Efficient Data Access in Cloudlet-based Mobile Cloud Computing

by

Zhijun Hou

Thesis submitted in partial fulfillment of the requirements
For Master of Computer Science degree

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

Abstract

The growth in mobile devices and applications has leveraged the emergence of mobile cloud computing, which allows the access to services at any place and extends mobile computing. Usually, the current mobile network consists of a restricting factor in supporting such access because, from a global perspective, cloud servers are distant from most mobile users, which introduces significant latency and results in considerably delays on applications in mobile devices. On the other hand, Cloudlet are usually on the edge of Mobile Networks and can serve content to mobile users with high availability and high performance. This thesis reviews both the traditional mobile cloud computing and the Cloudlet architecture. A taxonomy on the Cloudlet architecture is introduced and three related technologies are discussed. Based on the user needs in this environment, personal model which is used to predict individual behaviour and group model which considers caching popular data for several users are proposed. Making use of these two models and the Cloudlet architecture, two data access schemes are designed based on model distribution and data pre-distribution. We have conducted experiments and analysis for both the models and data access schemes. For the models, model efficiency and comparisons among different technologies are analysed. Simulation results for the data access schemes show that the proposed schemes outperform the existing method from both battery consumption and performance aspects.

Acknowledgements

First and foremost, I would like to express my very great appreciation to my parents for their valuable encouragement and understanding during my study. Thank them for supporting me to come to this country, this school to pursue what I want to do which is the best gift for me.

Next, I would like to express my deep gratitude to Professor Azzedine Boukerche, my research supervisor, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. The equipments and financial support he offered make me more concentrate on my studies.

Then, I am particularly grateful for the assistance given by Dr. Robson for his help in keeping my progress on schedule. In these two years of graduate life, he listened carefully to our ideas and made suggestions. His serious attitude and enthusiasm for academics have influenced me and made me to move forward with more determination.

Finally, I would also like to extend my thanks to the technicians of the PARADISE Lab for their help in offering me the resources in running the program. Advice given by Shichao Guan has been a great help in gradually building my own ideas. Hope this thesis will not be the end of academic thinking.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

AODV	Adhoc On Demand Distance Vector
AP	Access Point
BS	Base Station
CC	Cloud Computing
CDN	Content Delivery Network
EPC	Evolved Packet Core
HSDPA	High-Speed Downlink Packet Access
IaaS	Infrastructure as a Service
LAN	Local Area Network
LFU	Least Frequently Used
LRU	Least Recently Used
MANET	Mobile Ad hoc NETworks
MCC	Mobile Cloud Computing
MRU	Most Recently Used
PaaS	Platform as a Service
QoS	Quality of Service
RSUs	RoadSide Units
SaaS	Software as a Service
SDN	Software Defined Network
SLP	Service Location Protocol
VANET	Vehicular Ad-Hoc NETwork
VC	Vehicular Cloudlet
VM	Virtual Machine
VOD	Video-On-Demand
WAN	Wide Area Network
WMAN	Wireless Metropolitan Area Network

Chapter 1

Introduction

Mobile Cloud Computing (MCC) has introduced a new computing model generated from the combination of mobile computing, mobile networks, and cloud computing [54] [48] [77]. MCC provides services for mobile devices at any time and anywhere from several perspectives, allowing to access data-rich applications through the network. This perception originates from the fact that MCC has directly evolved from Cloud Computing and Mobile Computing; consequently, this new paradigm has permitted to move the burden of data processing and storage from restricted mobile devices to cloud servers [58]. This new method fundamentally changed the operating mode and lowered requirements for both performance and storage of mobile devices. As a result, the processing ability of devices can be notably leveraged, arising from the resources, services and powerful processing capability of the cloud. At the same time, the flexible environment of MCC presents high heterogeneity, which is vividly present by the use of a series of different mobile devices, such as *tablets* and *smart phones*, with a diversity of capabilities and applications with a variety of requirements.

1.1 Motivation

Even though MCC enables enhancement on mobile devices, it massively relies on the mobile network for retrieving data from cloud servers. Besides being restricted by coverage and access, this substantial dependency imposes a considerable burden on the network through constant communication. In the MCC, the cloud is formed by *countless* data centres which include servers with comprehensive computing and storage capabilities [54]. When mobile terminals connect to the server, the amount of data transmission between servers and mobile terminals, as well as servers and servers, is significantly large, which leads

to network latency due to the limited bandwidth of wireless networks. Network latency is a fundamental problem in MCC because it causes delays on applications and disrupts services.

In order to mitigate network delays in MCC architecture, Cloudlet [97] is introduced as a middleware between mobile devices and cloud servers that combines the Cloudlet into a traditional MCC architecture. The Cloudlet can be viewed as a *data center in a box* whose goal is to *bring the Cloud closer* so that mobile users can request service from the Cloudlet rather than a remote cloud server. This is very helpful for reducing the latency if the required data can be accessed directly from the Cloudlet or the computation on mobile devices can be offloaded to the server. Thus, some applications that need to interact with the user in real time [88] can get a performance boost. Another reason of using Cloudlets consists of that mobile terminals use WAN to connect with remote cloud while they use LAN to connect with a Cloudlet; this leverages the communication constraints because the bandwidth of wireless LAN is typically two orders of magnitude higher than the wireless Internet bandwidth available to mobile devices [97]. As a consequence, using Cloudlet can make an effective reduction of the latency between users and servers.

Using different frameworks can significantly improve efficiency from several aspects, but the Cloudlet-based MCC architecture also brings an issue. That is, what type of computation should be offloaded from the mobile device to the Cloudlet server or what kind of data items should be cached on the server to reduce the number of requests sent to the remote cloud servers. Because this thesis mainly deals with the problem of data storage and management on the Cloudlet server, we only discuss the latter problem related to cached data types in terms of motivation. The first data type we consider is individual user-related data. Given a scenario which often occurs in our lives: a user uses the news application to see the news of the day on everyday morning, but it takes a while to load the news of the day when he opens the application everyday. This situation motivated us to think that if we can use this user's previous behaviour to predict the data items he would like to access to in the future. Then when this predicted data exists in the network, it can be transmitted to the user's mobile phone cache or a nearer Cloudlet server. In this way, this user no longer has to spend time waiting for information to load.

The second data type is based on multiple users which is mainly to find the data items that a group of users are interested in. Take a scenario as an example: every work in a museum has a corresponding commentary which only written in one language. Then a group of people comes to the museum and they could not understand the language used to describe the works. At this time, they would search on the website and find out the corresponding translations to understand the works. In a normal situation with traditional MCC environment, all users will separately request the distant cloud server to

obtain the corresponding information. However, our idea is that under the Cloudlet-based MCC framework, if the information can be cached in the Cloudlet after the first person requested the information, then other users can obtain the needed information in a nearer server. Apart from this, accessing data items from Cloudlet can also reduce additional data transmission over WAN which alleviates the burden on the WAN. Therefore, motivated by these two data types, our research dedicates to data distribution and management in the Cloudlet-based MCC architecture in order to achieve a better user experience and access efficiency.

1.2 Objective and Contributions

The objective of our research is to study the architecture of Cloudlet and propose efficient data access schemes for this framework. Based on the discussed motivations, firstly, two methods should be proposed to generate the corresponding data types. Then these methods are needed to be applied to the appropriate component in the architecture and distributing the generated datasets in the entire system. Finally, data management is also needed to be considered among all the components. Thus, our major contributions are as follows:

- A Cloudlet taxonomy based on architecture is introduced.
- A personal model is proposed to generate individual-related datasets. In the personal model, data pre-fetching technology is applied to speculate on what kind of data the user may obtain in the future according to the behaviour of individual users and different usage patterns. To achieve this, we use association rules to find datasets with a high possibility to be accessed by users in the future and detect the relationships among different data items.
- Group model is introduced to generate popular data items among a group of users. Data caching technology is applied to the group model to cache the data items that accessed by users before and have the possibility to be requested again. In this model, we also adopted an efficient data replacement policy to replace unimportant data items to decrease the cache utilization.
- Model and dataset distribution in the Cloudlet-based MCC architecture is suggested. 2 models and the generated datasets are distributed to mobile devices, cloud server and Cloudlet server respectively which considers both the capability of servers and the importance of data items.

- Experiments and analysis on the proposed models and data access schemes are conducted. User access traces used to conduct the experiments are mapped from real datasets. Both the proposed models and access schemes are implemented in the simulator and analysed from both energy consumption and efficiency.

1.3 Outline

This thesis is organized as follows. Chapter 2 gives a literature review on the Cloudlet-based MCC architecture and compares Cloudlet with traditional cloud framework. A classification on the Cloudlet is also introduced. In Chapter 3, personal model and group model are proposed respectively which include basic technologies and experimental results. 2 data access schemes are discussed in Chapter 4 which shows the model and datasets distribution in the 3-tier Cloudlet-based MCC architecture. The performances of the proposed schemes are also evaluated in this chapter by implementing the whole system in a simulator. Chapter 5 summarizes the conclusions and future work.

Chapter 2

Literature Review

This chapter gives a literature review on Cloudlet architecture and data offloading technologies in Cloudlet-based Mobile Cloud Computing framework. Section 2.1 gives the background and explains some fundamental concepts of Mobile Cloud Computing area. By introducing different features of traditional Mobile Cloud Computing and Cloudlet architecture, a comparison between these two models is shown. In section 2.2, a comprehensive review and classification on the Cloudlet architecture is presented. Section 2.3 reviews data offloading on Cloudlet including data caching and data pre-fetching technologies.

2.1 Background

This section briefly gives a summary of Mobile Cloud Computing architecture including definition, application, advantages and then draws on some of these background concepts and motivates the need for Cloudlet. From the perspective of Cloudlet, we summarize some general characteristics and benefits which can make up for the weakness of the traditional framework applied in the mobile environment.

2.1.1 Mobile Cloud Computing

According to the Mobile Cloud Computing(MCC) forum, MCC is defined as:

'Mobile cloud computing at its simplest, refers to an infrastructure where both the data storage and data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just smart-phone users but a much broader range of mobile subscribers'.

MCC takes advantage of cloud computing to make up for the disadvantages of mobile computing. Cloud computing allows users to access information and services at anytime, anywhere and any devices without backup or copy documents themselves. From the perspective of deployment model, cloud computing can be classified into private cloud, public cloud and hybrid cloud. And from service model side, cloud provides Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [71, 77]. Many companies have already offered cloud services such as Amazon EC2, a public cloud provider, allows users to purchase and allocate their own Virtual Machines(VM) which follows IaaS on the cloud; DropBox provides users with data storage services which is SaaS. Therefore, after the combination of cloud computing and mobile networks, mobile devices can also use these cloud services and resources.

User experience is satisfied by the improvement of data processing efficiency, extended battery life and larger storage space on their mobile devices. With cloud's offloading technologies, mobile devices transmit energy-consumed applications to the cloud so that power consumption is reduced[73]. Operating efficiency of mobile devices is enhanced with the help of much more powerful cloud processors which executes applications with the computation offloading. Paper [123] also proves storage capacity of mobile devices is expanded by using SaaS services provided by cloud so that large files and documents can be stored online. With data stored on the cloud server, reliability is also improved so that malfunctions or virus no longer has the same big effect on data loss as before [98].

However, given MCC architecture applies traditional mobile networks which can be depicted as mobile devices use cellular data(2G/3G/4G) to access network and cloud services. In this environment, issues like constrained bandwidth, latency, unreliable, time synchronization [30], and fault identification [50] are resulted from the cellular network. So although cloud can provide such rich resources, mobile devices can't maximize utilization. Some applications demanding on latency requirements like applications needing real-time data processing and transmission can't benefit from MCC. Furthermore, in real life, the disadvantages of accessing cloud services over wireless radio networks have actually overshadowed the benefits brought by cloud services themselves. Thus, it is an evident opinion and best way in MCC that mobile devices should optimally utilize cloud services with the least amount of network traffic [6].

2.1.2 Cloudlets

Although centralized cloud computing exhibits abundant resources for computation-intensive tasks, the unpredictable and unstable communication latency between the mobile users and

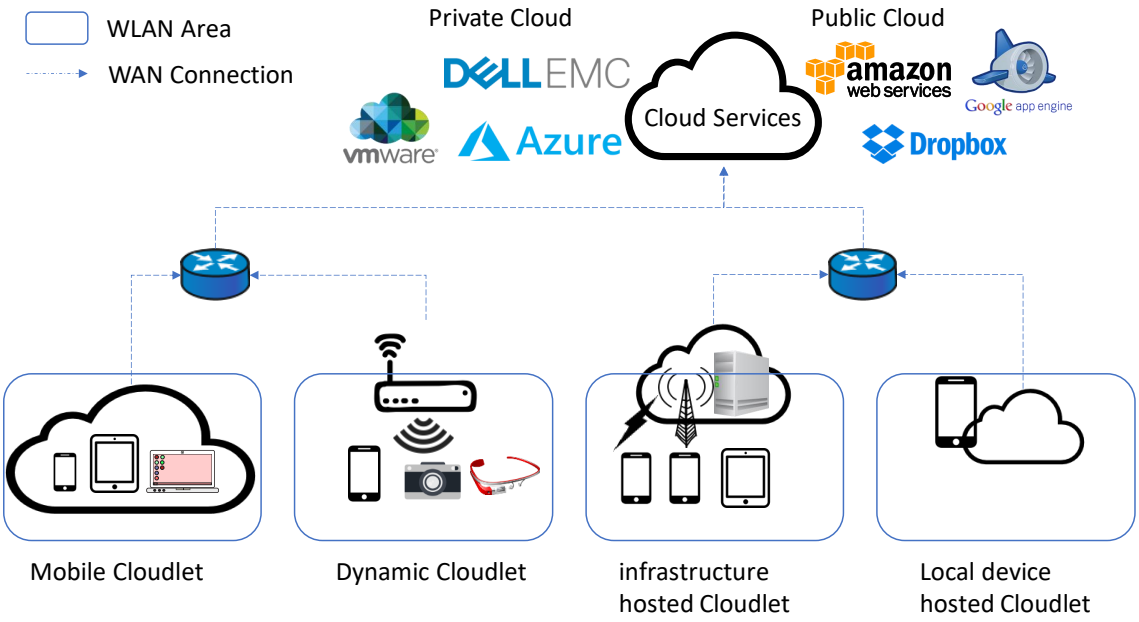

Figure 2.1: The general concept of a Cloudlet as described in [98]

the cloud makes it challenging to handle latency-sensitive mobile computing tasks. To address this issue, cloudlet [97] recently was proposed by pushing the cloud computing to the network edge closer to the users.

Cloudlet is first introduced in [97] as a trusted, resource-rich computer or cluster of machines that is well-connected to the Internet and available for use by nearby mobile devices. Later in 2014, paper [1] depicts Cloudlet as a proximate fixed cloud consists of one or several resource-rich, multi-core, Gigabit Ethernet connected computers aiming to augment neighbouring mobile devices while minimizing security risks, offloading distance (one-hop migration from mobile to Cloudlet), and communication latency.

Figure 2.1 gives the general concept and provides some promising architectures of Cloudlet. Because this is a relatively new research area, there is no unified definition and framework about Cloudlet. After researching some related papers, we summarized some basic features of Cloudlet as follows.

- Different from the remote cloud server, Cloudlet has a very close distance from the mobile terminals. Usually a Cloudlet is located at one wireless hop away from the mobile device such as a cellular base station or a Wi-Fi Access Point(AP) [86].
- A Cloudlet coverage refers to the area that formed by all the mobile devices connecting with a same Cloudlet server.

- No matter in what kind of classification, Cloudlet-based MCC architecture is consistent with the same 3-tier hierarchy with mobile devices, Cloudlet and cloud server.
- Cloudlet uses high speed Wireless Local Area Network(WLAN) links to connect with mobile devices and has a high speed access to the distant cloud server [109].

In contrast to the cloud-based MCC paradigm, Cloudlet uses a faster network to connect with users. Thus, Cloudlet is closer to mobile devices and it also has a high-speed network connection with real cloud servers. Additional to these, it has own storage and computation capabilities which can be provided to users. The following outlines some advantages of Cloudlet.

WAN latency. In traditional MCC, mobile devices rely on distant cloud to augment capability which brings Wide Area Network(WAN) latency. WAN latency, unlikely to improve in foreseeable future, has more impact on deeply immersive tasks like augmented reality than loosely coupled tasks [97]. With Cloudlets, those applications with strict latency requirements can be benefited by connecting with a nearby, resource-rich server and offloading computation. Since the one-hop away distant and fast WLAN link, the need for real-time interactive response will be satisfied. WAN latency which really hurts the efficiency in traditional MCC can have its improvement in the new paradigm and this is the strongest advantage a Cloudlet has over a Cloud [109].

WAN utilization. Every mobile device needs to consume WAN when accessing internet in traditional MCC which increase the load on WAN. In the Cloudlet-based MCC architecture, mobile devices use WLAN link to connect with Cloudlet which then connects to the cloud server. In this way, some mobile devices can be served on the Cloudlet without consuming WAN, the load on the WAN is alleviated, consequently enabling more Cloudlet to access the WAN to retrieve services and serve more mobile devices. Another perspective is that when mobile devices in the same Cloudlet coverage requests same data, this request will be just sent once by Cloudlet while in traditional MCC, the request will be sent by different mobile devices. In this situation, the latter one will waste WAN by sending the same request and transmitting same data for many times.

Cloud load. Cloudlet helps to alleviate the load on cloud by serving mobile devices itself. Once a Cloudlet can satisfy requirements sending by mobile devices such as processing complicate applications, the computation won't be offloaded to the cloud server. Therefore, the cloud doesn't need to deal with as many requests as before which means the load on cloud server will be reduced.

Higher bandwidth. The connection between mobile devices and offloaded server is WLAN in Cloudlet-based MCC while WAN in traditional MCC. The bandwidth of WLAN

is typically two orders of magnitude higher than the WAN bandwidth available to a mobile device, for example, the nominal bandwidths of currently WLAN (802.11n) and wireless Internet High-Speed Downlink Packet Access(HSDPA) technologies are 400 Mbps and 2 Mbps, respectively [97].

Data access efficiency. By caching data items on Cloudlet server, mobile devices can retrieve information from Cloudlet directly. In this way, data access latency is decreased. Some applications such as video streaming in wireless networks [24, 93, 19] which suffers high latency can be benefited. In the traditional Cloud model, a mobile device sends a request to the Cloud to request information; the serving time corresponds to sending a package via WAN plus processing time plus receiving a package via WAN. Under the same circumstance, the 3-tier Cloudlet model can reduce service time with lower transmission time via WLAN.

Cost effectiveness [98]. Different from cloud server which includes both hard state and soft state, a Cloudlet only includes soft state such as cache copies of data or code [97]. These data or code are available on other platforms such as mobile devices or the cloud. The cloudlet caches this information just for better serving mobile users. So as a result, it is not fatal if one cloudlet is lost or destructed. The cache data can be transmitted from the devices or cloud that the cloudlet copes from.

Offline availability. In MCC environment, mobile users may not be able to connect to the cloud to obtain a service due to traffic congestion, network failures, and the out-of-signal [48]. On the other hand, Cloudlet are deployed on a Local Area Network(LAN), some services such as computation offloading provided by Cloudlet does not depend on the availability of an active Internet connection. As a result, mobile devices can use the Cloudlet service at any time without accessing the Internet.

Management. From paper [97], a Cloudlet is self-managed and need little to no professional attention while a cloud server is a 24/7 operator that needs professional administered. This results in that a Cloudlet can be deployed in a local business location such a book store or a coffee shop while the granularity of cloud servers make only big companies can operate them. Thus, to manage and operate a Cloudlet is easier.

2.2 Cloudlet Taxonomy Based on Architecture

The concept of Cloudlet is of a resource-rich device proximate to the mobile devices and connected via low latency local area networks, capable of lending resources to the mobile device to perform resource-intensive tasks [98]. Although the concept is proposed, the

Cloudlet architecture and deployment hasn't been unified. Unlike cloud in the MCC which is now a mature technology, different cloud architectures have been built according to PaaS, SaaS and IaaS. The Cloudlet is a technology just put forward, so its architecture is not well unified and deployed. Derived from the concept, researchers built different Cloudlet architectures in order to satisfy different services. Depending on the deployment approach of a Cloudlet, we categorize the Cloudlet architecture into static Cloudlet which relies on existing infrastructure to place the Cloudlet and dynamic Cloudlet in which a group of devices can form a Cloudlet.

2.2.1 Static Cloudlet

Static Cloudlet architecture adds a server to the existing network infrastructure at a or some points to increase efficiency which relies on the network operator to deploy the Cloudlet. One typical feature of static Cloudlet is that any mobile device entering into a LAN area will get services from the same Cloudlet. Cellular base station or AP is the most common placement for static Cloudlet which is the which is just one-hop away from mobile devices. Many research papers have adopted the static Cloudlet-based approach in different scenarios which relies on the existing network structure to deploy the Cloud server.

Cloudlet in Access Point

Paper [66] merged the MCC concepts with the proposed Cloudlet framework by deploying the Cloudlet in Wi-Fi AP. In a traditional MCC model, mobile devices use 3G/LTE to access Internet which results in high network latency and huge transmission power consumption. The model proposed in this work integrates the Cloudlet and AP into one entity so that mobile devices can access AP while making use of services provided by the Cloudlet. The MCC Cloudlet-based model includes set of distributed and well-connected Cloudlet within the place where there is a high possibility for mobile users to use cloud services. Those Cloudlet also have connections with the Enterprise cloud which benefited from the connections between AP and cloud servers.

