z. Hasan, E. Magana, A. Clemm, L. Tucker, and S. L. D. Gudreddi, "Integrated and autonomic cloud resource scaling," in *2012 IEEE Network Operations and Management Symposium*, pp. 1327–1334, April 2012.

[63] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–12, Nov 2011.

[64] "Rightscale. set up autoscaling using voting tags.." http://support.rightscale.com/12-Guides/Dashboard_Users_Guide/Manage/Arrays/Actions/Set_up_Autoscaling_using_Voting_Tags/index.html. Online, accessed: 2016-07-29.

[65] B. Simmons, H. Ghanbari, M. Litoiu, and G. Iszlai, "Managing a saas application in the cloud using paas policy sets and a strategy-tree," in *2011 7th International Conference on Network and Service Management*, pp. 1–5, Oct 2011.

[66] J. Kupferman, J. Silverman, P. Jara, and J. Browne, "Scaling into the cloud," *CS270-advanced operating systems*, 2009.

[67] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on*, pp. 281–286, Oct 2009.

[68] S. R. Seelam, P. Dettori, P. Westerink, and B. B. Yang, "Polyglot application auto scaling service for platform as a service cloud," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pp. 84–91, March 2015.

[69] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing: Challenges and opportunities," in *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*, ACDC '09, (New York, NY, USA), pp. 13–18, ACM, 2009.

[70] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proceedings of the 8th International Workshop on Middleware for Grids,*

Clouds and e-Science, MGC '10, (New York, NY, USA), pp. 4:1–4:6, ACM, 2010.

[71] W. H. Liao, S. C. Kuai, and Y. R. Leau, "Auto-scaling strategy for amazon web services in cloud computing," in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pp. 1059–1064, Dec 2015.

[72] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: towards a fully automated workflow," in *Seventh International Conference on Autonomic and Autonomous Systems, ICAS 2011*, pp. 67–74, IEEE, May 2011. MoVe INT LIP6.

[73] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "Vconf: A reinforcement learning approach to virtual machines auto-configuration," in *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, (New York, NY, USA), pp. 137–146, ACM, 2009.

[74] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[75] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in *Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, ICAC '06, (Washington, DC, USA), pp. 65–73, IEEE Computer Society, 2006.

[76] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *2012 IEEE Network Operations and Management Symposium*, pp. 204–212, April 2012.

[77] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning servers in the application tier for e-commerce systems," *ACM Trans. Internet Technol.*, vol. 7, Feb. 2007.

[78] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Trans. Auton. Adapt. Syst.*, vol. 3, pp. 1:1–1:39, Mar. 2008.

[79] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *Proceedings of the Fourth International Conference on Autonomic Computing*, ICAC '07, (Washington, DC, USA), pp. 27–, IEEE Computer Society, 2007.

[80] D. A. Menasce, L. W. Dowdy, and V. A. F. Almeida, *Performance by Design: Computer Capacity Planning By Example*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[81] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, (New York, NY, USA), pp. 1–10, ACM, 2010.

[82] S. M. Park and M. Humphrey, "Self-tuning virtual machines for predictable escience," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pp. 356–363, May 2009.

[83] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, (New York, NY, USA), pp. 13–26, ACM, 2009.

[84] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A multi-model framework to implement self-managing control systems for qos management," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, (New York, NY, USA), pp. 218–227, ACM, 2011.

[85] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters," in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, HotCloud'09, (Berkeley, CA, USA), USENIX Association, 2009.

[86] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, (New York, NY, USA), pp. 117–126, ACM, 2009.

[87] G. Evensen, *Data assimilation: the ensemble Kalman filter*. Springer Science & Business Media, 2009.

[88] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "On the use of fuzzy modeling in virtualized data center management," in *Fourth International Conference on Autonomic Computing (ICAC'07)*, pp. 25–25, June 2007.

[89] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. A. B. Fortes, "Fuzzy modeling based resource management for virtualized database systems," in *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 32–42, July 2011.

[90] L. Wang, J. Xu, M. Zhao, and J. Fortes, "Adaptive virtual resource management with fuzzy model predictive control," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, (New York, NY, USA), pp. 191–192, ACM, 2011.