Paper [66] only pointed out that a Cloudlet server can be integrated with AP, but did not specify the existence form and placement problem of the server. Paper [69] and paper [119] firstly restrict Cloudlet to a cluster of computers that is colocated with an AP and then the research directions of both papers are focusing on the Cloudlet placements in a Wireless Metropolitan Area Network(WMAN). For WMANs, especially large WMANs including hundreds and thousands of APS, the location of Cloudlet is essential for mobile users when they request cloud services. Poor Cloudlet placement will result in access delays

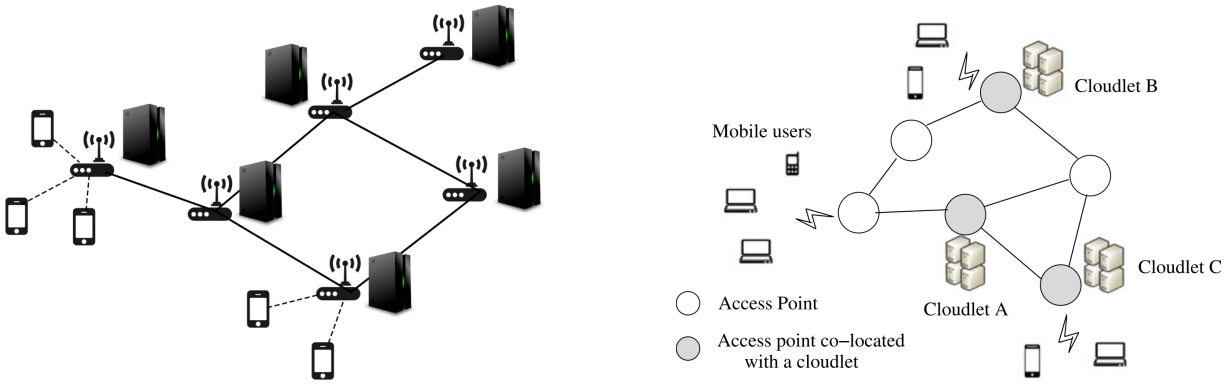

Figure 2.2: Cloudlet in access point decribed in [69] and [119]

and reduced resource utilization. Furthermore, the uneven resource utilization among Cloudlets will be caused by the uneven density situation. When the density of population in a certain place is high, mobile users' data access will focus on several Cloudlets which will cause server overload in this place while in another place with low density, resources of Cloudlets in this place may be wasted. The WMAN which consists of many wireless APs is often owned and operated by local governments as public infrastructures [40]. Thus, paper [69] assumes a pre-placement of Cloudlet servers that all the Cloudlet servers are set up in advance at a fixed location in the WMAN and all of them are connected with each other via the network connection as the left figure in 2.2 shows. User application on mobile devices can be partitioned into several discrete tasks which then can be offloaded to and processed by every Cloudlet in the WMAN. When a Cloudlet receives a task from a user, it can decide whether processing the task by itself or redirect the task to another Cloudlet.

However, paper [119], different from paper [69], doesn't have that Cloudlet pre-placement assumption. Authors focus on the Cloudlet placement problem that how to deploy k Cloudlets at m APS in the WMAN so that the average Cloudlet access delay in the whole network can be minimized. The right figure in 2.2 shows an example Cloudlet placement in a WMAN with three servers and six APs. Given the difference between deployment in traditional networks, capacity constraint and identical amounts of resource demands for every user. This paper considers that Cloudlets are configured with different computing capacity which can better satisfy real applications [68, 115]. In addition to this, different user requests have different amounts of resource demands are also taken into consideration. The paper proposes an Integer Linear Programming (ILP) solution and two approximation algorithms to deal with the placement problem. After the placement of all Cloudlet in WMAN, an on-line algorithm is designed to assign user requests to different Cloudlet dynamically.

In a traditional setting when a Cloudlet sever is built in an AP, the server which mobile devices choose to connect with is always the nearest one. In this way the transmission time will be reduces when offloading applications to the Cloudlet. However, this approach causes the uneven resource utilization on Cloudlet [69]. Getting services from the nearest Cloudlet leads to the problem that part of Cloudlets are overloaded while others may be very lightly loaded. The quality of services provided by these overloaded Cloudlet is poor. When a Cloudlet is overloaded which means that many people are using the service at the same time. Therefore, user experience for these people will be very bad. In paper [33], researchers find if the Cloudlet servers are connected with each other, the tasks offloaded by mobile devices can be processed in a cooperative method among Cloudlet.

For the connections among Cloudlets, the architetcure proposed in paper [14], every Cloudlet are connected with each other using the Adhoc On Demand Distance Vector (AODV) routing protocol. Thus, even if the mobile device is moving from one network to another, the Cloudlet can still provide services to the mobile device as required by it. Every Cloudlet provides services for surrounding mobile devices and are connected to the enterprise cloud. When a mobile device leaves a Cloudlet coverage and enters another coverage, the routing information will be swapped by making the use of AODV.

Cloudlet in Software Defined Network(SDN)

In the infrastructure-based Cloudlet architecture, in addition to the Cloudlet which built in an AP or connected to an AP, the architecture described in paper [33] connects Cloudlet with each other through Software Defined Network(SDN) switches to improve the computational resource utilization. Due to the connections, overloaded Cloudlet can transmit the offloaded task to another idle Cloudlet. In this model, there is a central Cloudlet manager which is used to coordinate the tasks among the Cloudlet servers and the manager is located at the SDN network core. According to the traffic load evaluated by every Cloudlet periodically, the central manager calculates the overloaded Cloudlet and the idle ones. Task assignment algorithm is conducted by central Cloudlet manager in order to minimize total network wide latency.

Paper [74] also considers SDN and proposes a SDN based Cloudlet architecture in order to mitigate the integration issues generated from deploying cloudlets over current Evolved Packet Core(EPC) network. The SDN is an OpenFlow based network which includes OpenFlow switches/routers and an OpenFlow controller. The Cloudlet is placed in local SDN-based Mobile Telephone Switching Office(MTSO). In MTSO, Cloudlet is connected with the existing EPC components through the OpenFlow switches/routers and the controller is used to supervise all the switches/routers within the local network.

Thus, the existing EPC elements and routing paths won't be modified. Applications in this architecture is split into an application front end which is in remote cloud and an application server engine(Cloudlet) which is placed in local MTSO. Requests from mobile devices are firstly uploaded to the front end and then transmitted to the local Cloudlet. The upload path is same as the path in original EPC network while the download path takes Cloudlet into consideration. This results in a download path with front end to Packet Data Network Gateway(P-GW) to Cloudlets to Serving Gateway (S-GW) to enhanced NodeB(eNB) to user while the path in original EPC network is front end to P-GW to S-GW to eNB to user(P-GW and S-GW is in the EPC system). In this step, an enhanced OpenFlow 1.3 protocol is proposed in this paper to redirect packets from P-GW to Cloudlet and from Cloudlet to S-GW. In general, this architecture enables a computation offloading in local Cloudlet without without any functionality and routing modification in the existing EPC network.

Cloudlet in Content Delivery Network(CDN)

In addition to having the Cloudlet coexist with an AP and SDN, Cloudlet Aided Cooperative Terminals Service Environment [91](CACTSE) connects a Service Manager(SM) with the Content Delivery Network(CDN) end. In the CACTSE system, SM is a Cloudlet like module which is responsible for local network coordination. By connecting with the CDN end, contents are pushed more close to users while the existing CDN will not be disturbed. Users in CACTSE system uses cellular network to access Internet and all mobile devices within a SM's service region are registered on the SM through WLAN. After registered, contents are uploaded under users' admission to SM. For the reason that users within a specific domain often share the same interests, contents uploaded by $userA$ may be contributed to data request by $userB$. This paper also considers different status of SM, such as when a SM fails to access Internet and when it confirms a missing content in the local domain, it will coordinate the terminals which have Internet access to split the original request and then collect the downloaded contents; When the whole system is down, means that no devices can access Internet, the SM can be viewed as a content sharing system in local area.

Cloudlet in Vehicular Ad-Hoc NETwork(VANET)

Cloudlet can also be applied in Vehicular Ad-Hoc NETwork(VANET) [3] by attaching local cloud servers(Cloudlets) to RoadSide Units(RSUs) [121, 67, 55]. In traditional VANET, communication types fall into vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I)

which are used for vehicles to communicate with each other [51, 43]. When combining VANET with Cloud Computing(CC), showed in figure 2.3, V2I communication uses Rroad-Side-Units (RSU) to deliver packets from vehicles to cloud server. Paper [121] first introduced a comprehensive cloud-based VANET architecture which includes Vehicular cloud, Roadside cloud(RSC) and Central cloud. From figure 2.3, the main difference between this new architecture and the traditional one is the RSC part which deploys Cloudlet servers in RSU. Cloudlet in RSC provide cloud services such as storage and computing to vehicles in its coverage. A vehicle can select a nearby RSC and use Cloudlet to customize a transient cloud(a VM like module) which will be deleted from the RSC after a vehicle leaves the RSC coverage.

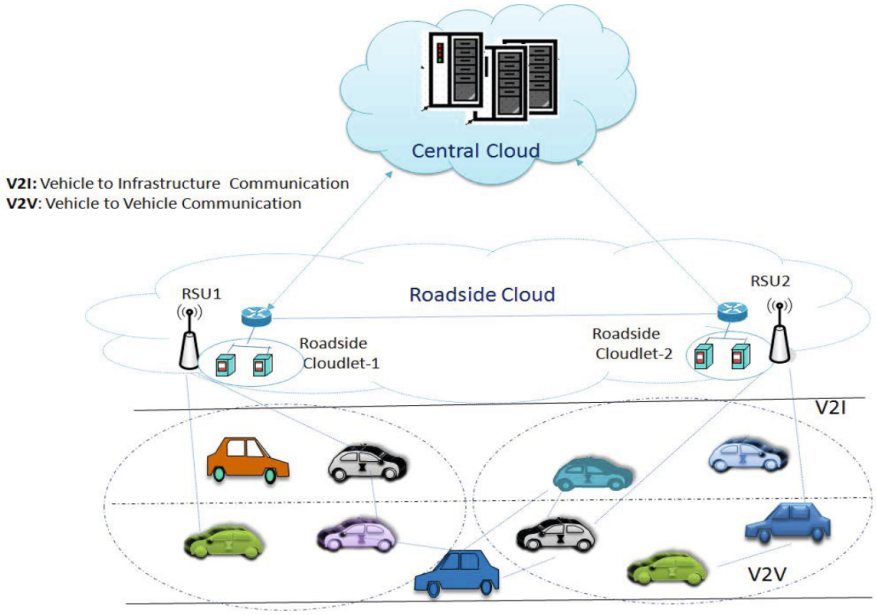

Figure 2.3: Cloudlet in vehicular ad-hoc network described in [67]

Compared with getting services from remote Cloud, using RSC reduces access delay and transmission time resulted from the geographical proximity of RSC and vehicles. The Cloudlet in RSC is the VM-based Cloudlet(a classification in [98]) which consists of the VM-base and the VM-overlay. When a vehicle requests a cloud service from RSC, a specific VM-overlay for this service is transmitted to the Cloudlet. The Cloudlet then combines the VM-base which is a basic template already in Cloudlet with the VM-overlay to generate a dedicated VM used to provide specific service for a vehicle. To guarantee the continuous cloud service for moving vehicles, the VM migration technology [66] is applied to transfer the VM to the corresponding RSC which the vehicle is reconnecting with.

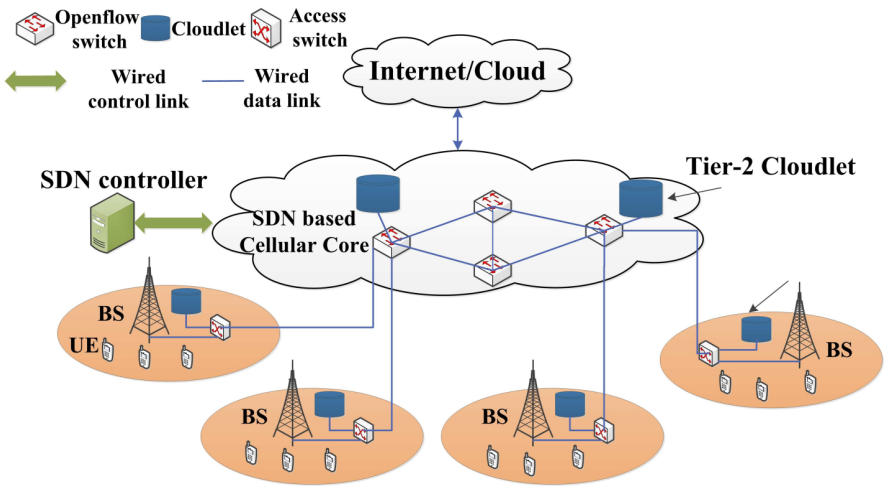

Figure 2.4: Two-tier Cloudlet in SDN and base station described in [52]

Cloudlet in Multiple Infrastructures

In the Green Cloudlet Network(GCN) architecture [104], Cloudlet is deployed in close proximity to Evolved Node B(eNB) with the connection of a wired data link so that the delay between the eNB and the cloudlet is negligible. From the perspective of 'Green' in GCN, Cloudlet is powered by both on-grid energy and green energy to reduce operational costs. Avatar, as introduced in this article, is a software clone(a VM like module) of the User Equipment(UE). Different from the VM we have mentioned, avatar is running on the same operation system as its UE which means every UE has its own specific avatar while the previous VM is in connection with applications. Avatar provides powerful computational units and communication caches as well as large storage disks for corresponding UE. GNC architecture is reliable in that Cloudlet is connected with a public data center and a storage area network. Therefore, in case of limited resources in Cloudlet in which no more avatars can be hold, avatars can be migrated to the public data center. Meanwhile, the virtual disk copy can be transmitted to the storage in the network to prevent data loss.

In GCN, eNB supports the communications among UE, Cloudlet and Internet which indicates if a UE doesn't need the services provided by Cloudlet, it can send request to Internet directly without passing a Cloudlet. In other words, the added Cloudlet doesn't change the original network which simplifies the deployment and management. GCN also considers the density of UEs: in rural areas where UEs are sparsely distributed, several eNBs connect with a same Cloudlet to provide services; on the other hand, smaller-size Cloudlet are connected in picocells and femtocells which GNC applies to increase the network capacity with a higher UE density.

The above articles are all following Cloudlet in one infrustructure that means the

Cloudlet is deployed in the same kind infrustructure like AP and RSU. However, in some papers, Cloudlets are deployed in different infrastructures at the same time. The 2-tier Cloudlet network architecture proposed in paper [52] is showed in figure 2.4. Same as the architecture in [74] which adopts the SDN, this hierarchical architecture has a SDN based cellular core that connects Base Stations(BS) with the Internet cloud. Tier-1 Cloudlet are connected with BS and tier-2 Cloudlet servers are placed at openflow switches in the SDN cellular core. The advantage of tier-1 Cloudlet is lower network delay due to the close distance to UEs while this kind Cloudlet have limited resource so that it is possible to be overloaded if UEs become more. On the other hand, the tier-2 Cloudlet have much more computation resources than the tier-1 Cloudlet which gains lower computing delay. Tier-1 Cloudlet and BSs also have access to the SDN cellular core. For UEs, BS is the direct request receiver and UEs can connect with different tier Cloudlet to get services by connecting with BS. For each BS, there is a tier-1 Cloudlet while the tier-2 Cloudlet can be shared among some switches. In this architecture, UEs can offload their workload to a nearby tier-1 Cloudlet and if this Cloudlet is overloaded, the workload can also be offloaded to its corresponding tier-2 cloudlet.

2.2.2 Dynamic Cloudlet

Dynamic Cloudlet is referred as a device or a group of devices which can be seen in wireless sensor networks [20]. The most distinctive feature of the dynamic Cloudlet is that it doesn't have dependency on the infrastructure. In other words, dynamic Cloudlet doesn't need to be deployed by a service provider like the static Cloudlet. Dynamic Cloudlet can be a cooperation among different devices within an area such as mobile phones in a house can cluster together to form a Cloudlet. Mobile devices in a home trust each other, so they can connect with each other easily. For other scenarios, for example in a classroom, users can firstly negotiate and then those agreed the rules can form a Cloudlet. There are 2 offloading forms in dynamic Cloudlet: the first one is doing offloading within the Cloudlet coverage meaning that resource-poor devices can make use of other devices to process an application; the second one is that a dynamic Cloudlet is formed to provide services for other mobile devices. The difference between these 2 forms is whether the requesting mobile device is a node in the dynamic Cloudlet. For the former type, mobile device offloads applications to other nodes within the Cloudlet via the high-bandwidth LAN [32]. But for the latter one, the formed Cloudlet needs to register on-line and decide the resources that they want to provide so that other mobile devices can use it.

Both centralized and decentralized management are applied to the dynamic Cloudlet [98]. From the perspective of a centralized Cloudlet, a master node is selected to perform all man-

agement operations within the Cloudlet coverage which is referred as leader selection [15]. Some standards such as reputation computation [25] can be used to select a master. This node coordinates and allocates resources over the mobile devices in the Cloudlet since it has a global view of the resources. Similarly, management can also be distributed to each node within the Cloudlet. In this situation, each node tracks its own resources and periodically informs all other nodes in the Cloudlet about its available resources as well as the current state of the job being performed. Since there is no master node to perform the task of select a suitable node and distribute a task, the assignment in this case is performed by the mobile device itself based on the information such as the remaining resources received from other nodes.

Static Cloudlet we have introduced have heavy dependency on network operator which means the deploy location of a Cloudlet is decided by operators like AP or eNB. Different from that, dynamic Cloudlet alleviate that dependency by forming a Cloudlet by mobile devices themselves. In the static Cloudlet, the application are often processed completely by the VM on a Cloudlet. But applications in dynamic Cloudlet are consisted of different components and each of them can be offloaded to a Cloudlet node referring to a mobile device makes up the Cloudlet. In this way, components can be distributed among various Cloudlet nodes according to application requirements, thereby achieving a more flexible resource allocation. That is, the latency sensitive component can be handled on the own device, while the non-critical part can be offloaded to other Cloudlet nodes.

Paper [109, 108] proposed a dynamic Cloudlet concept that any device in the same LAN network with available resources can form as a Cloudlet. Therefore, Cloudlets do not have to be fixed infrastructure close to the wireless access point which alleviates the dependency on service providers. Cooperation has shown its importance among mobile networks citeabumansoor2012secure,boukerche2009cross. Thus, in this dynamic framework, mobile devices can cooperate and share resources with each other to handle tasks. Since there is a cooperation in the Cloudlet, applications on mobile devices are partitioning into components which can gain a better performance [38] comparing with the traditional VM-based Cloudlet which offloads whole application. Component offloading is a more flexible way for resource offloading for that latency-critical can be offloaded to devices in the Cloudlet while other parts can be executed on the remote Cloud server.

The inside of this dynamic Cloudlet architecture can be seen in figure 2.5 which is consisted of 3 devices and each of them is defined as a node. All the nodes in a Cloudlet host a Node Agent(NA) and the node with the most resources within a Cloudlet will be selected to host a Cloudlet Agent(CA). CA has a global view in the Cloudlet which means CA monitors the resource usage of each NA to achieve the flexible component allocation. In this way, complex decisions are handled by more powerful node which reduces

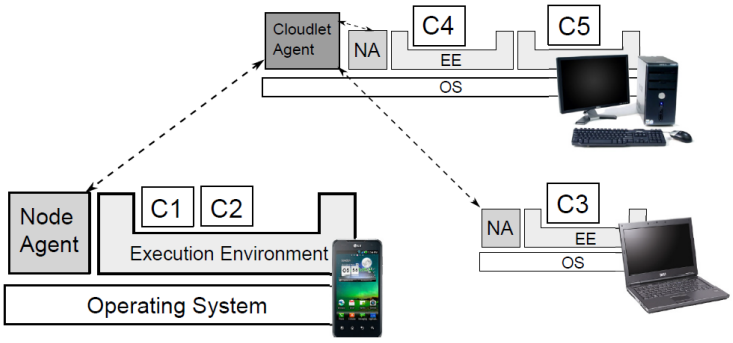

Figure 2.5: Dynamic Cloudlet architecture described in [109]

battery consumption and computing resources on other weaker nodes. Apart from this, this structure assumes that CA can communicate with each other so as to get a more convenient and easier component migration from one Cloudlet to another one. A node is composed of NA, Operating System(OS), Execution Environment(EE) and component which is part of an application. Similar with a CA, NA is responsible for observing and checking the resource usage of the whole node. Components on a node are managed by an EE which has the right to stop or start a component. One or multiple EEs run on the top of an OS which runs on virtual or real hardware. In case of violations, EE can stop a running component and if EE detects resource exhaustion, it can notify the NA which will communicate with the CA to do a migration. So, this hierarchical architecture can be interpreted as CA manages NA which manages EEs which manages components. Comparing with the way to let all EEs and nodes make decisions themselves, this solution is more scalable.

This paper also mentioned the CA selection problem. At the beginning of the formation of Cloudlet, all nodes host both NA and CA. When they connect with each other in the same LAN Internet, CA started to discover other CAs using the Service Location Protocol (SLP). After detected another CA with more powerful capability, the CA on weaker node stops and the NA registers to the stronger one. In this way, a Cloudlet with a CA which is the strongest node in the area can be formed. When nodes move in or out of the Cloudlet area, the CA will recalculate resource allocation and deployment. From the perspective of communications among CAs among different Cloudlet, all CAs are registered by their IP address to the public Internet after they sent the first uplink.