[91] P. Lama and X. Zhou, "Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee," in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 151–160, Aug 2010.

[92] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *2010 International Conference on Network and Service Management*, pp. 9–16, Oct 2010.

[93] J. Huang, C. Li, and J. Yu, "Resource prediction based on double exponential smoothing in cloud computing," in *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pp. 2056–2060, April 2012.

[94] H. Mi, H. Wang, G. Yin, Y. Zhou, D. Shi, and L. Yuan, "Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers," in *Services Computing (SCC), 2010 IEEE International Conference on*, pp. 514–521, July 2010.

[95] K. H. Yeung and C. W. Szeto, "On the modeling of www request arrivals," in *Parallel Processing, 1999. Proceedings. 1999 International Workshops on*, pp. 248–253, 1999.

[96] M. N. A. H. Khan, Y. Liu, H. Alipour, and S. Singh, "Modeling the autoscaling operations in cloud with time series data," in *Reliable Distributed Systems Workshop (SRDSW), 2015 IEEE 34th Symposium on*, pp. 7–12, Sept 2015.

[97] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, pp. 1–14, 2011.

[98] C. C. Chen, S. J. Chen, F. Yin, and W. J. Wang, "Efficient hybriding autoscaling for openstack platforms," in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pp. 1079–1085, Dec 2015.

[99] A. Y. Nikravesh, S. A. Ajila, and C. H. Lung, "Towards an autonomic auto-scaling prediction system for cloud resource provisioning," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 35–45, May 2015.

[100] A. A. D. P. Souza and M. A. S. Netto, "Using application data for sla-aware auto-scaling in cloud environments," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, pp. 252–255, Oct 2015.

[101] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pp. 500–507, July 2011.

[102] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, (Berkeley, CA, USA), pp. 337–350, USENIX Association, 2008.

[103] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," in *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '03, (New York, NY, USA), pp. 300–301, ACM, 2003.

[104] S. Khatua, A. Ghosh, and N. Mukherjee, "Optimizing the utilization of virtual resources in cloud environment," in *2010 IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems*, pp. 82–87, Sept 2010.

[105] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Gener. Comput. Syst.*, vol. 28, pp. 155–162, Jan. 2012.

[106] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Gener. Comput. Syst.*, vol. 27, pp. 871–879, June 2011.

[107] E. Caron, F. Desprez, and A. Muresan, "Pattern matching based forecast of non-periodic repetitive behavior for cloud clients," *J. Grid Comput.*, vol. 9, pp. 49–64, Mar. 2011.

[108] E. Caron, F. Desprez, and A. Muresan, *Forecasting for Cloud computing on-demand resources based on pattern matching*. PhD thesis, INRIA, 2010.

[109] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, (New York, NY, USA), pp. 5:1–5:14, ACM, 2011.

[110] C. Qu, R. N. Calheiros, and R. Buyya, "A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances," *J. Netw. Comput. Appl.*, vol. 65, pp. 167–180, Apr. 2016.

[111] O. Biran, D. H. Lorenz, E. Reichstein, and A. Weit, "Cloud services brokering for elastic workloads," in *Proceedings of the 8th ACM International Systems and Storage Conference*, SYSTOR '15, (New York, NY, USA), pp. 24:1–24:1, ACM, 2015.

[112] E. D. Coninck, T. Verbelen, B. Vankeirsbilck, S. Bohez, P. Simoens, and B. Dhoedt, "Dynamic auto-scaling and scheduling of deadline constrained service workloads on iaas clouds," *Journal of Systems and Software*, vol. 118, pp. 101 – 114, 2016.

[113] R. Poddar, A. Vishnoi, and V. Mann, "Haven: Holistic load balancing and auto scaling in the cloud," in *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*, pp. 1–8, Jan 2015.

[114] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pp. 58–65, May 2013.

[115] E. Campos, R. Matos, P. Maciel, A. Pereira, and F. Souza, "Stochastic modeling of auto scaling mechanism in private clouds for supporting performance tuning," in *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*, pp. 109–114, Oct 2015.