A more common architecture is to consider all mobile devices connecting to a same AP, the AP here doesn't have a server inside, as a Cloudlet [87]. In this way, the AP can be directly used as a controller or manager. Similarly, this deployment method can also be applied in VANET. Apart from deploying a Cloudlet server in the RSU which

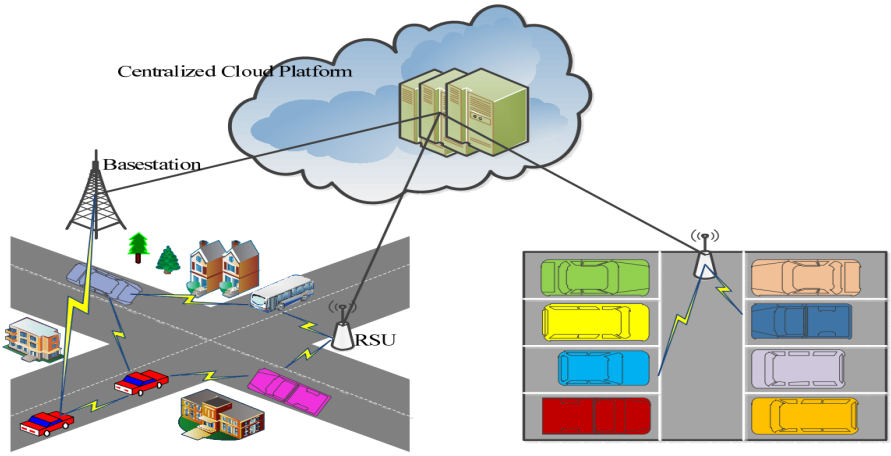

Figure 2.6: Two-tier vehicular cloud architecture [57]

we introduced as static Cloudlet in VANET. Dynamic Cloudlet among vehicles which is known as Vehicular Cloudlet(VC) [82] is also a concept that is gradually forming. The most direct idea is to form a Cloudlet between moving vehicles through V2V communication as figure 2.6 illustrated. The connection among vehicles is not fixed, it could be a tree structure [55] or mesh structure [121]. Paper [121] introduces 2 forms of VC, one is centralized and another is decentralized. In a centralized VC, there is a controller which is responsible for the creation, maintenance, and deletion of a vehicular cloud. All vehicles will virtualize their physical resources and register the virtual resources in the cloud controller. While in a decentralized VC, a vehicle will specify some vehicles as candidate cloud sites, and directly apply for resources from these vehicles. This paper doesn't specify how to select a controller or to definite it. Paper [120] gives a generally deployment of a controller: RSU plays role of cloudlet manager. However, a tree VC structure is always a centralized one, as defined in the article [55], the root vehicle of the tree is introduced in the VC as a vehicular cloud controller denoted Tree Controller.

While these papers are interested in moving vehicle Cloudlet, some other papers focused on one the parked vehicles. Paper [76] tries to integrate parked vehicles into traditional vehicular network. The parked cars are introduced as static nodes in the network compared to the moving cars. By combining both moving and stationary into same network, not only the resources capacity can be increased, the network connectivity can also be improved. ParkCast in [75] which is proposed by the same person uses parked cars on the roadside to distribute contents in urban VANETs. Figure 2.6 gives a complete VC architecture which combines Cloudlet in moving vehicles and Cloudlet in static vehicles.

Beehive framework which uses an inherent Cloudlet storage to replace the function of traditional cloud storage in Peer-to-Peer(P2P) system is proposed in paper [116]. As shown

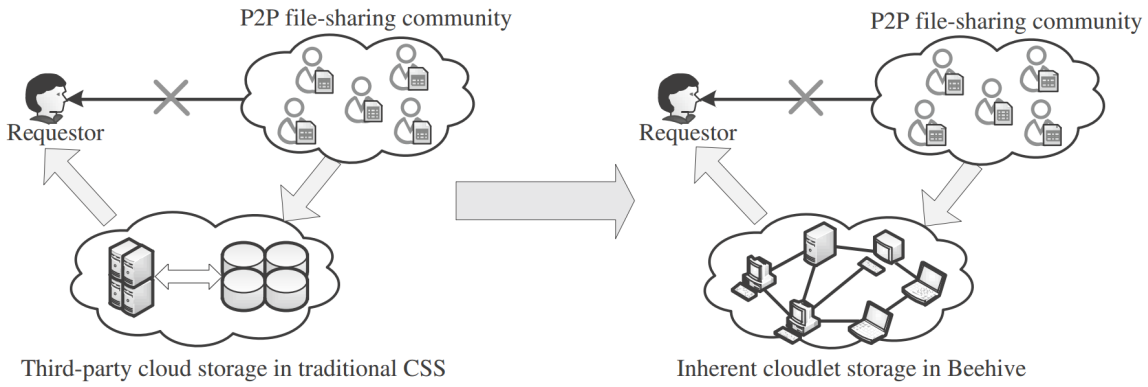

Figure 2.7: Traditional CSS vs. Beehive. Traditional CSS employs third-party cloud storage while Beehive maintains an inherent cloudlet storage [116].

in figure 2.7,the Cloudlet storage nodes in Beehive come from the original P2P community. This framework considers that many user nodes actually have abundant bandwidth/storage resources in the community and organizers these resources to build an inherent Cloudlet storage. Beehive also assumes each user in the original P2P system should contributes a certain amount of disk to serve as a Cloudlet peer in the storage pool. In Beehive, seeders upload files to the Cloudlet-based storage pool so that users can be notified and download the file. Comparing with getting files from the remoter cloud server in traditional system, getting files from this storage pool not only reduces the time spent but also reduces the cost of subscribing to the third-party cloud services. Since the number of seeders may be greater than 1, a management server is introduced to be responsible for the overall coordination.

From the perspective of the organization for the Cloudlet-based storage pool, Beehive implements a hierarchical architecture between cloud servers and Cloudlet peers. Each cloud server monitors and manages a cluster of Cloudlet peers which also has a backup node in case of malfunction in master server. All cloud servers are organized into a Distributed Hash Table(DHT) to assign devices to cloud server in a more dynamic way. As a result, the cloud server has a global view of peers distributed in the Cloudlet, requests routing will be easier. Routing is important [23] so that methods are designed in such as wireless sensor networks [114, 17, 112, 110] or underwater sensor networks [41]. DHT is also used in every Cloudlet peer to organize all the files. Each file is encoded into multiple different data blocks and then each of them is hold in the DHT by one peer in the storage pool. Master maintains pointers to all peers that store the encoded data blocks so that it can group an entire file by checking and notifying all related Cloudlet peers. The backup node will also periodically synchronize all the metadata for the master server. In case of a master server failure, DHT routing will automatically direct requests to the backup node.

The dynamic Cloudlet in the above articles are all formed by a group of devices, but the dynamic Cloudlet can also be a single device which provides extra resources for use by others [11, 2]. MObile Cloud Hybrid Architecture(MOCHA) illustrated in paper [102] is a 3-tier framework: mobile-Cloudlet-cloud. At first glance, this is the same structure as the Static Cloudlet. However, the Cloudlet in static Cloudlet is always embedded in the existing infrastructure like access point, the Cloudlet in MOCHA can be a laptop in [102] or a tank in [103]. MOCHA is dedicated to face recognition applications which increase the efficiency face detection and recognition from the respect of response time. In MOCHA, mobile devices send raw image to Cloudlet directly which is responsible for the computation partition. This paper assumes one job is consisted of different tasks which can be conducted by both Cloudlet and Cloud. So Cloudlet just need to determine how to distribute these tasks to optimize the overall Quality of Service(QoS). All the task run in parallel after the distribution.

In paper [103], MOCHA architecture is used in military operations. The mobile layer in MOCHA corresponds to the soldier's night-vision goggles which is used to acquire images. Images then will be sent to the Cloudlet which is located in a nearby tank or helicopter. Cloudlet and remote cloud servers are connected through a high-latency satellite link. MOCHA is benefited in decreasing the latency compared to sending images to remote cloud servers by mobile devices. However, MOCHA still lacks some considerations. This framework firstly doesn't take the processing capability of mobile devices into account and secondly, it doesn't consider the Cloudlet failure problem which can block the execution [94] since if part of the task is hampered, the whole result won't be incorporated.

2.3 Data Offloading in Cloudlet-based MCC

From the description about Cloudlet in section 2.1.2, the offloading types on Cloudlet are mainly divided into two types, computation offloading and data offloading. Since this thesis focuses on the studies regarding data, this section just lists the works in this area. Cloudlet is between mobile devices and cloud servers which brings low latency and high bandwidth for users benefited from the one-hop network distance. From this point of view, an efficient data access can be achieved if the data needed by mobile devices can be cached in a Cloudlet it connected with in advance. Thus, the data access time will be greatly reduced by getting data from the one-hop away server. This implementation relies on data caching technology which mainly focuses on three issues.

- Data prediction. Data pre-fetching is primarily used to collect data related to a single individual. The probability for mobile users to submit the same request shortly

is more than 70% [72]. Also, data requested by users present inter-relations and connections. Thus, there is a high possibility to fetch data that users may request in the near future in advance by analysing past individual behaviours. In this way, Qos can be improved and this is very important in wireless networks [16]. Pre-fetching improves cache hit rate and reduces data access time by identifying data with a high possibility to be accessed, fetching it, and then caching.

- Data replacement. In order to make use of the large amount of data that mobile devices download within the coverage of a Cloudlet, data caching technique is applied on the Cloudlet to collect the passing data. In a Cloudlet-based architecture, mobile devices request data from a service and download it via the Cloudlet. In this way, all communication passes through the Cloudlet. Caching of such data can be beneficial; however, just storing it in the Cloudlet brings a problem: storage waste. First, some data that have been downloaded by users are not likely to be requested by users again, but a Cloudlet still caches them. Second, a Cloudlet presents space limitations, which restricts the amount of information it can store, not allowing it to store all data that passes through it. Thus, an efficient data replacement needed to be applied to the cached data items.

- Data management on Cloudlet. This refers to the relations between data items and mobile devices which means which device holds the data item. The requirement for data management on the Cloudlet is firstly, updating and predicting the location of moving devices and secondly, finding the nearest object. In this way, when a data item is not in the Cloudlet cache storage while the data item is held by several mobile devices under the coverage. The Cloudlet can decide to send a request to the nearest mobile device according to the locations from the mapping relations.

In summary, in this section, we review some research works from these three areas: data prediction, data replacement and data management. Because Cloudlet is a relatively new concept and computation offloading is more popular than data offloading on Cloudlet, there aren't many papers involved in data offloading on Cloudlet. As a result, this section mainly introduces some technologies that can be applied to the Cloudlet-based architecture. For example, in data prediction part, the generated dataset can be transmitted to Cloudlet server which brings several advantages comparing with data prediction in traditional architectures.

2.3.1 Data Pre-fetching

In the original mobile cloud computing, mobile device-cloud architecture, predicted dataset is downloaded from the cloud to the mobile phone side in advanced [72]. For the reason that data prediction requires a lot of resources which is not suitable for processing on the computation and battery limited mobile devices. Even the mobile device is powerful enough, requesting those needed information from cloud servers to store them locally after prediction is still required. Since the emergence of Cloudlet, studies began to change the direction to transfer data from the cloud to Cloudlet rather than mobile devices.

We know that no prediction technology will always be correct which means some predicted data items won't be accessed by users. Therefore, when we use mobile devices to download all the predicted information, some of that information will not be accessed, but that information will also occupy the phone's memory and when mobile phone requests some information, every record will be going through including that useless information. This results in storage waste and energy consumption. However, when we use the Cloudlet to store all the predicted data, storage waste and energy consumption are no longer a problem given that only those data mobile phone wants to access to will be downloaded to the phone side.

In Cloudlet-based mobile computing, Cloudlet requires two kinds of information to complete the prediction; one is the mobility pattern that can predict when the user is more likely to access the network; another is user access path prediction, which is used to predict the context content that user may access in the future. The prediction in our work mainly focuses on user's access path prediction. With access knowledge in advance, a cloudlet is able to pre-fetch data from the cloud and cache the data in its storage. Following paragraphs introduces some prediction technologies such as using Markov chain as well as Cloudlet-based data pre-fetching frameworks.

Paper [106] uses Markov chain to do a prediction on the user access model and then fetches the data objects for mobile multimedia applications. Highlights in this paper are that the author proposes access-after relations to decide which page should be cached when doing a web browsing and considers compounds rather than just single items in a web page. In this paper, the proposed scheme takes the HTML as a compound rather than a single element. In the prediction part, the access model is based on compound requests since an HTML file contains many elements such as links, buttons, contexts, images, etc. And then a first-order Markov chain model is used on the access model in order to find what files should be fetched in advance whose idea is quite same with previous works whereas a highlight in this paper is that the author ignores go-backs and uses accessed-after relations between nodes (web pages) instead of accessed-from semantics.

Apart from Markov chain model, association rules are also a method to predict individual related information. Association rules are used to discover useful and invaluable information between variables in databases. The proposed Cache-miss initiated prefetch(CMIP) scheme in [101] analyses users' access traces and generates frequent itemsets as well as two kinds association rules. By applying the association rules, the method constructs two user-related datasets. The first one dataset refers to the information that user always access during past days. Another dataset maps the relationships between one item and a group data items in which when a data item is accessed, the group of data items have a high possibility to be accessed too.

Interest degree can also show users' preference. In [49], user interest degree on different web pages for each user is calculated by analysing their web access logs. If the user interest degree of one web page is high, the user is more interested in the web page. According to mobile user interest degree, a web pre-fetching algorithm is proposed to generate a web pre-fetching sequence. The input of this algorithm includes a set threshold; if user interest degree on one content is higher than the value, the data will be put into this sequence. The final sequence includes all the data that users are interested in which can be fetched by Cloudlet if the mobile device is under its coverage.

And in [85], user's access patterns are clustered so that different users' access patterns will lead to totally different clusters. According to this scheme, a proxy would fetch all objects in one cluster if the user requests one of the objects in this cluster. Traces of proxy will also be recorded by a web navigational graph. By calculating each arc's support and confidence, some of them will be cut from the graph. Using Breadth-First-Search(BFS) algorithm can partition the navigational graph so that new clusters will be produced. Using the clusters, the changes in the web users' patterns can be tracked since the proposed scheme can compute the clusters periodically.

State abstraction and reinforcement learning can also be used to analyze logs to obtain user's behaviour. In [118], a Prediction-base pre-fetching (PREP) scheme is proposed to support VCR-like operations in a P2P Video-On-Demand(VOD) system. In this scheme, the user viewing logs on the tracker are firstly collected to generate user interactive behaviour which is then used to produce a prediction-based model. Using the model, the probability to be requested of different segments of the video will be presented. Thus the content with the highest probability can be fetched in advance. A highlight in this paper is that the model can be updated with the changes of user logs.

Using other people's behaviour to predict user's future access trace is also applied in some research studies. The purpose of paper [63] is to use pre-fetching technology to predict users' behaviour in the VOD environment. The proposed scheme uses statistics

from other people who have watched the same video and then do a gossip-based statistics aggregation. After collecting all information, the access probability of any segment in the video can be estimated. According to the probability, this paper designs an optimal pre-fetching scheme to fetch the corresponding data. Guidance, a parameter for every specific user, decides which part of the video should be pre-fetched. And according to user's guidance, predictions can be done individually that means each user can seek to the positions that they are interested in.

2.3.2 Data Caching

On the other hand, there are multiple phones within the cloudlet coverage, so if some hotspot information is accessed by users for multiple times, the cloudlet only needs to store that information when it is accessed for the first time. When other users want to access the same data, they don't need to download from the cloud but directly from the cloudlet to obtain data. This speeds up the user's access to the data and improves the user experience. The technology reflected from this situation involves data caching which is mainly reflected in several parts: cache architecture, data replacement algorithm in cache and data placement in cache [101].

In our scheme, the Cloudlet server is used to cache the data passing it. Due to the limit cache space of Cloudlet server, our work focuses on the data replacement algorithm to find popular data items and remove unpopular data items. Concerning data replacement algorithm, the studies so far are mostly summarized in the following aspects: cost-based data replacement algorithm, locality-based data replacement algorithm, semantic-based data replacement algorithm and utility-based data replacement algorithm [89]. In data replacement algorithm, the most important point is to determine a standard to decide whether a data item should be removed from cache. In this section, we reviewed some classic data replacement algorithms as well as some frameworks applying caching technology which are summarized as follows.

Data Replacement Algorithms

Cost-based data replacement algorithm makes use of data replacement to improve system performance. A feature of this kind method is using evaluation function to get the replacement weight of data and then quantify these weights as a basis for replacing data. Usually, this kind algorithm replaces data item from the highest cost consumption until the cache space is sufficient to accommodate new data items. The most classic one cost-based algorithm is Stretch Access rate Inverse frequency(SAIU) which is proposed in [117] to replace

data blocks with low usage and high update rate. A gain function shown in formula 2.1 is defined for each data item to decide the replaced data.

$$gain(i) = \frac{L_i * A_i}{s_i * U_i} \tag{2.1}$$

This equation calculates the gain for data item i, where L_i represents data retrieval delay, A_i represents the access rate, U_i represents the update frequency of the data item and s_i represents the size of the data item.

Huaping shen et al. [99] designs a Greedy Dual Least Utility(GD-LU) caching mechanism which includes a cache replacement as well as a passive pre-fetching algorithm. In each replacement process, the paper aims to reduce the energy consumption of mobile devices by selecting the data item with the smallest energy consumption to be replaced. The energy consumption calculation method adopted in this paper comes from paper [53], as shown in formula 2.2. Where s is the size of data item, and m denotes the energy consumption per unit data item, and h is the energy cost for the communication load.

$$\varepsilon(s) = m * s + h \tag{2.2}$$

Due to the limited computing power and storage capacity of nodes in Mobile Ad hoc NETworks(MANET), using a small number of parameters to quantify the replacement weight of data item has been widely used. A calculation method applied in paper [81] only considers the access frequency of data items and sorts the access frequency from small to large. The measurement method is shown in formula 2.3 in which d_i represents data item i and $order(d_i)$ shows the access frequency of d_i. S denotes the size of data item i. In this data replacement policy, data with large size and the most frequently accessed data will be preferentially replaced from the cache.

$$Value(d_i) = S_i * order(d_i) \tag{2.3}$$

The main feature of locality-based cache replacement algorithm is that the data left in the cache is the data that may be accessed again by the user in the near future. We reviewed three classic locality-based cache replacement algorithms including Least Recently Used(LRU), Least Frequently Used(LFU) and Most Recently Used(MRU).

Least Recently Used(LRU) is one of the most widely used cache replacement algorithms which sorts the data items in the cache in ascending order according to the accessed time. When the cache space is insufficient, the last data item in the sequence is deleted and then insert the newly requested data into the first item of the sequence. The disadvantage of

this algorithm is that it only considers the access time of the data, and doesn't take the access frequency and other factors into consideration.

Least Frequently Used(LFU) sorts data items in ascending order according to the frequency of each data access. When the cache space is insufficient, the algorithm deletes the first item in the sequence, that is, the data with the lowest frequency of access. Then places the newly requested data in the first item of the series. The main drawback of this algorithm is that when a certain data has frequently been accessed in the past, but in the case that the user has no longer accessed recently. This data won't be replaced for a long time because it has a high frequency of access. Thus, this data still occupies the buffer space which will lead to a drop in hit rate.

Most Recently Used(MRU) [34] is optimized for defects in the LRU algorithm. The main idea is that the data items that has just been accessed may not be immediately accessed, while the data item that hasn't been accessed for a long time may be accessed. MRU sorts the data items in the same way as the LRU. The difference is that when the buffer space is insufficient, MRU removes the first item in the sequence while LRU replaces the last item. Paper [45] proved that MRU has more hits than LRU due to its tendency to retain older data.

Data Caching based Frameworks

Due to its direct importance to communication performance, data caching technology has been extensively explored in a number of works. In the mobile networks, cooperative caching scheme(COCA scheme) is a popular caching method in which mobile devices can communicate with each other and share information. This scheme has been studied in wired network [44, 92] as well as MANET [61, 95, 62]. Based on COCA scheme, paper [35] proposes a COCA with the data replication scheme in which mobile devices are divided into Low Activity Mobile(LAM) and High Activity Mobile(HAM) host according to the frequency of sending requests in this framework. Based on this, the Mobile Support Station(MSS) connecting with all devices which acts as a coordinator can copy some appropriate data to the LAM so that HAM can take advantages of the LAM replicas in this architecture. Classifying mobile devices can more reasonably allocate system resources. Thus the overall system performance can be significantly improved.

Also based on the COCA strategy, paper [36] proposes a GROup-based COoperative CAching scheme(GroCoca) in which mobile hosts with similar mobility pattern and data affinity form a group so that information can be shared within a group. In addition to this, the idea of grouping improves the similarity of requests sent by mobile devices in the same

group which also improves data hit ratio. If the COCA scheme is applied in a group with a low degree of similarity, the access times of a data time won't be very high. Without introducing a MSS as proposed in [35], Dimokas et al. [47] presents a Node Importance-based Cooperative Caching (NICoCa) scheme to measure the importance of the nodes in the network. Therefore, some important nodes can be selected to be "mediators" to coordinate caching decisions and provide information about accessing the requested data or even as caching points.

Applying data caching technologies to a Cloudlet framework, these mentioned caching schemes can also be used. For example, the grouped mobile devices in [36] can form as a dynamic Cloudlet which we introduced in section 2.2.2 and the GroCoca scheme can be applied to this Cloudlet. In NICoCa [47], the node with highest importance can be viewed as a Cloudlet. Work in [91] uses an idea similar to this to present a content distribution system called Cactse for mobile devices. In Cactse, a service manager(SM) enable mobile terminals to connect with each other directly and a Cloudlet refers to the SM together with the connecting devices. Different from COCA scheme, SM in Cactse doesn't have its cache space while just an index of files is maintained so that it knows in which mobile device the file is stored. Mobile devices in a Cloudlet request information via SM. Thus, an SM can determine if a requested file is a local one which means the index of this file can be found or the file can just be accessed online. However, Cactse doesn't consider the node mobility that is the dynamic updates of indexes on the SM don't exist when mobile devices leave or enter which can bring very high overhead in the real-life mobile network with high node mobility.

2.3.3 Data Management

In this section, we focus on the mobile data management method. As mentioned above, high overhead in Cactse [91] will be caused due to node mobility. Those data with temporal and spatial information change at a high frequency so that it is hard to update the data about mobile devices via common databases. In paper [21], a data distribution management method is designed for distributed systems. Spatio-temporal access method is designed to store and update data of moving objects [78, 111]. The main idea of this method is using parametric rectangles to index the original time-space and the most classic one is Rectangle-tree(R-tree) [59]. Figure 2.8 shows a simple example of an R-tree for 2D rectangles. Calculating a bounding rectangle in different time is important so that the enclosed mobile device can be surrounded by the same rectangle. All rectangles which show objects' positions can be recorded to form an index structure. In the figure, the indexing structure formed corresponding to the rectangles on the left is shown on the

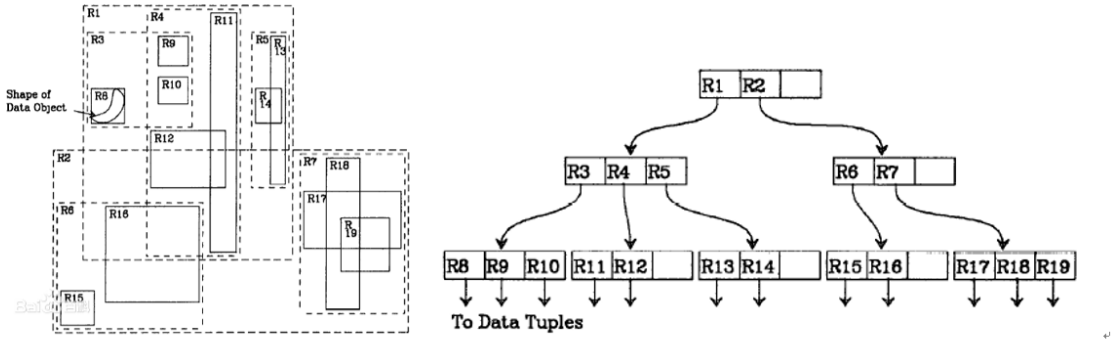

Figure 2.8: A simple example of an R-tree for 2D rectangles

right. Following reviews some research studied derived from R-tree.