[116] M. Sides, A. Bremler-Barr, and E. Rosensweig, "Yo-yo attack: Vulnerability in auto-scaling mechanism," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), pp. 103–104, ACM, 2015.

[117] Z. A. Baig, S. M. Sait, and F. Binbeshr, "Controlled access to cloud resources for mitigating economic denial of sustainability (edos) attacks," *Computer Networks*, vol. 97, pp. 31 – 47, 2016.

[118] Y. W. Ahn, A. M. K. Cheng, J. Baek, M. Jo, and H. H. Chen, "An auto-scaling mechanism for virtual resources to support mobile, pervasive, real-time healthcare applications in cloud computing," *IEEE Network*, vol. 27, pp. 62–68, September 2013.

[119] N. Kumar, K. Kaur, A. Jindal, and J. J. Rodrigues, "Providing healthcare services on-the-fly using multi-player cooperation game theory in internet of vehicles (iov) environment," *Digital Communications and Networks*, vol. 1, no. 3, pp. 191 – 203, 2015.

[120] V. Vijayakumar, V. Neelanarayanan, M. Parekh, and B. Saleena, "Big data, cloud and computing challenges designing a cloud based framework for healthcare system and applying clustering techniques for region wise diagnosis," *Procedia Computer Science*, vol. 50, pp. 537 – 542, 2015.

[121] B. Xu, L. Xu, H. Cai, and L. Jiang, "Architecture of m-health monitoring system based on cloud computing for elderly homes application," in *Enterprise Systems Conference (ES), 2014*, pp. 45–50, Aug 2014.

[122] S. Poorejbari and H. Vahdat-Nejad, "An introduction to cloud-based pervasive healthcare systems," in *Proceedings of the 3rd International Conference on Context-Aware Systems and Applications*, ICCASA '14, (ICST, Brussels, Belgium, Belgium), pp. 173–178, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[123] T. C. Lin, M. Y. Pai, C. L. Chen, and C. C. Chen, "Load-balanced cloud service interface for the hiba mobile cloud environment," in *Consumer Electronics - Taiwan (ICCE-TW), 2015 IEEE International Conference on*, pp. 360–361, June 2015.

[124] S. Qureshi, T. Ahmad, K. Rafique, and S. ul islam, "Mobile cloud computing as future for mobile applications - implementation methods and challenging issues," in *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, pp. 467 –471, sept. 2011.

[125] D. Chang, G. Xu, L. Hu, and K. Yang, "A network-aware virtual machine placement algorithm in mobile cloud computing environment," in *Wireless Communications and Networking Conference Workshops (WCNCW), 2013 IEEE*, pp. 117–122, 2013.

[126] C. M. S. Magurawalage, K. Yang, L. Hu, and J. Zhang, "Energy-efficient and network-aware offloading algorithm for mobile cloud computing," *Computer Networks*, vol. 74, Part B, pp. 22 – 33, 2014. Special Issue on Mobile Computing for Content/Service-Oriented Networking Architecture.

[127] "Ieee standard for local and metropolitan area networks part 16: Air interface for fixed and mobile broadband wireless access systems amendment 2: Physical and medium access control layers for combined fixed and mobile operation in licensed bands and corrigendum 1," *IEEE Std 802.16e-2005 and IEEE Std 802.16-2004/Cor 1-2005 (Amendment and Corrigendum to IEEE Std 802.16-2004)*, pp. 01–822, 2006.

[128] J. Power and Associates, "2012 u.s. wireless smartphone and traditional mobile phone satisfaction studies," 2012.

[129] S. Abolfazli, Z. Sanaei, M. Alizadeh, A. Gani, and F. Xia, "An experimental analysis on cloud-based mobile augmentation in mobile cloud computing," *Consumer Electronics, IEEE Transactions on*, vol. 60, pp. 146–154, February 2014.

[130] H. Niksic, "Gnu wget," *available from the master GNU archive site prep. ai. mit. edu, and its mirrors*, 1998.

[131] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, pp. 225–230, 2007.

[132] R. Hat, "libvirt: The virtualization api." Online, accessed: 2016-05-09.

[133] N. Gray, "Performance of java middleware-java rmi, jaxrpc, and corba," 2005.

[134] H. Wu, Q. Wang, and K. Wolter, "Methods of cloud-path selection for offloading in mobile cloud computing systems," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pp. 443–448, 2012.

[135] Y. Wu, G. Min, and A. Al-Dubai, "A new analytical model for multi-hop cognitive radio networks," *Wireless Communications, IEEE Transactions on*, vol. 11, pp. 1643–1648, May 2012.

[136] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, Mar. 2014.

[137] S. C. Cripps, *RF Power Amplifiers for Wireless Communications, Second Edition (Artech House Microwave Library (Hardcover))*. Norwood, MA, USA: Artech House, Inc., 2006.

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

[139] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[140] R. Buyya, R. Ranjan, and R. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *High Performance Computing Simulation, 2009. HPCS '09. International Conference on*, pp. 1–11, 2009.

[141] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 149–163, Oct. 2003.

[142] W. Yuan and K. Nahrstedt, "Energy-efficient cpu scheduling for multimedia applications," *ACM Trans. on Computer Syst*, vol. 24, pp. 292–331, 2005.

[143] D. Puthal, B. P. S. Sahoo, S. Mishra, and S. Swain, "Cloud computing features, issues, and challenges: A big picture," in *Computational Intelligence and Networks (CINE), 2015 International Conference on*, pp. 116–123, Jan 2015.

[144] J. Rao, X. Bu, C. Z. Xu, and K. Wang, "A distributed self-learning approach for elastic provisioning of virtualized cloud resources," in *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 45–54, July 2011.

[145] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, "Elastic application container: A lightweight approach for cloud resource provisioning," in *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, pp. 15–22, March 2012.

[146] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control," in *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date*, ScienceCloud '12, (New York, NY, USA), pp. 31–40, ACM, 2012.

[147] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pp. 423–430, June 2012.

[148] Y. Anzai, *Pattern Recognition & Machine Learning*. Elsevier, 2012.

[149] K. Chen and R. Duan, "C-ran: The road towards green ran, white paper version 2.5, china mobile research institute, oct. 2011."

[150] S. Bhaumik, S. P. Chandrabose, M. K. Jataprolu, G. Kumar, A. Muralidhar, P. Polakos, V. Srinivasan, and T. Woo, "Cloudiq: A framework for processing base stations in a data center," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, (New York, NY, USA), pp. 125–136, ACM, 2012.

[151] "Amarisoft lte 100." http://www.amarisoft.com/index.php?p=amarilte. Online, accessed: 2016-09-01.

[152] "Openairinterface - 5g software alliance for democratising wireless innovation." http://www.openairinterface.org. Online, accessed: 2016-09-01.

[153] "sflow real-time." http://sflow-rt.com. Online, accessed: 2016-07-29.

[154] "Rickshaw - a javascript toolkit for creating interactive time series graphs." http://code.shutterstock.com/rickshaw. Online, accessed: 2016-07-29.

[155] "D3 data driven documents." https://d3js.org. Online, accessed: 2016-07-29.

[156] A. Bhamri, N. Nikaein, F. Kaltenberger, J. Hamalainen, and R. Knopp, "Three-step iterative scheduler for qos provisioning to users running multiple services in parallel," in *2014 IEEE 79th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, May 2014.

[157] A. Bhamri, N. Nikaein, F. Kaltenberger, J. Hmlinen, and R. Knopp, "Preprocessor for mac-layer scheduler to efficiently manage buffer in modern wireless networks," in *2014 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1544–1549, April 2014.

[158] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[159] S. Hemminger *et al.*, "Network emulation with netem," in *Linux conf au*, pp. 18–23, Citeseer, 2005.

Appendices

Appendix A

Cloudifying Android OS: Android on OpenStack

The author performed following modifications sequentially to create a Mobile Clone that runs on OpenStack clouds.