Paper [96] presents a method called Time Parametrized R-tree (TPR-tree) that can index the positions of continuously moving objects in one-, two-, and three-dimensional space. The TPR-tree employs the idea of parametric bounding rectangles in the R-tree and extends the technique of R*-tree [13]. The TPR-tree uses so-called conservative bounding rectangles to enclose the moving objects and these rectangles also bound other rectangles. The TPR-tree considers the velocity of each moving object so that it can predict the future position of the moving objects. In the conservative bounding rectangles, the lower bound is set to move with the minimum speed of the enclosed points, while the upper bound is set to move with the maximum speed of the enclosed points. And when splitting the nodes in the tree, both the current positions and the velocities of the objects will be considered. But the TPR-tree only can compute the objects that move in a straight line which doesn't conform to real life.

As shown in [31], the PR-tree(Parametric R-Tree) is introduced for parametric rectangles. The algorithms described in this paper are mainly based on R-tree and is similar to the TPR-tree. But in PR-tree, each parametric rectangle has a time interval that represents the start time and the end time of its movement while in TPR-tree, there isn't such a time interval so that objects are considered to move forever. In this case, a polygon in PR-tree is used to represent a moving object. Given the movement end times, the bounding rectangles of a set of moving objects(represented as polygons) can be computed as the convex hull of the moving objects.

The paper [90] proposes Spatio-temporal Self Adjusting R-tree(STAR-tree) in order to improve the efficiency of different queries related to the moving points. In this structure, the indexes are not necessary to be reconstructed periodically and updated frequently. In

addition to that, the STAR-tree is self-adjustment in the sense that it re-organizes itself locally whenever its query performance deteriorates because the index always maintains certain auxiliary information. Besides that, the structure can both balance trade-offs between the storage used and the performance [7] along with the time spent in updating the index and answering queries.

A spatio-temporal access method called TPR*-tree based on TPR-tree is presented in [105]. In order to solve some disadvantages in TPR-tree that obtains from analysing the performance of TPR-tree, TPR*-tree is proposed. The significant changes in TPR*-tree are the improvements in insertion and deletion algorithms where TPR-tree just uses the same functions of the R*-tree. TPR*-tree improves in deletion algorithm by tightening its MBR (Minimum Bounding Rectangle) without any additional cost. Experiments show that TPR*-tree is nearly-optimal and significantly outperforms the TPR-tree under all conditions.

Chapter 3

System Model

Our work focuses on efficient data access with a Cloudlet-based approach in the mobile cloud computing paradigm. Although a nearer Cloudlet server enables computation and data offloading for mobile devices under its coverage, it still can't make a big difference in the data access area. Given a scenario in a class, the professor uploads the slides needed for the course to the cloud server so that all students can download and use them. In the case of using a Cloudlet server, the data access pattern is that every student sends a request to the Cloudlet. After receiving the request, Cloudlet forwards the request to the cloud server and then downloads the required content and finally sends it to each student. There is no essential difference between this mode and the access mode in the traditional MCC architecture. The cloud server needs to process every requests sent by students and the files need to be transmitted over and over again which causes waste in WAN usage.

In the traditional MCC structure, the requests sent by students are directly sent to the cloud server. But in the Cloudlet-based structure, these requests are needed to be forwarded by Cloudlet so that they can be received by the cloud. Therefore, the requests can be preprocessed in the Cloudlet. It can be found through observation that these requests sent by students are same, and the downloaded documents are also same. There are two ways to improve data access efficiency in this situation: one is checking every request, another is caching passing documents. Checking request means if the Cloudlet finds same requests, then it will combine them and only send one request to the cloud server. When the result of the first request is downloaded, the Cloudlet server forwards the result to all devices requesting it. But this solution can only work in a certain time like during the period after the first request was sent and before the result was received. Since after the results are sent out by Cloudlet server, even if the server receives a same request, it has no result to return, so it needs to request data from the cloud again. The second method can alleviate this problem by caching passing documents which means when

a Cloudlet server downloads the files, besides sending them to the mobile terminal, the server also cache them to its storage. So that when other devices request for the same document, the server returns it directly by searching its cache.

While the second method reduces the number of requests sent and doesn't have effect time limit, it still has its disadvantages such as balancing the limited cache on Cloudlet and the massive of files needed to cache. Although the cache space of Cloudlet is more than mobile devices, it still cannot compare with the storage space of the cloud server. Some considerations and management need to be taken in the choice of data that needs to be stored given the limited storage space. Our scheme starts with the following two points to improve data access efficiency: data generation and data distribution. Data generation mainly solves which type of data is more likely to be accessed by the user for multiple times. Data distribution discusses some schemes to distribute the generated datasets to the 3-tier Cloudlet-based MCC architecture to achieve a higher system efficiency. This chapter gives discussions on the data generation part, and next chapter will introduce the data distribution.

Therefore, from the point of data generation, two forms of data are considered to be generated in our system: individual-related and group-related data. On the individual-related data side, we consider that if we can analyse each user's behaviour to get the information that is possible requested by the user in the future and then cache them in a nearer server, the request time will be decreased. For example, given the behaviour of $userA$ that he has used news software to read the news in the past seven days and the future behaviour can be extracted as '$userA$ will use this application to read the news in the next day.' Thus, the next day's news data will be obtained in advance and transmitted to the mobile phone or the cloudlet. This operation eliminates the need to send a request to the remote cloud server when the user opens the software to read the news. For a user, this reduction is small. But in fact the number of mobile phone users is hundreds of millions, this reduction in bandwidth utilization is enormous, and the cloud server can also be released to provide more additional services.

On the group-related data side, users within the same area tend to have some common interests [107] such as visitors in the same museum are interested in requesting information about this museum and collections, students in a classroom need to download same slides. Instead of downloading files for every single user, downloading once and then store them in a shared cache space that every user can visit reduces the number of requests sent to the remote cloud server.

In this chapter, two kinds data generation model are depicted, and we denote them personal model and group model respectively. The first section illustrates personal model

which is used to generate individual-related data. The personal model applies data pre-fetching technology to predict users' future access traces from users' history logs. The second section shows the group model using data caching technology to extract popular data items among a group of mobile users showing the common interests among them. In the last part of this chapter, efficiency analysis on both models is given by comparing hit ratio among different caching and pre-fetching algorithms.

3.1 Personal Model: Data Pre-fetching Policy

Personal model mainly utilizes data pre-fetching technology to predict data items according to users' history access trace. The data items denote the information that users may access in the future. Paper [100] shows that 93% of human behaviour is predictable. Thus, if this kind of data can be predicted correctly and fetched to a nearer server such as Cloudlet in advance, the efficiency of data access will be significantly improved in the reason that the time to find data is saved.

From the research papers, many algorithms enable prediction such as dependency graph, association rules, and Markov chains. In the personal model, we make use of association rules to predict since they allow to identify the correlations between data items from the massive data sets, which suits better with the context of our work. Figure 3.5 demonstrates the personal model which shows the process to generate individual-related data items. The input of this model is the user's history access record which can be divided into different sessions after session definition. Frequent itemsets which are used to generate association rules are calculated from sessions and *minsup*, a concept in association rules. According to association rules, two types of pre-fetch datasets containing the needed data items are constructed.

We can conclude that personal model is mainly composed of 3 parts: session definition which is used to formalize and classify users' access trace to the pattern that can be dealt with; Association rules generation algorithm introduces the method to generate frequent itemsets as well as the association rules; Prefetch datasets generation algorithm is applied to generate the final individual-related data items. In this section, we will give a detailed introduction to these three parts respectively.

3.1.1 Session Definition

The session definition is related to the dataset process which will be detailed discussed in the experimental part in section 3.3.1. But here we give a simple mention of this part

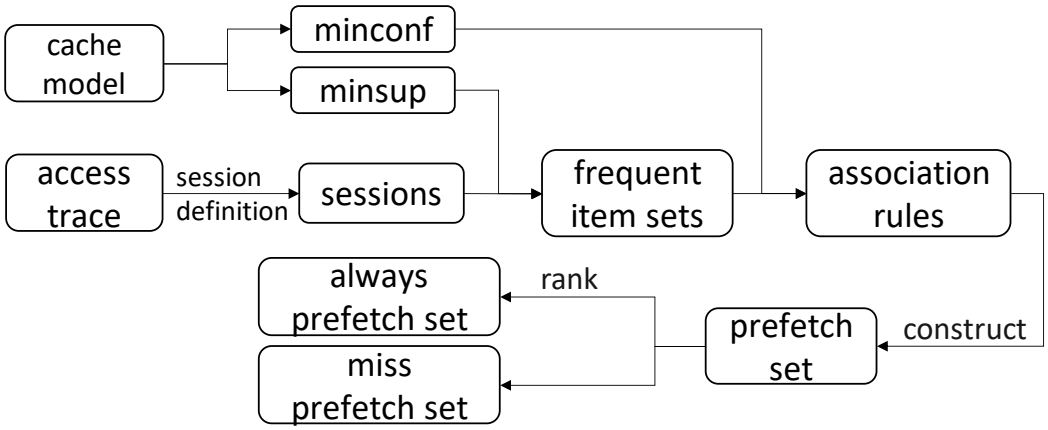

Figure 3.1: Data generation in the personal model

because the results of session definition are needed to be considered in the other two parts of the personal model. In the session definition process, user's access trace is divided into two types. One is application level sessions which refers to the user's total application browsing history for each day and does not include the record of the application's internal information. The other is inside-application sessions which integrates the user's daily browsing history inside an application.

3.1.2 Association Rules Generation Algorithm

Association rules are used to discover useful and invaluable information between variables in databases[5, 4, 10]. These rules have been applied in many research areas, such the work in [79, 29] which uses an association rule-based method to do a web personalization; from medical area, the work in [84] uses rules to improve heart disease prediction; and the work in [70] employs association rules, mining methods to extract knowledge on operations and information management. In this section, some basic concepts about association rules as well as the algorithm used to generate needed rules are presented.

Based on our scheme and inspired from paper [101], we make a corresponding translation with the concepts of association rule [4]. User's everyday access trace T can be divided into different parts according to different applications A, and every application includes a set of consecutive sessions S. One session consists of some data items I. The *support* of an data item I_n in an application A which contains sessions S, $support(I_n)$, is defined as the proportion of the sessions in S that include data item I_n. For an example, given

m sessions in an application and within them, n sessions contain data item I_n. In this situation $support(I_n) = n/m$. An *itemset* denotes a set of data items I and the number of items in an *itemset* is called the size of the *itemset*. Thus, a *k-itemset* totally contains k data items where k can also be 1. The support of an *itemset* is defined as the percentage of the sessions that have all data items in this *itemset*.

An association rule is defined as an implication of the form $X \Rightarrow Y$ in which both X and Y represent an *itemset* and the data items in this *itemset* are all accessed by user. Formulas 3.1 gives the calculation method of the *support* for an association rule R, $X \Rightarrow Y$, in which $X \cup Y$ shows the data items in both *itemset* X and Y. The confidence of an association rule R is calculated by the *support* of the rule divided by the support of X which is shown in formula 3.2.

$$Support(R) = support(X \cup Y) \tag{3.1}$$

$$Confidence(R) = \frac{(support(X \cup Y))}{(support(X))} \tag{3.2}$$

In the personal model, we need to find all the association rules in user's access trace T that have a higher *support* and $confidence$ value than minimum support *minsup* and minimum confidence $minconf$. According to paper [4], this problem can be divided into two subproblems: firstly, find all $frequent$ *itemset* which denote the *itemset* that has a greater *support* than *minsup*; secondly, using $minconf$ and $frequent$ *itemset* to generate association rules.

Cache Model

On the basis of the association rules generation algorithm proposed in paper [101] and considerations about the limited cache in mobile devices. A cache model is proposed to decide the *minsup* and $minconf$ according to different cases instead of using user-specified *minsup* and $minconf$.

In the environment of MCC, the limited cache size of mobile devices must be considered when we try to distribute the generated data items to mobile devices. Thus, we make the available cache size of mobile devices to determine the *minsup* and $minconf$. In the personal model, the size of the generated dataset is determined by the size of association rules which is actually decided by *minsup* and $minconf$. When the values are small, the generated dataset is large, and larger values will result to dataset with smaller size. Meanwhile, those data items with a higher calculated value represent that they have been accessed by users with a high frequency. When the value of *minsup* and $minconf$ are big,

the size of generated dataset is small but important. Thus, we use the available size of mobile device's cache to set the *minsup* and $minconf$ value since the generated data items are needed to be cached into mobile device in our scheme.

$$minsup\ (cache) = \begin{cases} 0.4 & cache \leq 0.2 \\ 0.6 & 0.2 < cache \leq 0.4 \\ 0.8 & 0.4 < cache \end{cases} \tag{3.3}$$

$$minconf(cache) = \begin{cases} 0.6 & cache \leq 0.2 \\ 0.8 & 0.2 < cache \leq 0.4 \\ 0.9 & 0.4 < cache \end{cases} \tag{3.4}$$

Formulas 3.3 and 3.4 show the relationships between *minsup*, $minconf$ with cache percentage of a whole cache size of mobile devices. When the cache percentage of a mobile device is low, we set both parameters at a higher value in order to generate a relatively smaller but important dataset. In this way, the mobile device can use smaller cache to satisfy most requests since those data items it cached have a higher possibility to be requested in the future than other data items. Although in this way, important data items can be generated, the dataset may also be larger than the cache size. When this occurs, the mobile device just caches the top of the dataset that as same large as its cache size.

Association Rules Generation

In this part, we need to find all *itemset* that has a greater *support* than *minsup*. Algorithm 1 gives steps to generate $frequent\ itemset$ $\{F_1, F_2...F_k\}$ and each of them is the set of all *k-itemset*. For example, F_2 denotes the set of all the *itemset* whose size is 2. The input of this algorithm is *sessions* and *minsup*. Firstly, $frequent$ 1-$itemset(F_1)$ is generated from sessions which are defined by users' access trace. By calculating *support* of every data item in sessions and comparing the value with the *minsup* got from the cache model, if the calculated value is bigger than *minsup*, the item will be inserted into F_1 as an *itemset*. In this way, we get F_1 with the form of $\{\{i_1\}, \{i_2\}...\{i_m\}\}$. Table 3.1 shows the notation used in the following algorithms.

Table 3.1: Notations for association rule generation algorithm

F_k	The set of frequent k-itemsets
F	The set of all frequent itemset
l_i, l_j	Any of the frequent (k-1) itemsets within F_{k-1}
l	Itemset within F_k
$l_i[item_k]$	The kth item in itemset l_i
R	The set of association rules
I	The set of items that could be the antecedent of a rule
s	The maximum subset of an itemset

ALGORITHM 1: Frequent itemsets generation algorithm

Input: sessions, *minsup*
Output: frequent itemsets F

```
1  F_1 = ∅
2  for every data item d in sessions do
3      if support(d) > minsup then
4          F_1 = F_1 ∪ d
5      end
6  end
7  F = F_1
8  for (k = 2; F_{k-1} ≠ ∅; k++) do
9      F_k = generateF_k(F_{k-1})
10     for every itemset l in F_k do
11         for every (k-1)-subset s in l do
12             if s ∉ F_{k-1} then
13                 remove l from F_k
14             end
15         end
16         if (support(l) < minsup) then
17             remove l from F_k
18         end
19     end
20     F = F ⋃ F_k
21 end
```

The second step shows the initialization of F which is denoted with F_1. Next step is to generate $frequent\ k\text{-}itemset(F_k)$ from F_{k-1} when the size of F_{k-1} is bigger than 1. First of all, using algorithm 2 to generate a candidate F_k by comparing any two *itemset* in F_{k-1}. The specific operation is getting two *itemset* from F_{k-1} and then comparing each data item of them. If the first k-2 data items are equal and just the last one data item is not equal, a new *k-itemset* is generated by combining these two *itemset*. A candidate $frequent\ k\text{-}itemset\ F_k$ consists of all the *k-itemset* generated from F_{k-1}. But this step will result in a large size F_k. Thus, a pruning strategy is applied in order to delete some useless data items: given each *itemset* in F_k, if the (k-1) subset of this *itemset* is not in the F_{k-1}, the *itemset* will be deleted. For example, give two 3-*itemset*: $\{i_1, i_2, i_3\}$ and $\{i_1, i_2, i_4\}$, the generated 4-*itemset* should be $\{i_1, i_2, i_3, i_4\}$. But if $\{i_2, i_3, i_4\}$ is not included in F_3, this 4-*itemset* will be deleted from the candidate F_4. After this step, the *support* of every *itemset* in the F_k will also be calculated and if the value is smaller than *minsup*, the *itemset* will be removed. The last step is to add the suitable $frequent\ k\text{-}itemset(F_k)$ into F and then the iteration will start with a larger k value to generate next $frequent\ itemset$.

ALGORITHM 2: Candidate frequent k-itemset generation

Input: F_{k-1}, *minsup*
Output: F_k

```
1  F_k = ∅
2  for each itemset l_i in F_{k-1} do
3      for each itemset l_j in F_{k-1} do
4          if l_i[1] = l_j[1] ∧ l_i[2] = l_j[2] ∧ ... ∧ l_i[k-2] = l_j[k-2] ∧ l_i[k-1] ≠ l_j[k-1]
            then
5              l = l_i ∪ l_j
6              F_k = F_k ∪ l
7          end
8      end
9      return F_k
10 end
```

The last problem in this section we need to consider is association rules generation. They are generated from the $frequent\ itemset$ as figure 3.5 shows and there are totally 2 kinds rules will be generated:

1. Generate the first kind of association rule with $frequent$ 1-*itemset*. For every data item in the 1-*itemset* of F_1, if $support(l_j[1]) \geq minconf$, then output rule "$\Rightarrow l_j[1]$".

2. From the largest $frequent\ itemset\ F_k$, using maximum true subset to generate second kind of rule, if $\frac{support(l_i)}{support(l_i[j])} \geq minconf$, then outputting an association rule: "$l_i[j] \Rightarrow \{l_i\text{-}l_i[j]\}$".

ALGORITHM 3: Association rules generation algorithm

Input: frequent itemsets F

Output: association rules R

```
1  R = ∅; I = ∅
2  //generate first kind rule
3  for every itemset l_j in F_1 do
4      I = I ∪ l_j[1]
5      if support(l_j[1]) ≥ minconf then
6          R = R ∪ {'⇒ l_j[1]'}
7      end
8  end
9  //generate second kind rule
10 k = | F |
11 while F_k ≠ ∅ and I ≠ ∅ do
12     for each itemset l_i in F_k do
13         for each dataitem l_i[j] in itemset l_i do
14             if l_i[j] ∈ I and support(l_i)/support(l_i[j]) ≥ minconf then
15                 R = R ∪ 'l_i[j] ⇒ {l_i − l_i[j]}'
16                 I = I − l_i[j]
17             end
18         end
19     end
20     k − −
21 end
22 return R
```

Algorithm 3 illustrates the method of how to generate these two kinds of association rules. The algorithm uses $frequent\ itemset\ F$ generated by algorithm 1 and $minconf$ got from cache model as inputs. The first kind rules are generated from F_1 with the form "$\Rightarrow l_j[1]$" in which the $support$ of $l_j[1]$ is larger than $minconf$. Since $minconf$ is always bigger than $minsup$, the size of generated first kind association rule will be smaller than F_1. Dataset I is denoted to all data items in F_1 that have possibility to be deduced second association rule. For the F_k, each $k\text{-}itemset$ in it can mostly generate k second

association rules. For example, if there is a 3-$itemset\{i_1, i_2, i_3\}$, the candidate rules will be '$i_1 \Rightarrow \{i_2, i_3\}$', '$i_2 \Rightarrow \{i_1, i_3\}$' and '$i_3 \Rightarrow \{i_1, i_2\}$'. In order to decrease the size of second association rules and just find the most possible data items that will be accessed after request one item. The $confidence$ of each rule will be calculated according to formula 3.4 and comparing with the $minconf$.

One problem is that same data item may generate more than one rules, like in a 4-*itemset* and 3-*itemset*, rule for i_1 can be '$i_1 \Rightarrow \{i_2, i_3, i_4\}$' and '$i_1 \Rightarrow \{i_2, i_3\}$' respectively. In order to solve this problem, we start the process from F_k so that we can first generate a larger itemset. We also make use of the itemset I, a data item will be removed from I after generating its second kind association rule. Thus, when another rule needs to be generated from a smaller itemset, the algorithm will firstly check whether there is another rule has been already generated by finding if the data item is still in dataset I.

3.1.3 Pre-fetch Datasets Generation Algorithm

Personal model constructs two pre-fetch datasets as figure 3.5 shows, $always\text{-}prefetch\ set$ and $miss\text{-}prefetch\ set$, according to users' access trace. The association rules generated from algorithm 3 have two kinds: "$\Rightarrow l_j[1]$" and "$l_i[j] \Rightarrow \{l_i\text{-}l_i[j]\}$". The first kind association rule is used to generate candidate $prefetch\ set$, such as given a rule like "$\Rightarrow i_1$", 'i_1' will be added into $prefetch\ set$. The possibility for user to access data items in the $prefetch\ set$ is high because these kind of data items were accessed by users with a high frequency in their history record. The $miss\text{-}prefetch\ set$ generated from the second kind association rule is different for each data item. For example, given two second kind association rules: '$i_1 \Rightarrow \{i_2, i_3, i_4\{$' and $i_5 \Rightarrow \{_1, i_6, i_7\}$, the $miss\text{-}prefetch\ set$ for i_1 is '$\{i_2, i_3, i_4\}$' while for i_5 is '$\{i_1, i_6, i_7\}$'.

Different from the generation method in paper [101] in which denotes the candidate $prefetch\ set$ generated directly from the first kind association rules. In the personal model, as depicted in section 3.1.1, user's access trace is divided into two types which are application level sessions and inside-application sessions. In our scheme, just the first kind association rules are generated for application level sessions so that no $miss\text{-}prefetch\ set$ will be generated. The $prefetch\ set$ for this type session indicates user's favourite application. For the inside-application sessions, both kinds association rules will be generated. Thus, there are two kinds $prefetch\ set$ and one type $miss\text{-}prefetch\ set$. The $miss\text{-}prefetch\ set$ for personal model denotes with the $miss\text{-}prefetch\ set$ generated from inside-application sessions. While for $always\text{-}prefetch\ set$, the application-related data items in the candidate $prefetch\ set$ for application level sessions will be selected. Then for each application, finding the $prefetch\ set$ for inside-application sessions. The set of all $prefetch\ set$ meeting

this standard will be pushed to a candidate *always-prefetch set*. For all data items in this candidate set, using $support(application) \times support(data_item)$ to rank them so that both the importance of application and data items are considered. For example, $app1$ gets importance of 0.9 which is higher than 0.75 of $app2$, $data_1$ in $app1$ gets 0.6 and $data_2$ in $app2$ gets 0.8, after calculation, $data_1$ finally gets 0.54 and $data_2$ gets 0.6. In this way, the *always-prefetch set* and *miss-prefetch set* in the personal model are generated.

3.2 Group Model: Data Caching Policy

Group model uses data caching technology to store data items that may be requested by users again in the future. This model is functioned among several users to generate common data items. The most significant difference between group model and personal model is that group model is a passive caching strategy whereas personal model uses active caching strategy. In other words, group model caches data locally after user accessed it which doesn't include prediction and a personal model is to predict and fetch data items in advance so that when a user first accesses a predicted data item, it is in the cache.

Using caching technology to store data items increases data access efficiency, but just caching accessed data items also brings some problems: firstly, storage waste, if a certain cached data item won't be requested by users again in the future, it still occupies some cache space; secondly, low efficiency, according to the caching strategy, as the time passes, when user accesses more and more data, the size of cached data will become larger, so that finding a data item becomes slower. The most extreme situation is that if a requested data is not in the cache and all cached data still need to be traversed which is a great waste of time and increases request latency; thirdly, cache limit, just caching all data items is very likely to cause cache overflow. Even the cache capacity of mobile devices or Cloudlet server has grown very fast in recent years; it still can not compare with the amount of data, it is impossible to hold all the data items.

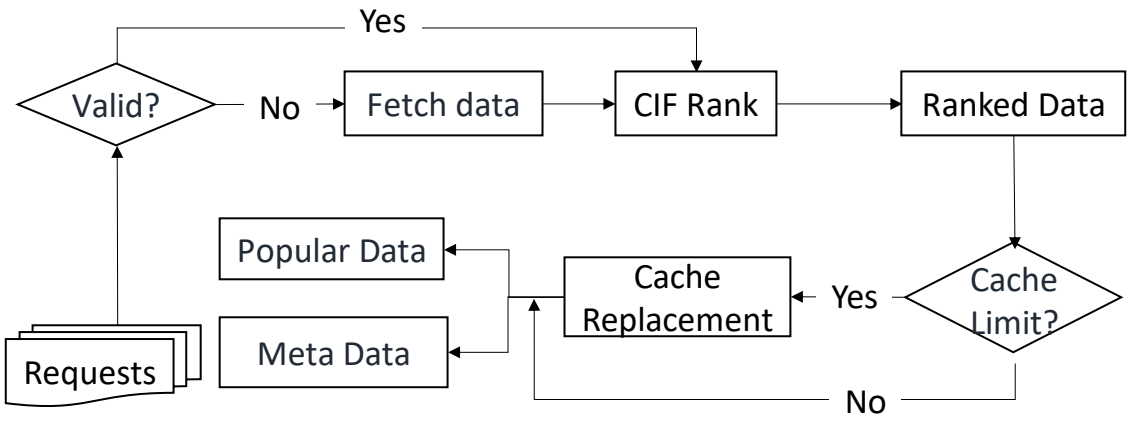

Figure 3.2: Data generation in the group model

Therefore, group model is not only used to cache data that users have accessed to, it can also reduce the use of cache by filtering some important data. Figure 3.2 shows a general process steps of a group model which relies on caching data that has been accessed in the past to satisfy users' future data access. By ranking all data items in cache according to their Cache Importance Factor(CIF), a ranked data set can be generated. Then a proposed data replacement policy is applied to reduce the footprint of the cache that generates two forms of data, popular data and meta data, and each form constitutes a dataset. These two datasets eventually form the output of the group model. In this way, this section will firstly introduce the generation of ranked dataset and then the proposed data replacement policy.

3.2.1 Data Generation

At the initiation, the cache space of group model is empty with none data items. At the beginning, because there is no data in the cache, when the model receives a request, it can only forward it to Internet to obtain data. When the result is received, in addition to meeting current requirements, this data is also stored in own cache. This situation is called cache miss. With the accumulation of time, there are more and more requests and more and more data items are saved in the cache. At this time, it is possible that data items in the cache can serve new request which is called a cache hit. After caching some data items, they need to be ranked from the most important data item to the least important one. Important data item in group model means it is likely to be accessed by users recently. In this way, the most important data item will be put front so that it can be found in the

least amount of time. It is an efficient way to spend the least time searching for the most visited data item.

After explaining the reason to sort all data items in cache, the following issue is the criteria. There are some classic standards like frequency and time. Frequency means data items are sorted according to the times they are accessed and time refers to they are ranked by the time of visit. Two representative algorithms are Least Frequent Used(LFU) and Least Recently Used(LRU): LFU sorts the data in the cache in ascending order according to frequency of data access; LRU uses the time each data item was accessed as the ranking criteria. So, LFU is frequency-related and LRU is freshness-related.

$$CIF = \begin{cases} CIF + 1 & hit\ data\ item \\ CIF \times e^{-\omega} & none\ hit\ data\ items \\ 1 & new\ data\ item \end{cases} \tag{3.5}$$

The criteria used in our group model is Cache Importance Factor(CIF) which takes both frequency and freshness into consideration. CIF is used to measure the importance of a data in the cache and the methods to calculate CIF are shown in formula 3.5. CIF takes frequency as a main standard which can be seen in the first case in formula 3.5 that is designed for the hit data item in the cache. For hit data, the operation of add 1 frequency to its original value which comes from LFU. The second case in formula 3.5 is designed for other data items which enables an exponentially decrease ranking score. For none hit items, their CIF will be multiplied with $e^{-\omega}$ in which the parameter ω is used to control the rate of decline in ranking scores. This also take the freshness of data items into consideration. For example, according to the formulas, if data item A is hit 100 times one month ago, and data item B is hit 100 times a week ago, the CIF of B is higher.

Therefore, when a cache hit occurs, the corresponding CIF will be calculated. Apart from this, there is another case which is cache miss. So when a cache miss occurs, group model will sends a request to the distant cloud server and fetch a new data item. At this time, an initial CIF value which is 1 will be assigned to this data item. Therefore, according to each data item's CIF, they can be sorted from the most important item to the least one. In this way, a ranked data set is generated.

3.2.2 Data Replacement Policy

After data generation step, we got a ranked data set which is a shared data set among a group of users. As mentioned, the cache space is limited so that it is impossible to hold

all accessed data items. Thus, data replacement policy can be used to remove some data items from cache to prevent cache overflow. For LFU and LRU, after calculated their own specific criteria, for LFU the value is the frequency and for LRU is the time, the least important data item will be removed from cache space. However, in our scheme, this solution which just removing unimportant data items is not so efficient for that those data items still have their value. Considering the scenario in which one data item has been deleted from the cache of Cloudlet(group model will be deployed in a Cloudlet which will be illustrated in next chapter) and then one user requests this data from the Cloudlet. Because the data item has been deleted, Cloudlet needs to request from cloud to retrieve the data item. However, the data item may exist in the cache of a mobile device under the coverage. It is faster to request the data from the mobile device rather than Cloud. Thus, we introduce another data type, meta-data, to store these should-be deleted data items.

In this way, it is easy to divide the ranked data set into two types after data replacement: data items stayed in cache and those should be deleted. Instead of directly removing those items as traditional data replacement policies did, group model puts them into a meta-data set. For those stayed items, popular data set is used to store them. Thus, the ranked data set is divided into popular-data set and meta-data set. In our scheme, a given threshold which is decided by cache size is used to determine whether to remove a data item to meta-data set. If the cache size is large, the threshold will be low which allows more data items to be stored and vice versa. So if cache overflow occurs, those data items with low CIF will be removed. Figure 3.3 shows the data distribution after a replacement policy of group model.

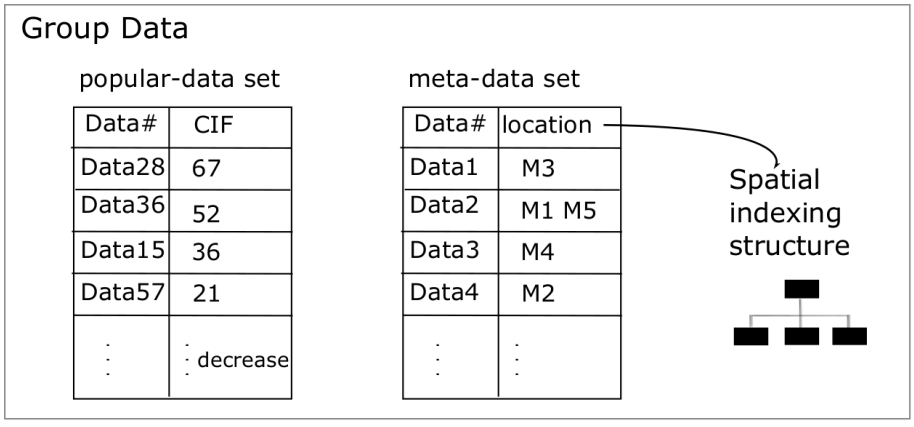

Figure 3.3: Group data distribution

Therefore, our replacement policy firstly uses CIF to filter as well as store the most popular data items into a popular-data set and secondly, those data items haven't been accessed for a long time will be transfer to the meta-data form and move to another

dataset. Data items in popular-data set are maintained in a table where all items are sorted according to the value of CIF as figure 3.3 shows. The item with the highest CIF will be ranked in the first places for that this data item is the one users are likely to access to. And putting it at first saves time to search. Data items in popular-data set are all actual data which means they can be returned directly to serve a request.

Meta-data just stores some specific information of an actual so that it won't occupy too much cache space. As figure 3.3 shows, data items in meta-data set are maintained in a table which maps the relationship between a meta-data and the location where the corresponding actual data is. Since the group model will be deployed in a Cloudlet server which is used to collect common interests among users under the Cloudlet coverage. If an actual data item is stored in the coverage, it will be in one of the mobile devices. Thus, in the meta-data set, each data item points to the mobile device which stores the actual data. For example, *meta-data*1 points to $M3$ which means the actual *data*1 is now in the mobile device 3.

However, using this mapping relationship to request corresponding actual data item, the locations of mobile devices need to be managed well. Our mechanism uses a spatial indexing structure to store the information of the mobile devices considering from two perspectives, firstly, the movement of mobile devices under is spatial which needs the storage method capable update locations quickly and efficiently; secondly, if several mobile devices contain the same meta-data, the more suitable device should be selected. Thus, given the advantages of spatial indexing structure, such as it can update and predict the location of the moving objects quickly, which can be used to find the nearest mobile device to speed up the data access. Spatial indexing structure makes use of the structure of a tree to store multi-dimension data to improve the efficiency of searching and updating. Consequently, if a mobile device enters into or leaves the coverage, the structure can update quickly. Moreover, the structure enables the scheme to find the newest neighbour of the target location. This situation indicates that if the Cloudlet finds several mobile devices holding the same data, it can find the nearest one to communicate.

3.3 Experimental Analysis

Based on the two models we have illustrated, a serious of experiments are implemented. In this section, we firstly have a detailed description and processing of the database we use. Then we have conducted a performance analysis over the dataset to evaluate cache hit ratio for both personal model and group model.

3.3.1 Dataset Process

There are two places in the experiment that need to use the dataset. The first one is the user's access trace in the experiment, and another one is the input of the personal model to obtain the prefetch dataset. A total of two different types of datasets are used in the experiment: a fictitious dataset and a mapping dataset from the real dataset. Experimenting with these two kinds datasets can help us get more convincing results. From the perspective of a fictitious dataset, according to [60], users' access trace is in line with 80/20 principle that is 80% of the requests hit 20% of the data, and this situation is consistent with the Zipf distribution. Thus, each users' access trace can be generated according to Zipf distribution. The real dataset we use in the experiment is from Boston University which is a real client-side web trace [39]. This dataset helps to better understand user behaviour and also gives researchers a real data source. The database records up to half a million client requests for WWW documents on the Internet for up to 5 months. The types of requests also consist of a variety of types including pictures, texts, videos, and so on. The contents requested in this client-web model is very similar in mobile environments where we often browse images or exchange texts on our mobile devices, so we used this database in our experiment.

For the fictional dataset, the process is simple because the generated dataset can be directly used. For every user, data items are generated as sessions in random time interval according to the Zipf distribution. Thus, this dataset can be used not only as inputs for simulation experiments but also as inputs for the personal model. But for real datasets, because the original data doesn't meet the needs of the experiment, two steps are required to deal with these traces. The first step is to map the trace of the original dataset to the simulation experiment so that they can be used as the input of simulation. The second step is session definition which is mentioned in the personal model. In this way, data items can be classified into sessions that can be processed by the personal model.

An example trace from the dataset of Boston University is '$cs17\ 786146567\ 716362\ http://www.bu.edu/lib/pics/bulogo.gif\ 1935\ 0.647182$'. Each parameter denotes with 'machine name', 'linux time stamp', 'user id', 'URL', 'document size' and 'access time' respectively. To better fit with our simulation, following steps are applied to transform the traces: firstly, users in this dataset are mapped to the mobile users in our simulation. Secondly, different URLs are mapped to different data items requested by mobile users. Thirdly, 'linux time stamp' and 'document size' are mapped to the time sent the request and the size of data item respectively. Finally, the access time of data items is mapped to a time delay on the server which means the interval between user sends a request and receive the data item.

Therefore, this dataset can be used as input in our simulations after the trace mapping

step. However, using a URL directly to perform a session definition is more complicated. Thus, the first step is to simplify the URL of each trace. Every URL corresponds to a data item in our scheme, and we want to use a simple number or string to represent the URL. But the relationships between different URL should be kept. For example, for those requested data items which come from the same web page and in the process of conversion, this relationship must be preserved. As a result, these data can be seen as coming from the same application for mobile devices. Thus, one URL with the form of '$http://cs-www.bu.edu/lib/pics/bu-logo.gif'$ can be divided into root address and content address. If corresponding this to the mobile phone side, it can be an application and contents of the application. In order to do the simplification, all URL in traces are traversed and we use numbers to replace them. For example, '$http://cs-www.bu.edu/lib/pics/bu-logo.gif'$ is mapped to '$04415'$ with '$http://cs-www.bu.edu/'$ to '$0441'$ and '$lib/pics/bu-logo.gif'$ to '$5'$. Because in the process of summing up, there are more than 1000 different root addresses. We use the first four digits to represent an application and the remaining digits represent the content accessed in the application. Thus, '$0441'$ represents an application and '$5'$ refers to a data item in an application. Other examples like '$044217'$ represents the content '$17'$ in application '$0442'$.

Table 3.2: Sample application level sessions for $user2$

Day	1	2	3	4	5	6	7
App Access	0441 0442 0443 0444 0445 0446 0447 0448	0441 0442 0443 0444 0445	0441 0442 0444 0445 0449 0447 04410	0441 0442 0443 0444 0445 04411 04412 04410	0441 0442 0443 0445 04410 04413 04414	0441 0443 0444 0445 04411 0447 04412 04413 04414 04415	0441 0442 0443 0444 04413 04414 04415 04416 04413

Before applying the personal model, we first need to divide traces into sessions so that it can be processed. There are two kinds sessions in our scheme: application level sessions and inside-application sessions. Grouping traces into application level sessions after URL simplification is easy. For each user, traces with same number prefix can be

grouped into same applications and every day's access log is a session. Table 3.2 shows a sample 7-day access log for $user2$. So for example, on the second day, $user2$ accessed '0441 0442 0443 0444 0445′ in order. Every day's trace can be extracted to an application level session. Since the input of a personal model needs several sessions, we use seven sessions which is a week as the input sessions. That is, when it is necessary to generate an app-level pre-fetched dataset for a day via the personal model, the input is the previous 7-day application level access traces. So for the sample sessions in table 3.2 and with $minsup = 0.6$, we get '$0441, 0442, 0443, 0444, 0445$′ is the $prefetch\ set$ for $user2$ from application level.

The second type session definition, inside-application sessions, needs to separate the access traces inside each application to generate inside-application $prefetch\ set$. For each URL, the root address referring to an application in a mobile device is regarded as a start of one session. By using this method, access trace inside an application will be divided into different sessions. Table 3.3 shows example sessions of one application for $user2$. After calculating the support of each data item and comparing with $minsup$, we can get the 1-$itemset$. With these sample sessions and with the $minsup = 0.6$, the 1-$itemset$ is '$04411, 04412, 04413, 04414, 04415$′ with support of 0.8 and '04417′ with 0.6. With the $minconf = 0.8$, we get the frequent itemset '$04411, 04412, 04413, 04414, 04415$′ which is the $prefetch\ set$ for application 0441.

Table 3.3: Sample inside-application level sessions for $user2$ $App0441$

Session	Requested Data Items
1	04411, 04412, 04413, 04414, 04415, 04416, 04417, 04418
2	04411, 04412, 04413, 04414, 04415
3	04411, 04412, 04413, 04414, 04415, 04419, 04417, 044110
4	04411, 04412, 04413, 04414, 04415, 044111, 04417, 044112, 044113, 044114, 044115, 044116, 044117, 044118, 044113, 044114, 044115,044116, 044111, 04417, 044112
5	04411, 04412, 04413, 04414, 04415, 044113, 044114, 044115, 044116, 044113, 044114, 044115, 044116

3.3.2 Results and Analysis

In this section, we first compared the hit ratio with different cache replacement strategies with the parameter of $hitratio$. This comparison experiment uses the fictitious dataset

which is consistent with Zipf distribution. With each user, 400 data items are generated in a random time interval. The results are shown in Figure 3.4, where four cache replacement algorithms are compared. They are FIFO, LRU, LFU and our cache replacement policy in the group model. The method in this experiment firstly generates access traces according to Zipf distribution and then gives an empty cache space with size from 1 to 40. This cache space applies different cache replacement strategies and the generated requests are used to continuously access the cache space. If there is requested data item in the cache space, the number of hits increases by one. For different caching strategies and different cache sizes, the access traces are randomly generated. Finally, this graph shows the lines that present relationship between cache size with the hit ration for the cache replacement policies.

The results of all cache replacement strategies show that the hit rate increases as the cache size increases. This is because more data items are cached, which increases the possibility of future access hitting the cache. When the cache size is 1, the cache hit rate is very low given that in this case, only two identical data items are accessed in succession results to a hit. The algorithm used in our architecture shows the highest hit ratio. Actually, with a random data sequence, the algorithm does not show this high efficiency. But for that the data accessed by users is in line with 80/20 principle, the algorithm we used can cache the 20% important data items first to provide a high hit ratio.

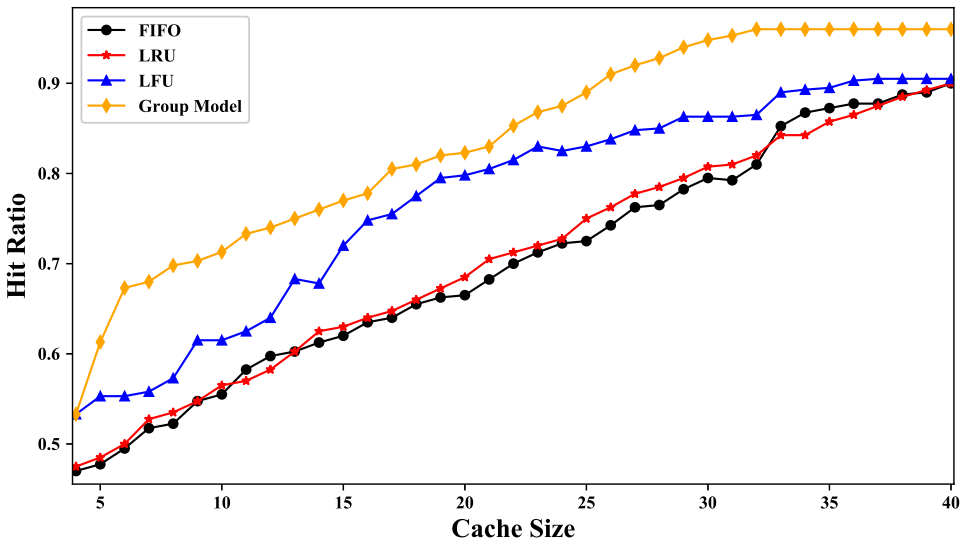

Figure 3.4: Efficiency of cache replacement policies

In addition to comparing various caching strategies, we also conducted separate experiments to research and analysis on the performance of the two models. In both experiments,

we use the real dataset. Figure 3.5 and figure 3.6 are the results of the performance of the personal model and the group model, respectively. The number of users is set to 10, 20, and 50, then experiments are conducted by giving different cache sizes. The method of calculating hit numbers adopted in these experiments is same as the previous one. One difference is that the time for each user to send a request is converted from the original 'linux time stamp' of access traces in the database. For the experiment for personal models, the cache size refers to the cache that each user owns. But for the group model, the cache is shared by all users, so the cache size represents a space that all users can access.