1. Clone/download the Android version 4.4 from the Android-x86 project repository (http://www.android-x86.org).

2. Insert VirtIO modules into the kernel by amending following lines at the end of the ”kernel/arch/x86/configs/android-x86_defconfig” and ”kernel/arch/x86/configs/android-x86_64_defconfig” files.

```
// Enable virtualisation drivers
CONFIG_VIRT_DRIVERS=Y
// Enable virtio generic drivers
CONFIG_VIRTIO=y
// support for PCI passthrough
CONFIG_VIRTIO_PCI=y
// support for memory mappted virtio
CONFIG_VIRTIO_MMIO=m
// support for dynamic memory allocation.
CONFIG_VIRTIO_BALLOON=m
// support for virtual block devices for Qemu–KVM
CONFIG_VIRTIO_BLK=y
// support for virtio virtual networks
CONFIG_VIRTIO_NET=m
```

```
// support for ring buffers
CONFIG_VIRTIO_RING=m
// support for console
CONFIG_VIRTIO_CONSOLE=m
// support for virtual random number generator hardware
CONFIG_HW_RANDOM_VIRTIO=m
```

3. Android-x86 does not detect virtual block devices that are presented to the virtual machine by Virtio. To fix the issue, the author changed the following code snippet in the Android source code in the file "bootable/newinstaller/initrd/init".

   Change line from;

   for device in $ROOT:-/dev/sr /dev/[hs]d[a-z]* /dev/mmcblk*; do*

   to;

   for device in $ROOT:-/dev/sr /dev/[hsv]d[a-z]* /dev/mmcblk*; do*

4. Include SSH software libraries to the OS. Add following packages to PRODUCT_PACKAGES in device/generic/x86/packages.mk.

```
ssh-keygen
sshd_config
start-ssh
```

5. Force gets the DHCP lease from the DHCP server and starts SSH daemon at the startup by altering following lines at the end of the file "device/generic/x86/init.sh".

```
//Force dhcp on eth0 interface.
netcfg eth0 dhcp
//Start SSH daemon at startup.
start-ssh
```

6. Recompile Android with all above changes.

7. Install androidx86 OS onto a disk storage (DVI, vmdk, qcow2).

8. To fetch the nova keypair, create fetchsshkeys script with following lines in /data/local/.

```
#-----------------------------------------------------------------
!/system/bin/sh # use bourne shell

# Fetch public key using HTTP
cd /data
wget http://169.254.169.254/latest/meta-data/public-keys/
0/openssh-key
# Add new recieved key to local authorized key list
cat /data/openssh-key > /data/ssh/authorized_keys

# Set permision of file so that owner may read and write
chmod 0600 /data/ssh/authorized_keys
# Reset security context
restorecon /data/ssh/authorized_keys

# Remove temporary key file
rm /data/openssh-key
#-----------------------------------------------------------------
```

Then make the script executable by,

chmod 755 /data/local/fetchsshkeys

9. Execute the script at the startup of the VM. Add the following lines to the /etc/init.sh file.

```
#Fetch ssh keys from openstack metadata service.
/data/local/fetchsshkeys
```

10. Enable Android GUI

    Amend keyword ”nomodeset” into the main GRUB record.

11. The image is ready to be used to create OpenStack VMs.

Appendix B

List of Publications

- C. S. Magurawalage, Kun Yang, Ritosa Patrik, Michael Georgiades, and Kezhi Wang. A Resource Management Protocol for Mobile Cloud Using Auto-Scaling. available in *https://arxiv.org/abs/1701.00384*.
- T. Li, C. S. Magurawalage, K. Wang, K. Xu, K. Yang, H. Wang. On efficient offloading control in cloud radio access network with mobile edge computing, submitted to IEEE ICDCS 2017
- K. Wang; K. Yang; C. S. Magurawalage, "Joint Energy Minimization and Resource Allocation in C-RAN with Mobile Cloud," in IEEE Transactions on Cloud Computing , vol.PP, no.99, pp.1-1 doi: 10.1109/TCC.2016.2522439
- K. Wang, K. Yang, X. Wang and C. S. Magurawalage, "Cost-effective resource allocation in C-RAN with mobile cloud," 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, 2016, pp. 1-6.
- C. S. Magurawalage, K. Yang, L. Hu, J. Zhang, Energy-efficient and network-aware offloading algorithm for mobile cloud computing, Computer Networks, Volume 74, Part B, 9 December 2014, Pages 22-33, ISSN 1389-1286