In the experiment for the personal model, some of the user's access traces are selected in advance and then divided these traces into different sessions according to the session definition. Then, through the personal model, prefetched datasets are generated for different types of sessions. Since the data items in the dataset has been sorted according to the probability of the user's future access which can also be referred as the importance. According to the size of the cache, part of the data items are selected to be stored in the cache. Every user has different data in the cache space. After initialized every user's cache space, an one-day access trace of the corresponding user is used to test the cached data items. This day is the day after the selected access traces that are used to generate the prefetched dataset. In this experiment, the cache hit ratio is calculated for each user and then averaged the hit ratio for all users to get the final result which is shown in figure 3.5.

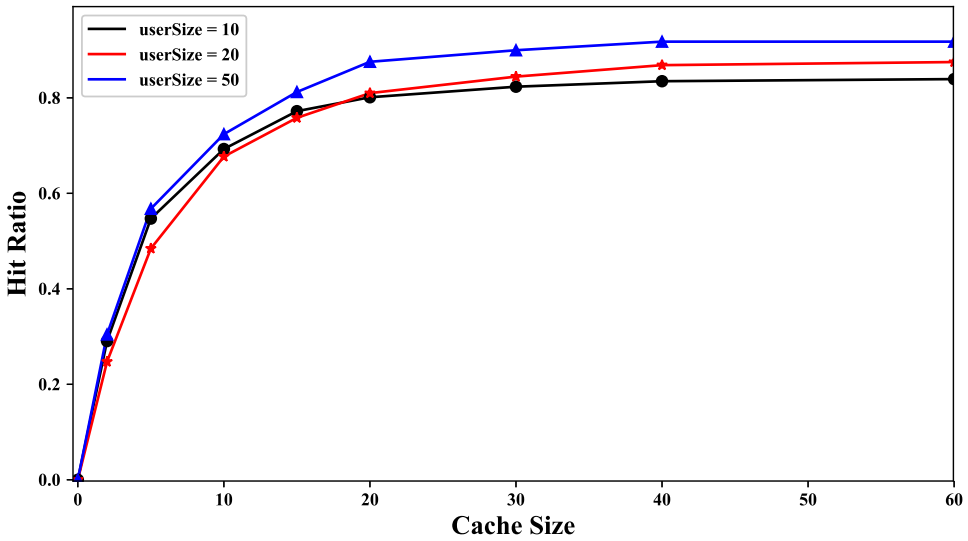

Figure 3.5: Personal model efficiency

There are 3 sets of experimental results in the figure with user size equals to 10, 20

and 50. From the perspective of all the results, one trend is that as the size of the cache increases, the cache hit rate first increases at a very fast rate and then this increase speed becomes slow and finally it approaches a stable value. This is because in the personal model, all data items are sorted. When the cache size is small, the cached data items are very important meaning that these cached items will bring many hits. But with the increase in the size of the cache, the importance of data has also decreased, so the cache hit ratio grows slowly. In the end, all experimental results are close to a certain stable value for that the size of the dataset generated by the personal model is not infinite. Therefore, when the user's cache size exceeds the size of this dataset, the number of cached data items will not change which results in the same hit ratio. A comparison among the 3 groups of different experiments shows that with the larger user size, a higher hit ratio can be got. When the number reaches 50 and the cache size for every user is 40, the hit ratio can reach 0.9. While in the same situation, the hit rate finally stabilized at about 0.8 with a user size equals to 10.

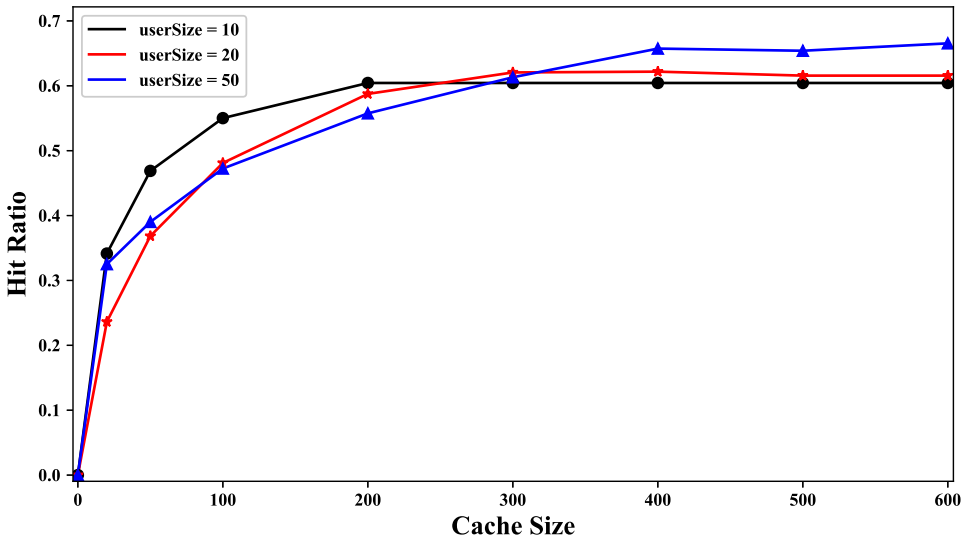

Figure 3.6: Group model efficiency

Figure 3.6 gives the result of conducting experiments for group model. First, we can see from the figure that the cache size in this experiment is much larger than the cache size in the figure 3.5. This is because in this experiment, the cache is shared by all users and each user doesn't have his own cache space. Therefore, in order to be able to compare the results with the personal model, the cache size set by this experiment is very large. When the user is 50, if the given cache size is 500, the cache size for each user is 100. This

is comparable to the cache size set in the personal model. We don't set a larger cache size for that with this set cache size, the hit ratio has reached a stable value. Therefore, it is not necessary to set a larger cache size. The 3 lines in the figure correspond to the relationship between the cache hit ratio and the shared cache size when the number of users is 10, 20, and 50 respectively. The results for group model is not the same as those for personal model which shows a better performance with larger user size. In this set of experiments, the results are fluctuating. For example, with a cache size of 100, 10 people use group model receives the highest hit ratio, followed by 20 users, and the lowest hit rate corresponds to when the number of users is 50; but when the cache size reaches 300, the order is reversed with user size equals to 50, 20 and 10 respectively. When the number of users is small, fewer requests are issued. So compare two situations: 50 users with 100 cache space and 10 users with a same cache space. Based on the number of users, the latter case has more cache space which will have better performance.

Chapter 4

Data Access Schemes

In the chapter 3, personal model and group model are proposed which are based on the individual and group behaviour respectively. In this chapter, we will introduce the distribution of these two models in the Cloudlet-based MCC framework and its corresponding data access patterns. The first two parts in this section illustrate model distribution and dataset distribution. The latter two sections give two types data access schemes, and within each of them, data access algorithm, dataset update in each component as well as experimental results are shown.

Our scheme adopts the static Cloudlet paradigm which has been introduced in Chapter 2 and all the Cloudlet servers in this framework are restricted to be separate and unrelated. Thus, the entire system can be viewed as the combination of a number of separate 3-tier models, each of which is ultimately connected to the cloud server. Each part has no connection with other models which is like a tree structure. Mobile devices connecting with a Cloudlet server can be seen as leaf nodes. Relative to the distant cloud servers, the Cloudlet as well as all devices in its coverage is local area.

Based on the static Cloudlet architecture, this section is organized in 3 parts: model distribution, dataset distribution and two proposed data access schemes. Model distribution explains the deployment of personal model as well as group model in the 3-tier architecture. If a model is applied to a component, the generated dataset will also be stored directly in that component. In the dataset distribution part, we introduces the method to distribute datasets in advance to different components. The generated dataset are transferred among different components rather than just be stored on the component the corresponding model is applied to. Based on the data distribution, the architecture is implemented and some experiments are conducted to show the system efficiency.

4.1 Model Distribution

Figure 4.1 shows a single 3-tier architecture of the whole system in which the models and datasets are also distributed to different components. This simple framework is composed of three parts: a group of mobile devices are connected with a Cloudlet server which can communicate with the remote cloud via high speed network. Personal model is placed in the cloud server and group model is in the Cloudlet server. This distribution method considers the characteristics of these two models respectively as well as the advantages and disadvantages of cloud servers and Cloudlet.

Firstly, from the perspective of personal model, this distribution considers from following several points:

- The algorithms in the personal model such as association rule generation are quite complicated which need rich resources to be conducted. From the processing power of Cloudlet and cloud, cloud server is powerful and can handle information in a more efficient way.

- In this single Cloudlet framework, all mobile devices are connected to the cloud server while the Cloudlet servers they communicate with are different. Using Cloudlet to process personal model results in resource waste because of the mobility of mobile devices. Every time when a mobile device enters into a Cloudlet coverage, the Cloudlet needs to get history behaviour of this device and run personal model to get corresponding data items. When this mobile device leaves the coverage, the Cloudlet this mobile device will enters into needs to process the personal model again given there is no connection between Cloudlet servers. However, using cloud to process personal model for each users can solve this problem since when a mobile device leaves a Cloudlet coverage and enters to another one, the personal dataset can be fetched from cloud server directly by the latter Cloudlet since this Cloudlet server also connects with the cloud server.

- Relative to Cloudlet, cloud servers can fetch the data sets from Internet directly which won't occupy bandwidth. While for Cloudlet server, if personal model is applied locally, Cloudlet needs to request the generated datasets from Internet so that the bandwidth will be occupied to some extent.

- Given the convenience provided by cloud server that each mobile device has its corresponding Virtual Image (VI) [65]. VI in the cloud handles the processing requirements for its physical mobile device so that the personal model can be processed by every VI. In this way, every VI stores information for a user which brings security for

individual data. However, Cloudlet doesn't have this mechanism given the mobile devices under its coverage are always changes. For cloud server, because every mobile devices are connected to it finally, the VI can be created for each device no matter which Cloudlet coverage it moves to.

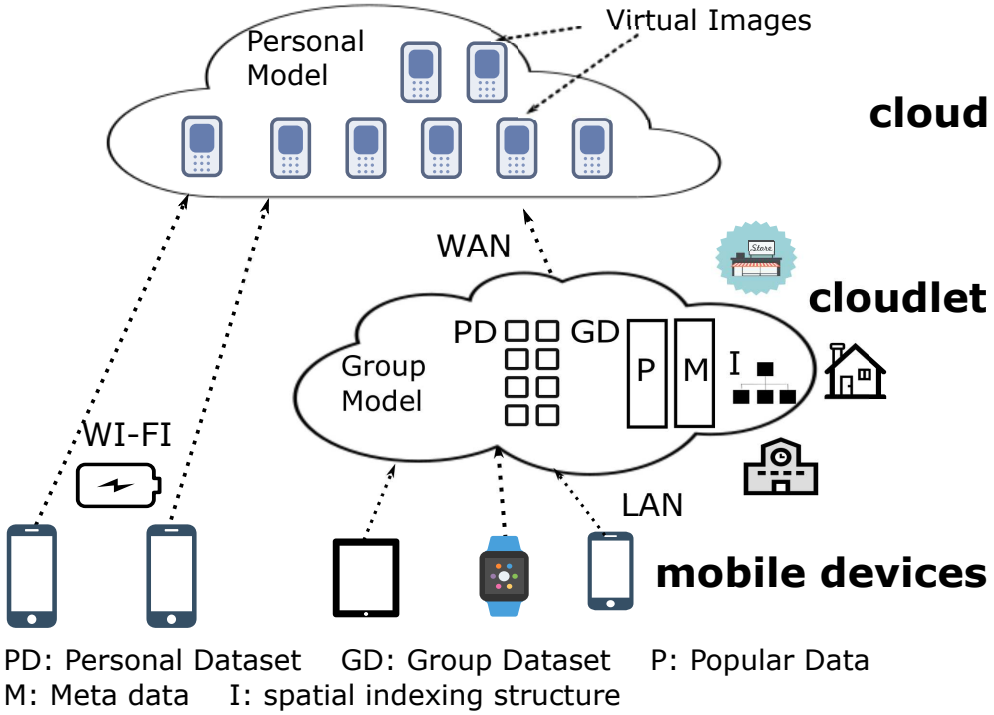

Figure 4.1: Cloudlet-based MCC architecture with models and datasets distributed in different components

Secondly, distributing group model on the Cloudlet server is motivated from following considerations. As mentioned, group model is used to cache popular data items among a group of users. So if this model is conducted on a cloud server, those data items will be calculated among all users who requests data from that cloud server. Usually a cloud server needs to serve a large number of users which results in a large popular dataset. Apart from this, given a wide range of places where a cloud server can provide services, users requesting information from same server don't have a high degree of similarity. Thus, the cached data items won't have a high possibility to be accessed by other users. However, a Cloudlet server gains natural advantage for that it has a service area such as within a coffee shop where all users in this coverage are connecting with this server. Thus, the number of mobile devices will be controllable and the calculation won't be very complex which can be handled by Cloudlet. Meanwhile, paper [72] shows people in the same area

have a tendency to share some common interests, one example is that students in the same class need to download same slides. For the Cloudlet scenario, mobile devices connecting with the same Cloudlet server are in the area which is in line with this option. And every mobile device under its coverage sends requests through the Cloudlet so that it is easy for the Cloudlet to collect and analyse data among users.

4.2 Data Distribution

After deploying the models to a suitable server, personal model in cloud server and group model in Cloudlet, we need to consider how to distribute the datasets generated by the model which plays a key role in improving data access efficiency [18, 113, 28]. In this 3-tier framework, our scheme considers three cache spaces which are in mobile devices, Cloudlet and corresponding VI in cloud respectively. The size of corresponding cache space and the distance from a mobile device are two points needed to be considered when placing the datasets. The cache space in the mobile device itself gains fast access speed given if a requested data item is cached locally, the request time can be ignored while its cache size is the smallest among the 3 cache spaces so that just a small size of data items can be caches on mobile devices. For the cache space on the VI of cloud, it has the largest storage space while the far distance leads to high latency. Cloudlet server is in a medium position in both points. Therefore, our scheme distributes a small-size but important data items in a nearer server so that access efficiency can be increased. Because important items show a relative high possibility to be accessed by users and if they are cached in a nearer server, the access time will be reduced. Even if a requested data item is not cached locally, the total access time will not be long because this data won't be accessed at a high frequency.

First of all, group model is deployed on the Cloudlet, so the generated dataset is cached on the Cloudlet directly. This dataset is a shared dataset among all users under the Cloudlet coverage and each user can access it to acquire data they need. In this way, both the popular-data set and the meta-data set are in the cache of Cloudlet. However, for individual model, it is processed for every user respectively in the corresponding VI. If all the generated data items are stored in the VI directly, users still need to send requests via the low-speed WAN. Therefore, it is necessary to distribute the generated datasets to mobile devices and Cloudlet to improve data access efficiency. Before explaining the distribution of datasets, we first review the datasets generated by the individual model and their importance.

The input sessions of personal model are divided into two types by session definition, application level sessions and inside-application sessions. The way to define these sessions

is detailed described in the section 3.1.1. For application level sessions, just the *frequent* 1-*itemset* will be generated which represents users' favourite applications. For inside application sessions, two kinds association rules as well as the corresponding *prefetch set* will be generated. Each application owns different inside application sessions and with the sessions of every application, *prefetch set* and *miss-prefetch set* are constructed according to the association rules. In conclusion, *prefetch set* is generated for both application level and inside-application sessions and *miss-prefetch set* is only constructed for inside application sessions which is shown in figure 4.2. These datasets are generated in each corresponding VI on the cloud so that the cloud server stores all the data items. But for mobile devices and Cloudlet, given the cache limit, data distribution plays an important role for system efficiency [12].

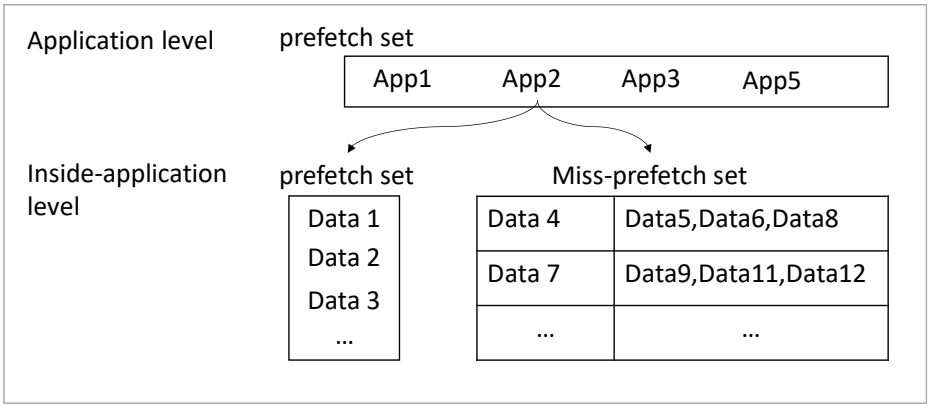

Figure 4.2: Generated datasets in personal model based on two types sessions

Firstly, the distribution of datasets on mobile phone is determined by the importance of the data items. As explained, the most important data items should be placed in the phone's cache. If the data is directly obtained from the cache of the mobile phone, the access time can be ignored [72]. So the mobile phone's cache should store the data that is most likely to be accessed by user. In section 3.1.3, *always-prefetch set* includes the data items in *prefetch set* for inside application sessions and these selected applications are in the *prefetch set* of application level sessions. From figure 4.2, *app*4 is not included in the *prefetch set* of the application level sessions so that the data items in the *prefetch set* of this application won't be added into *always-prefetch set*. But *app*4 still has its *prefetch set* and *miss-prefetch set*. Thus, *always-prefetch set* actually represents data items that have a high possibility to be requested by users. In section 3.1.2, a cache model is used to decide the value of *minsup* and *minconf* which determine the generated data set size so that a smaller but important dataset will be got. Based on this cache model, the ranked *always-prefetch set* is helpful when the size of the generated dataset

is larger than the cache size of mobile device. In this situation, the most important data items will be transmitted to mobile devices first. Thus, distribution of generated data items from the personal model on mobile devices is clearly stated.

However, the distribution of these datasets on Cloudlet is different. Given a larger cache space on Cloudlet, more data items can be stored. The cache space on the Cloudlet is divided into two parts, personal dataset and group dataset, shown in figure 4.1. In the Cloudlet-based MCC architecture, Cloudlet connects with several mobile devices under its coverage. So it can cache the personal dataset for each mobile device and the personal dataset in our scheme is divided according to the number of mobile devices. Each single storage space holds data items for corresponding devices. We know that even if the size of each dataset generated from individual model is not very large, when the number of mobile devices become more, the total size will increase fast. In this situation, Cloudlet is not possible to store all datasets for all mobile devices under its coverage especially when the number of users become more. Rather than applying a same policy as distributing dataset in mobile devices that ranking data items and fetching important items. Considering a relative large cache space on Cloudlet, a more balanced approach is applied which considers data items in all applications. Thus, in our scheme, when a mobile device enters into a Cloudlet coverage and connects to the server. The Cloudlet sends a request to get corresponding dataset. The dataset refers to the data items in the *prefetch set* of inside application which also belongs to the application level *prefetch set* which is shown in figure 4.2. Our scheme first filters some less accessed applications by only finding the applications in the application level *prefetch set* and then in these applications, just the *always-prefetch set* will be fetched by Cloudlet. These datasets will be transmitted and used to initialize a cache space for a mobile device on Cloudlet server.

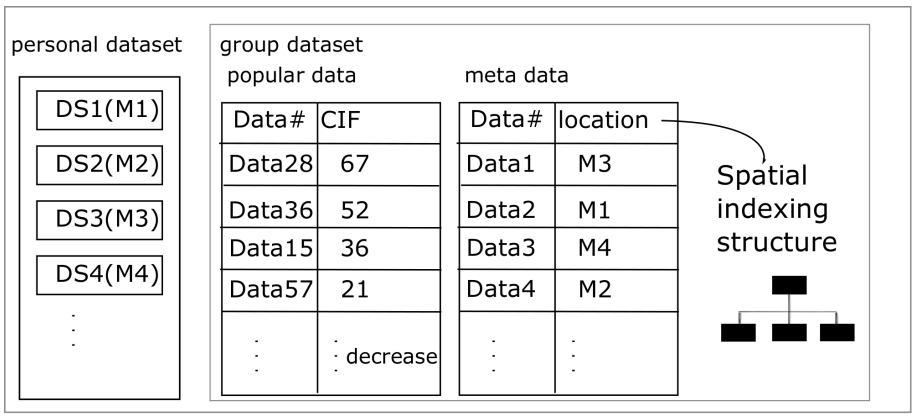

Figure 4.3: Data distribution on Cloudlet server

In summary, the whole data distribution is shown in figure 4.3. The cache space is

divided for personal dataset and group dataset respectively and the allocation proportion will be introduced in different data access schemes. For individual dataset, every mobile device under the Cloudlet coverage has its own single space and they are completely separate. While for a group model, the cache space is a shared one among all users in the coverage.

4.3 Caching and Pre-fetching Scheme

In order to reduce data access time, the proposed CAching and pre-FEtching (CAFE) scheme defines two types of requests according to the data type of our architecture: personal request and group request. Adding a property to distinguish requests improves access efficiency in that if a request is a personal one, then the Cloudlet will directly find the requested data item in the corresponding individual cache space without searching in a group dataset. Therefore, if a cache miss occurs in the individual dataset, the request will be sent to distant cloud server directly. This part introduces data access patterns using CAFE scheme and data update in different components. Experiments and comparisons between different access schemes are conducted by implementing the 3-tier architecture in the OMNET++ simulator. This work has been published in [64].

4.3.1 Data Access

Corresponding to these two request types, two different data access methods are introduced in our architecture. Upon the arrival of a request for personal data, the search follows a bottom-up approach, mobile device to the Cloud, as described in Algorithm 4. The mobile device first checks if the requested data is in its memory. If there is no information, the mobile device sends the request to the Cloudlet. The Cloudlet then searches the information in its corresponding specific data set. If there is no information, the Cloudlet then sends the request to the Cloud directly rather than finding the information in the general data set. After getting the information, the Cloudlet returns the information to the mobile device. Table 4.1 shows the notation used in the following access algorithms.

Table 4.1: Notations for data access algorithms

$req_{personal}$	Personal request
$cache_{mobile\ device}$	The cache of mobile device
$data_p$	Requested data item
$individual\ cache_{cloudlet}$	The individual cache space on Cloudlet
req_{group}	Group request
$mobile\ device_j$	The mobile device stored the requested data item
$Request$	A general request

ALGORITHM 4: Personal data access

Input: $req_{personal}$

Output: $data_p$

if $req_{personal} \in cache_{mobile\ device}$ **then**
    return $data_p$
else
    send $req_{personal}$ *to Cloudlet*
end
if $req_{personal} \in individual\ cache_{Cloudlet}$ **then**
    return $data_p$
else
    send $req_{personal}$ *to cloud*
    receive $data_p$
    return $data_p$
end

From Algorithm 5, we can see given a group request, the mobile device sends the request to the Cloudlet directly since it just caches part of specific data. In our scheme, we assume that there is no no duplication between personal data items and group data items. Thus, when the Cloudlet receives the request, it starts the search directly from the group dataset. It first finds the information in the popular dataset. If there is no information in this set, the Cloudlet then finds the information in the meta dataset. If the information can be found in the metadata, the Cloudlet gets the nearest mobile device $mobile\ device_j$ by searching $mobile\ device_{1...i}$ holding the needed data from the spatial indexing structure. After finding the mobile device, the two mobile devices are connecting with each other via a point-to-point link to share the needed information. If there is no information in

the metadata set, the Cloudlet sends the group request to the cloud. After getting the information, the Cloudlet returns the information to the mobile device.

ALGORITHM 5: Group data access

Input: req_{group}
Output: $data_p$

```
1  send req_group to Cloudlet
2  if req_group ∈ dataset_popular then
3      return data_p
4  else
5      if req_group ∈ dataset_meta then
6          mobile device_j ← search(mobile device_{1...i}, spatial indexing structure)
7          connect(mobile device_j, mobile device)
8          return data_p
9      else
10         send req_group to cloud
11         receive data_p
12         return data_p
13     end
14 end
```

4.3.2 Dataset Update

The dataset update is mainly occurs on the Cloudlet which can be divided into 2 types: one is when a mobile device enters into and leaves a Cloudlet area and another one is when a cache miss occurs on the Cloudlet. When a mobile device enters into a Cloudlet coverage, the server will initialize a individual cache space in personal dataset for the user by fetching corresponding dataset from VI on the distant cloud server. And other individual cache spaces' size will be decreased given the total allocated personal cache space is unchanged and it needs to be divided into more parts. Thus, if the size of cache data items is larger than the allocated cache space, the extra data items will be deleted. Since the data items have been ranked by importance, the most important data items will be deleted last.

The Cloudlet sends messages to all users under the coverage at regular intervals to confirm if the mobile device is still in the area. When a mobile device leaves the coverage, the link will become broken. If this happens, for group data, only the meta data pointing

to this mobile device in the spatial indexing structure is deleted. The popular dataset which stores real information doesn't change under this situation. The data items in this dataset can be continuously accessed by other mobile device even if the phone originally requesting this data left the area. Whereas for meta data, even the data still has the possibility to be accessed, the corresponding actual data item can't be accessed since the mobile device is not under the coverage. In this way, these data items are deleted directly after the connection between a mobile device and Cloudlet is broken. For personal dataset, since this cache space stores information completely related to a single user, the dataset related to the mobile device which leaves the Cloudlet coverage will be deleted directly.

When a cache miss occurs, it can be a personal dataset cache miss or a group dataset cache miss. For a group dataset, the update on data items is easy according to the data replacement policy which has been introduced in personal model section.

However, for personal dataset on the Cloudlet, the update can be analysed from 2 datasets when a cache miss occurs. In our scheme, when a request cannot be processed by the Cloudlet, the request is sent to the cloud. The dataset update on Cloudlet occurs when a dataset is transmitted from cloud to Cloudlet server. The transmitted dataset is considered from 2 aspects: *always-prefetch set* and *miss-prefetch set*. A new *always-prefetch set* is fetched by Cloudlet when a cache miss occurs on the application level and a *miss-prefetch set* is transmitted when a cache miss occurs on the data item level.

An *always-prefetch set* is transmitted when the requested data item belongs to a new application and the Cloudlet server couldn't get any information about this application in its cache. If this data item belongs to a new application and there is corresponding *always-prefetch set* but without requested data item. This cache miss is summarized in the situation where cache miss occurs on the data item level. So when the requested data item belongs to a new application, a new *always-prefetch set* will be constructed for this application according to personal model. Cloudlet fetches this new *always-prefetch set* as well as the requested data item and updated *always-prefetch set* in the mobile device's individual cache space.

On the other hand, when the requested data item still belongs to the previous application, the cloud first searches all data items in the *miss-prefetch set* of the application and if this data item can be found, the corresponding dataset as well as the requested data item will be transmitted to Cloudlet. In this situation, just the personal dataset on the Cloudlet will be updated. Another situation is that when the cloud server can't find the requested data item in the *miss-prefetch set*. At this moment, cloud generates relevant association rules and uses them to get new *miss-prefetch set*. This dataset will also be transmitted to the Cloudlet. In this situation, both the personal dataset on Cloudlet and

cloud will be updated.

So far, we have explained data update in different situations on the Cloudlet. But from the perspective of personal model, as users are sending new requests, access traces are also changed. If personal model continuous generates new datasets with updated access traces, the consumption includes power and time will be great. Thus, for *always-prefetch set*, data items in it are firstly ranked by the percentage of sessions that includes the data item of all the sessions and then ranked by the frequency of the data item accessed by user in the access trace. The data items with a higher *support* value are stored into the *always-prefetch set*. Thus, data items in *always-prefetch set* represent that these data items are accessed by users with a high frequency everyday. In the work [72], these data are called static data, showing that the static data for each user are not changing at a high frequency. As a consequence, for all the *always-prefetch set* whether in the Cloud or the mobile devices or the Cloudlet, the dataset just updates once a day when the mobile device connects to WI-Fi and is charged. In this way, the access trace is uploaded and there is no need to consider the battery of mobile devices. For *miss-prefetch set*, the update is more frequent given that firstly, when new access trace is uploaded, the *miss-prefetch set* will be updated. And secondly, when a cache miss occurs on cloud server, the *miss-prefetch set* will also be constructed and updated.

4.3.3 Experimental Analysis

For our simulations, we use the OMNET++ simulator to model the simulations scenarios; we have implemented the proposed architecture and a basic model in it. With the support of INET framework, we have configured the network in experiments as *IdealWireless* pattern: the simplest and optimal mode to avoid the interference of the network. UDP packages are used to transmit data between a Cloudlet and mobile devices. The objective of this analysis consists of showing that our architecture can reduce the latency and improve the efficiency of data access; consequently, the experiments observed the time taken to request needed data from a Cloudlet, as well as the amount of received data in the same time interval. The dataset we used is provided by Boston University which we have mentioned in chapter 3. The simulator is OMNET++ and the simulation parameters are shown in table 4.2. Figure 4.4 shows the hit ratio on Cloudlet and local mobile devices. Figure 4.5 gives a comparison between performances of personal dataset and group dataset on Cloudlet with 3 different allocation policies.

Table 4.2: Simulation parameters

Number of Mobile Users	1-50
Number of Requests	10414 items
Number of Log files	872 files
Mobile device Cache Size	0-20 data items
Cloudlet Size	0-300 data items
mobility Type	ConstSpeed Mobility
mobility speed	5mps
Request send Interval	exponential(100ms)

Figure 4.4 compares the cache hit ration on Cloudlet and mobile devices with cache size 100 and 10 respectively. With the perspective of the cache on mobile devices, just the most important 10 data items are cached according to the data distribution policy. Cloudlet's cache space is assigned to personal dataset and group dataset with 3 proportion policies. The 4 lines in figure 4.4 show the cache hit ratio on the Cloudlet and mobile devices. The 3 closer lines refer to performance of 3 different allocation policies on Cloudlet. Because no matter what strategy is adopted on the Cloudlet server, the cache space of the mobile phone is fixed in this scenario. Only the yellow line shows performance of the mobile phone cache. Since each person's mobile cache's hit ratio is independently irrelevant, the calculation we take is to first get the number of cache hits for each person and then add them up. Finally, we divide this obtained number by the sum of all user requests from which we can get our result. In this way, the performance for mobile devices is fluctuating from the resulting graph. Because for some users, the total number of requests is small, 10 data caches may result in a relatively high hit rate. While some users have more visits, the data set obtained from the personal model is very large. For a large number of visits, 10 cache spaces appear to be few. Even with some hits, the result will be very small by dividing the hits by a large number. Thus, this situation will form a more fluctuating performance line as figure 4.4 shows.

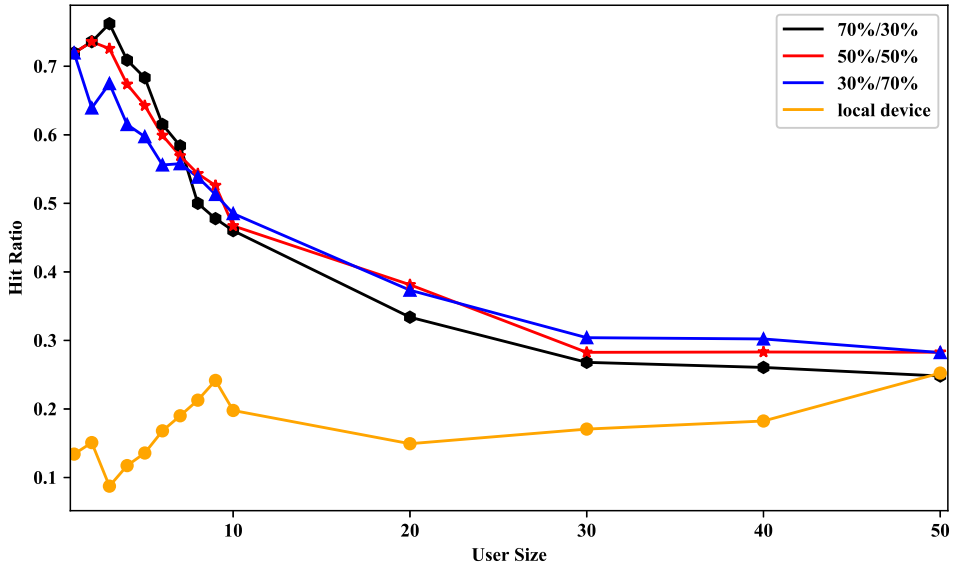

Figure 4.4: Performance on Cloudlet with different cache distribution

The overall trend of the 3 closer lines in figure 4.4 is that as the number of people increases, the performance deteriorates. When the number of users is in a relatively small size, an increase in the number of people will bring about a great change in performance. This is because at this time, cache hits are mainly due to hits on personal dataset. When the user size is small, a little increase in the number of people will bring a large decrease to the cache capacity of the personal dataset. This leads to a rapid drop in cache hits for personal data sets, which is reflected in the overall performance. As the number of users increases to a relatively large size, the overall performance gradually approaches a stable value. When the number of users increases, there is less and less personal cache space allocated to each individual. Such as, the total cache allocated to personal dataset is 50 and when user size increases to 25, the individual cache is 2 which won't bring too much for overall performance. At this time, cache hits on group datasets become a major part of performance. From the results of group model, the performance will not change much when the cache size is fixed. Thus, all 3 lines in figure 4.4 show a stable hit ratio when personal dataset plays a minor role.

Figure 4.4 also gives a comparison between the three different strategies on cache space allocation on Cloudlet with 70%/30%, 50%/50% and 30%/70% respectively. The first number is the ratio of the cache space allocated to the personal dataset and the second number is for the group dataset. The 3 distribution methods do not show a large gap in overall performance. When the number of users is relatively small, the strategy that

allocates a larger cache space to personal dataset gains more advantages. When the user size is large, a larger group dataset space becomes more dominant. Thus, the proportion of 70%/30% shows highest hit ratio at beginning while the 30%/70% distribution policy performs better at last.

The details of the performance of personal dataset and group dataset of these 3 policies can be seen in figure 4.5 respectively. All these experiments are conducted under a fixed cache size of Cloudlet which is 100 data items. The three black lines represent the performance of personal dataset under different strategies. From top to bottom, the total cache size of personal dataset is reduced resulting to the deteriorating performance. Take the cache hit rate at 0.2 as an example, when the dataset size is 70 and the number of users reaches 15, the hit rate is 0.2. When the size is reduced to 50, the corresponding number of users is 11. While the personal dataset accounts for 30% of the total cache, the hit rate drops to 0.2 when the number of users increases to 7. As the number of users increases, the hit rate approaches a value. For a total personal dataset size of 70 items, this number is 0.1. For 50 items and 30 items cache size, this number is less than 0.1. When the percentage is 30%, the hit ratio is closer to zero.

The performance of the group datasets represented by the three red lines shown in 3 different figures is more stable with respect to the performance of the personal datasets. When the number of users is small and as the number increases, the hit ratio of the group dataset increases which is very different from the performance of the personal dataset. The reason is that when the number of users is relatively small, the overlap rate among users' requests will not be very high. For example, when the user size is 1, the hit rate on group dataset will increase only if this user accesses the data he has accessed before. When the number of users is 2, if one of the users access a data that any two users have ever visited, the hit rate increases. As a result, if the number of users reaches 10, and one of the users accesses the data that this user or the other 9 users have visited, the hit rate will increase. This is different when the number of users is 1, only by accessing the data that the user himself has visited will increase the hit rate. The final approaching values of the three lines are 0.2, 0.25 and 0.3 respectively. The larger the cache size of the data set, a better performance will be got. Because given some unimportant data items which will still be accessed in the near future. When the data set size is large, these data items will be retained while they will be deleted when the data set size is relatively small. Thus, when there is access to this data, large data sets can increase the hit rate.

Figure 4.6 shows the percentage of reduced uplink requests using CAFE scheme which is defined as the ratio of the number of saved uplink requests to the total number of requests. The upload link will only be calculated if a cache miss occurs locally, which refers the situation that when a Cloudlet server sends a request to the cloud. Measurements of

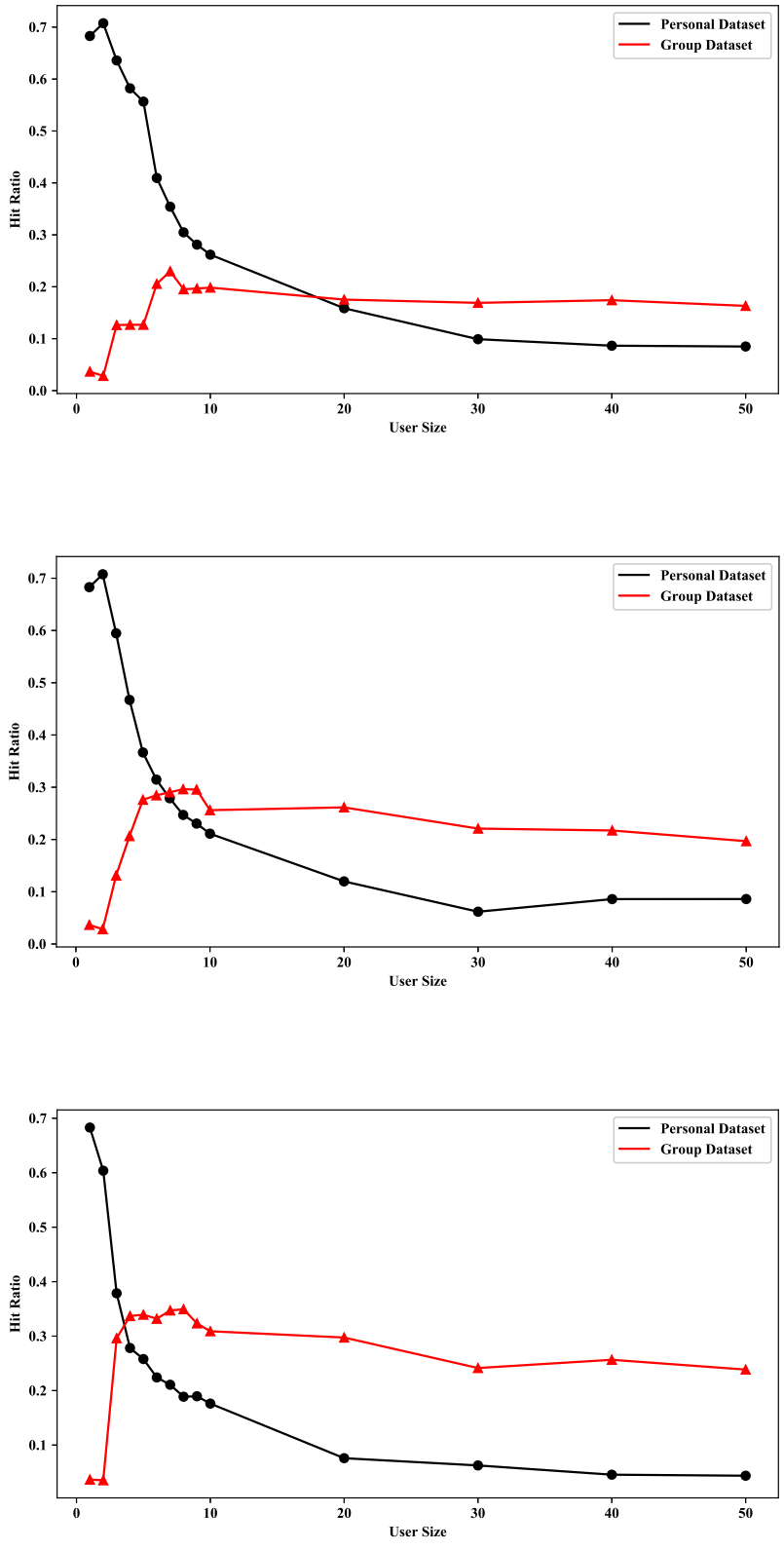

Figure 4.5: Performance of 2 datasets on Cloudlets with 70%/30%, 50%/50%, 30%/70% cache size Distribution

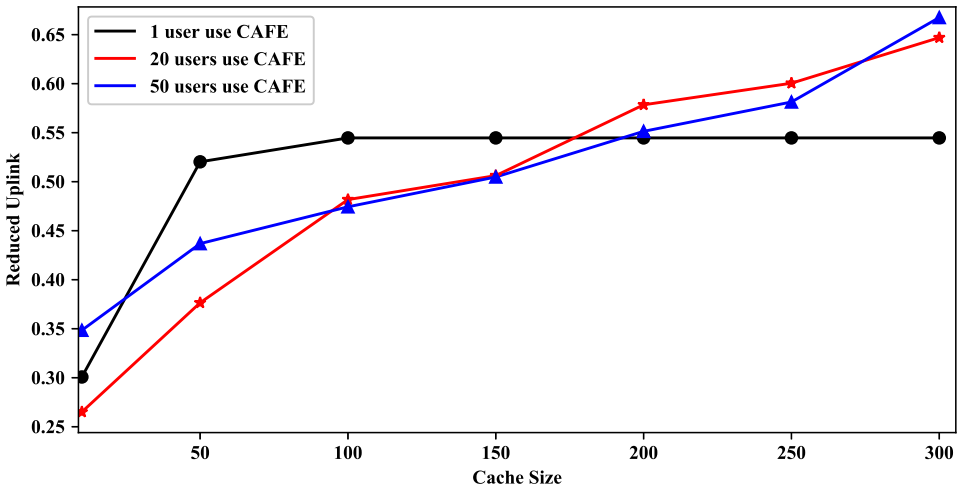

Figure 4.6: The percentage of reduced uplink requests using CAFE scheme

the reduced uplinks can be used to derive the energy consumption of local servers include mobile devices and Cloudlet. We want to show the overall consumption of local equipments, so we didn't measure the energy consumption of any single device individually. In the experiment, the number of users connected with a cloudlet reaches 50. It will be very troublesome to measure the energy consumption of each device. Thus, we just calculate the percentage of reduced uplink requests to analyse the energy consumption of the local system.

In this simulation, we set the cache size of Cloudlet from 10 to 300 and conduct the experiments when user size is 1, 20 and 50 respectively. The cache allocation policy on Cloudlet is 50%/50% for 2 datasets respectively. All mobile devices have a 10 data items cache size to store their own personal dataset. The results show that when cache size is small, the percentage of reduced uplink requests is low and when cache size becomes large, the percentage increases. From the perspective of user size, a larger user size with corresponding cache size can reduce more uplinks. The black line represents the proportion of reduced uplink requests as the cache size increases when one user uses CAFE scheme. It can be seen from figure 4.6 that when total cache size reaches 100, the percentage is the largest. Later, as the size of the cache increases, the value remains the same number. This can be explained as when there is only one user, the maximum cache size that satisfies this user's requirement doesn't exceed 100. That is, the CAFE scheme won't occupy more than 100 cache sizes during the period when user sent all requests. This figure also shows that when the Cloudlet cache size is 10, 50 users use the CAFE scheme performs better

than 20 users using the CAFE scheme. This is because when the Cloudlet cache size is 10, the cache allocated to each user is almost zero and no data items can be stored. At this time, the reduced uplink mainly comes from user device's local cache hit. Each user is independent and has no contact with each other. Their local hit rate is not the same which results in such a situation.

4.4 Adaptive CAFE Scheme

In the data distribution part, as mentioned in section 4.2, both personal dataset and group dataset are cached on the Cloudlet. When a mobile device sends a request to the server, it must has a scheme to process the request. One method is letting the request processed by the different dataset one by one until finding the data item. This method brings low efficiency from time perspective since in a worse situation, both data set need to be checked. For this reason, we separate the requests from personal one and group one in the previous CAFE scheme. In this way, for personal requests, the server doesn't need to check the group dataset and for group requests, personal dataset is not checked. The time waste problem is solved efficiently by using this method. However, when the number of users under the coverage becomes more and more, the efficiency decreases quickly since the large space personal dataset needs. Additional to this, requests by users are divided into 2 types which is not very realistic and well-implemented in real life because requests are not so easy to be categorized into one of the types.

As we know, both the processing capacity and cache ability of Cloudlet is less powerful than the cloud. If both the personal data and group data are stored on the Cloudlet as in CAFE scheme, the burden of it becomes heavy. For group dataset, taking all users as a whole leads to a smaller space possession but with a lower hit ratio. For personal dataset, it gains a higher efficiency resulting from its feature of customization. However, from the analysis of the two data types, we can easily find that when the number of users is not very large. The size of personal dataset is not very large while gaining a high efficiency. When the number of users are large, using personal cache set may cause a burden to the Cloudlet, processing also slows down. Even with a higher hit ration, the latency increases. In addition to this, given the limit space on Cloudlet, when users' size becomes larger, the allocated space for every user also decreased. And the decline rate is fast for example, there is just one user and the cache space is 100, when the size of user becomes 2, the corresponding cache space decreases to 50 which reduces by half. Thus, the number of cached data items decreases so that many requests can't be served. However, the individual dataset still occupies large cache space in this situation, for example, even

the single allocated space for each user is 5, when the size is 20, the whole personal dataset still occupies 100. Actually, 5 cache space for personal data can't serve many requests which means the 100 cache space is not so useful.

In summary, when the cache space is limit and the number of users gradually increases, the individual cache space size decreases quickly resulting in a sharp drop in hit rate whereas the hit ratio for group dataset won't change a lot for that this dataset always stores items that all users in the group are interested in. In order to reach a balance between personal and group dataset, we use an adaptive scheme based on the number of users. Given that when the user size is small, individual dataset can show a better efficiency and when the size becomes large, personal model performs better. And individual dataset is not so useful when cache space becomes small so that if this space can be allocated to group dataset which gives a great boost to the efficiency of group model. Thus in our adaptive CAFE scheme, the data set can be changed from the personal one to the group one when the number of devices is large. The data set also changes from the group one to the personal one when the number of devices becomes small. In this way, the access algorithm can be unified which is simpler than the previous one by using two access algorithms for different request types.

4.4.1 Data Access

Given that in this adaptive scheme, data accesses are unified so that just one access algorithm is shown in Algorithm 6. First of all, all requests are needed to be processed in the local cache of mobile device itself. This is different from the data access algorithm in CAFE scheme in which it is not necessary to check a request locally when it is a group one. After checking the local cache, if the data item can be found in locally, it will be returned directly. If not, the request will be sent to the Cloudlet for processing. Given that in this adaptive scheme, there is no difference among all requests. The Cloudlet must has a unified scheme to process the requests. When the Cloudlet receives a request from a mobile device, it will directly look in the current cache space for this request data. That is, if the current cache space is the personal dataset, the server will search directly at the individual dataset allocated to corresponding device. If the current cache space is the group dataset, the server will perform the same algorithm as the group request access algorithm 5 in the CAFE scheme.

ALGORITHM 6: Data access in adaptive CAFE scheme

Input: *Request*

Output: $data_p$

if $Data_{req} \in Cache_{mobile\ device}$ **then**
    return $data_p$
else
    send Request to Cloudlet
end
if $Cache_{Cloudlet} == Dataset_{personal}$ **then**
    if $data_p \in Cache_{Cloudlet}$ **then**
        return $data_p$
    end
else
    send Request to cloud
end
if $Cache_{Cloudlet} == Dataset_{group}$ **then**
    if $data_p \in Dataset_{popular}$ **then**
        return $data_p$
    else
        if $data_p \in Dataset_{meta}$ **then**
            $mobile\ device_j \leftarrow search(mobile\ device_{1...i}, spatial\ indexing\ structure)$
            $connect(mobile\ device_j, mobile\ device)$
            return $data_p$
        end
    end
else
    send Request to cloud
end

4.4.2 Dataset Update

In terms of dataset updates, this adaptive scheme is similar to the previous CAFE scheme. There will be a slight difference in some places, where we will make descriptions according to the previous dataset updates types. First, when a mobile device moves in or out of a Cloudlet coverage, the dataset update is related to which dataset the server is used at this time. When the device enters a server's coverage, if the server is using a personal dataset, the server needs to create a separate new storage space for the device like the CAFE scheme. But if it is using a group dataset, the server doesn't need to create this. Then, when a mobile device moves out of the server's coverage area, it also needs to be divided into two situations for discussion. If the server is currently in personal dataset, all personal data items associated with the removed device in this dataset must be deleted. Similarly, if the group dataset is used at this time, the information related to this device in the meta dataset and indexing spatial structure will also be deleted.

Second, the biggest update difference comes from the difference between these 2 schemes that is the dataset changes in the adaptive scheme. In order to increase the cache utilization in the server and the overall data access efficiency, the server can switch back and forth between the group dataset and the personal dataset which causes the dataset update problem. A straightforward idea to solve this issue is to delete the previously used dataset directly when switching to another dataset. However, this operation will reduce efficiency. For example, the accumulated information after using the group dataset for a period of time represents data item that all users under the coverage are interested to access. If this dataset is deleted, it will be time-consuming to re-accumulate such information. Therefore, in the adaptive scheme, when the Cloudlet changes from group dataset to the personal dataset, just the meta dataset and information stored in spatial indexing structure are deleted, and the popular data still remains. Because in our scheme, we have used a cache replacement policy in order to reduce the size of the group dataset, the actual data won't occupy too much cache. If this dataset is large enough that when the server switches to the personal dataset, it can't provide a high cache hit rate. In this case, the scheme will use importance of data items in the group dataset to trade off so that the personal dataset can occupy more than 70% of the total cache. When the server switches from the personal dataset to the group dataset, all data in the personal dataset will be deleted. This is because the personal dataset on the Cloudlet is obtained directly from the cloud which doesn't need to be generated by Cloudlet itself. Thus, the personal dataset has no data items when switching from group dataset to personal dataset. At this time, when the Cloudlet receives the first request from mobile device, it creates an individual dataset for the mobile device and fetches corresponding dataset from cloud to initiate the personal

dataset.

4.4.3 Experiments and Analysis

The data sets used in this experiment are the same as those in the CAFE scheme which are from Boston University. To evaluate the performance of the proposed adaptive scheme, we firstly conducted a performance analysis over 2 different datasets, personal dataset and group dataset, to determine the threshold for the Cloudlet to switch datasets in different situations. Secondly, for this scheme itself, performance analysis is also carried out by implementing the entire framework as well as data access algorithm in the OMNET++ simulator. Finally, we also compared the performance efficiency and energy consumption of this adaptive scheme with the previous CAFE scheme.

First of all, in the adaptive scheme, a threshold is needed to determine when the server needs to change from one data set to another. That is which dataset the Cloudlet should be used to provide service for mobile devices. Thus, we pre-set the cache size of the Cloudlet, and then run the program just with one data set, personal dataset or group dataset, with different number of users from 1 to 50. The cache size we chose is 100, 150 and 200 respectively and the hit ratio of each dataset in every situation are recorded. The results are presented in Figure 4.7. The intersection of every 2 same color lines is the point when the hit ratio is same between 2 datasets with same cache size. Overall, in the three groups of experiments, the hit ratio of personal dataset is higher while after the point, the group dataset gains a higher performance. The intersection of the 2 lines can be explained as follows. When the user size increases and since the cache size is determined, the individual cache set for each user becomes smaller so that less data items can be cached which leads to the decreasing hit ratio. For the group dataset, the hit ratio at the beginning is high since the data set is served for smaller size users. As the number of users increases, there is a larger diversity on the requests, so the hit ratio decreases. However, the hit ratio becomes stable when user size increases in the reason that the common data items among users become more consistent. In general, the hit ratio for personal dataset is high at beginning and decreases fast and finally approaches to 0 while that for group dataset is relatively stable which gives the intersection between these 2 lines.

From the figure, the three curves of the personal dataset intersect at one point when user size is 1 for that the dataset need to be fetched in advance is smaller than 100 so in these 3 situations, all the personal data items has been fetched to Cloudlet when user size is 1. In another way, the data items fetched in these 3 situations are same and with same requests, the result will be same too. For personal dataset, as the number of people increases, everyone's cacheable space continues to decline. So the results show that the

cache hit rate is always decreasing as the number of user increases. For the group dataset, the cache hit rate is increased at the beginning. Because when the number of users is small, the number of requests sent is also small and there is no need to cache too much data so that the cache space isn't used up. Therefore, when the number of mobile devices gradually increases, the usage rate of the cache space will also increase. Until all the cache space is occupied, the hit rate will decrease. Therefore, all the 3 lines of group dataset show that the hit rate rises first and then decreases. As the number of people continues to increase, the cache hit rate is related to the similarity of the data acquired between users which results in the fluctuating hit rate.

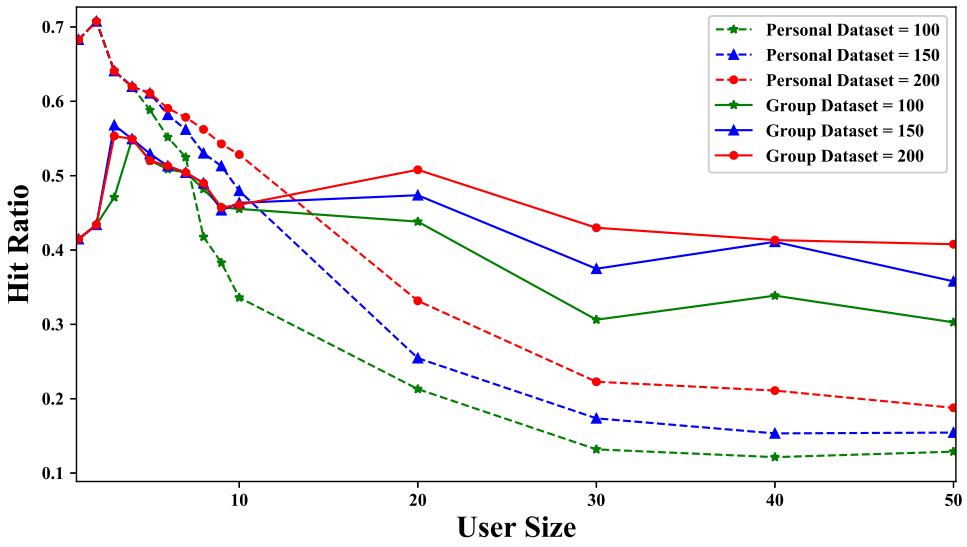

Figure 4.7: Performance of the 2 models with different cache size

In this figure, two lines of the same color have a point of intersection. By analysing these intersections, a larger cache space has a larger value of the x-axis corresponding to the intersection point. When the size of 2 datasets is 200, the user size corresponding to the intersection is 15; 150 corresponds to 11 and 100 corresponds to 8 users. From these lines, when the number is smaller than this number, the personal dataset is more efficient than the group dataset so that in this situation, the Cloudlet should use a personal dataset to get a higher efficiency. When the number is greater than this number, the group dataset performs better. From these numbers, it can be concluded that the larger a cache space is, a better performance can be got by using the personal dataset than using the group dataset with more users. Therefore, one extreme situation is that the server's cache space is large enough to support the the storage of all users' personal data. At this point, the server does not need to switch from the personal data set to the group data set.

Figure 4.8 gives the cache hit ratio on Cloudlet using the adaptive CAFE scheme. In this scheme, when the server first starts to provide services which is also the first time that a mobile device enters Cloudlet coverage and connects with the server. At this point, the cache used by Cloudlet is the personal dataset, and the cache hit ratio equals with the hit ratio of this dataset. When the number of users gradually increases until the cache switches to the group dataset, all data items in the personal dataset will be deleted. Thus, the cache hit rate is the hit rate of the group data set. When the cache switches from the group dataset to the personal dataset as the number of devices decreases lower than the threshold. At this time, the popular data in the group dataset won't be deleted. Therefore, the performance of the server is the sum of the hit ratio of the actual dataset and the personal dataset.

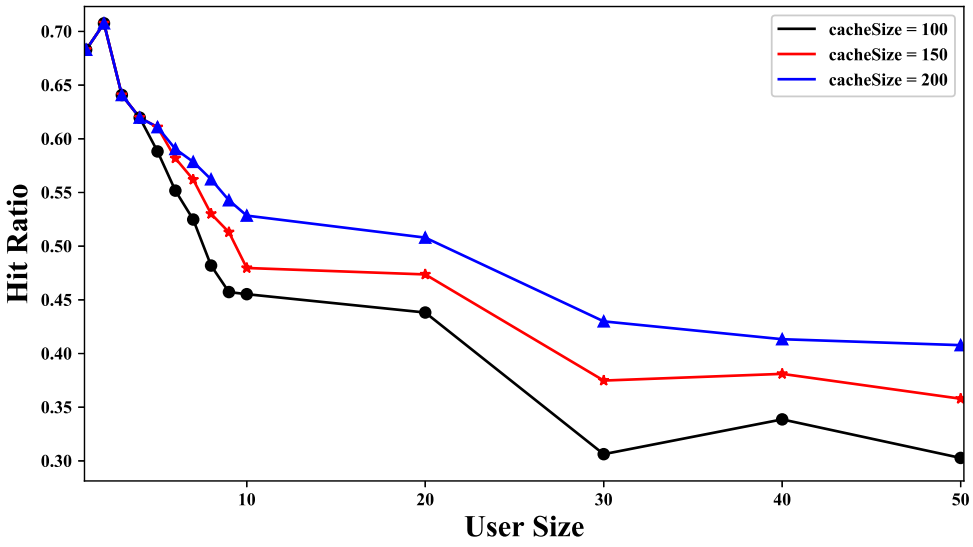

Figure 4.8: Hit ratio on Cloudlet using adaptive CAFE scheme

In this experiment, we select 3 cache sizes of Cloudlet to show the overall hit ratio of the model which uses the adaptive CAFE scheme. From Figure 4.8, 3 curves represent the hit ration of the system when the cache size of the Cloudlet is 100, 150, and 200, respectively. When the number of users increases from 1 to 50, all the 3 curves show a downward trend because the cache space on the Cloudlet for each mobile device becomes less when the user size becomes large so that less data items can be cached which results in a low efficiency. Apart from this, when the number of users increases from 1 to 10, these lines show a sharp drop such as the hit ratio of the black line dropped from 0.7 to 0.45. However, when the user size becomes larger, the trend is stable because in this situation, the dataset has been changed from personal to group, which cached some popular data items for all the

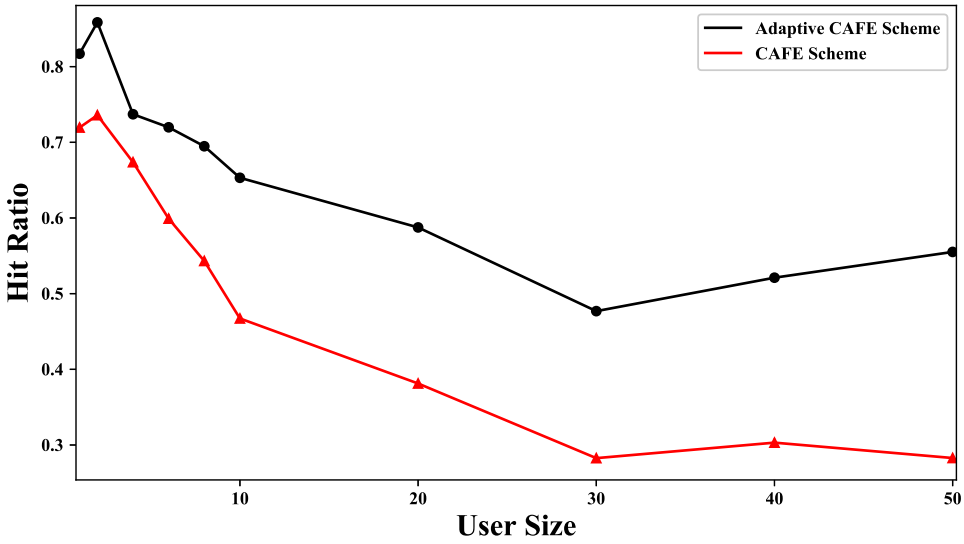

Figure 4.9: Performance comparison between adaptive CAFE scheme and CAFE scheme

users under the Cloudlet coverage. Thus, regardless the changes of individual requests, the popular data does not change for everyone at a high frequency.

In addition to the experimental analysis of the adaptive scheme, we also compared the two proposed schemes from effectiveness and energy consumption. Figure 4.9 shows the performance comparison between adaptive CAFE scheme and CAFE scheme. The performance refers to local hit ratio of this system which is the sum of hit ratio of cache on the Cloudlet and mobile devices. In this experiment, we choose 100 as cache size of the Cloudlet. The two lines in the figure represent the performance of the adaptive CAFE scheme and the previous CAFE scheme, respectively. In the CAFE scheme, we choose the 50%/50% cache allocation for personal dataset and group dataset. The reason is that in the results obtained in the experiments of CAFE scheme, this allocation gets a relatively balanced performance. The 70%/30% allocation pattern performs best when the number is small while when user size is large, it performs least. And the other distribution ratio, 30%/70%. gets the opposite result. Therefore, the result of 50%/50% distribution ratio in CAFE scheme is chose to compare with the adaptive CAFE scheme.

The results in this figure show that the adaptive scheme performs better regardless of the number of users. With the increase in the number of users, the gap between the local cache hit rates of these two kinds of schemes is also increasing. When the number of people is small, a part of the cache is assigned to the group dataset at the beginning because the CAFE scheme performs the allocation of the cache in advance. But in the adaptive scheme, the cache is not allocated in advance. When the number of mobile devices is small, personal

dataset can make use of all the cache space on the server. So the personal dataset can get the maximum hit ratio. When the number of users gradually increases under the coverage, this gap still exists. For example, when the number increases to 50, the personal dataset still occupies part of cache space according to the CAFE scheme and the space available to each person is only one data size which can not improve the hit ratio. However, according to the adaptive scheme, all the cache space is allocated to the group dataset at this time after the server switches dataset, so more cache hits are obtained in this scheme than in the CAFE scheme.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The data access efficiency in the mobile cloud computing can directly affect quality of service, so fast data access renders better user experience. In this thesis, we employed the 3-tier Cloudlet architecture that closes the distance between mobile users and cloud. Compared to adopting the traditional mobile cloud computing framework, Cloudlet improves the efficiency of data access in mobile networks by computation and data offloading. We addressed data offloading problem in the Cloudlet architecture in this thesis. In other words, suitable datasets are selected to be cached so that mobile users are able to get information directly from the one-hop away server. After selecting the appropriate framework, two data types that meet the user's needs are considered according to different situations. Therefore, in our scheme, data are classified into personal data and group data which represents the individual user behaviour and common interests of several users. At the same time, caching technology and pre-fetching techniques are used to generate corresponding datasets according to the characteristics of the data types. Based on the techniques, we propose 2 corresponding models: personal model and group model. Considering the required resources of these 2 types data generation algorithm and the capabilities of different servers, cloud and Cloudlet server, in the framework, the two models are assigned to suitable servers to run for better efficiency. Then, we propose two different data access schemes, CAFE scheme and adaptive CAFE scheme, after the models and generated datasets are distributed and in each scheme, the relevant data access algorithm and datasets update are described in detail. Experiments and simulations are implemented in the OMNET++ simulator with both fictitious dataset and real dataset in which performance of both data access schemes are evaluated and discussed from battery consumption and cache hit ratio perspectives.

5.2 Future Work

In this thesis, we focused on the data distribution and management under the Cloudlet-based mobile cloud computing architecture. There are many research interests that can be further studied and they are summarized as follows.

- Mobility prediction of a mobile device. This thesis mainly researches on access pattern prediction of individual users in order to fetch suitable datasets to Cloudlet in advance. Mobility prediction[8, 80] is another kind of important prediction in Cloudlet-based mobile cloud computing environment which and when mobile device has a possibility to connect with a Cloudlet[86, 122]. Information such as using GPS [27] to get location can be used to do the prediction in mobile wireless networks. Mobility prediction can be combined with access prediction to make the system more effective by first predicting possible datasets and then fetching them to a Cloudlet according to mobility prediction. Mobility prediction helps locate the future location of the device and the importance of localization can be found in papers [22] [46] [26] [83]. In this way, the Cloudlet can fetch datasets for mobile devices in advance if a mobile device shows a possibility to enter into its coverage.

- Data access in group Cloudlet architecture. Besides single Cloudlet architecture, group Cloudlet is also a research direction. In the group Cloudlet, there is a connection between different Cloudlets so that datasets can be transmitted from one Cloudlet to another Cloudlet which can be referred as data propogation [9]. Thus, based on our datasets distribution, some related datasets such as personal dataset don't need to be deleted when a mobile device leaves a Cloudlet coverage. These datasets can be sent to the new Cloudlet by mobility prediction.

- Computation offloading to a Cloudlet. Data offloading and computation offloading are 2 main research areas in Cloudlet-based mobile cloud computing. There are some issues in computation offloading including offloading objective which usually refers to performance and energy [42] [37] [56]; offloading decision which indicates whether and when an application should be offloaded; partition granularity[86] that means the smallest size of the offloaded unit which can be divided into coarse-grained and fine-grained [42].

References

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

[2] Saeid Abolfazli, Zohreh Sanaei, Abdullah Gani, Feng Xia, and Wei-Ming Lin. Rmcc: Restful mobile cloud computing framework for exploiting adjacent service-based mobile cloudlets. In *Proceedings of IEEE 6th International Conference on Cloud Computing Technology and Science*, CloudCom, pages 793–798, 2014.

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

[4] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM International Conference on Management of Data*, SIGMOD, pages 207–216, 1993.

[5] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, volume 1215 of *VLDB*, pages 487–499. VLDB Endowment, 1994.

[6] Ejaz Ahmed, Abdullah Gani, Mehdi Sookhak, Siti Hafizah Ab Hamid, and Feng Xia. Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges. *Journal of Network and Computer Applications*, 52:52–68, 2015.

[7] Elie El Ajaltouni, Azzedine Boukerche, and Ming Zhang. An efficient dynamic load balancing scheme for distributed simulations on a grid infrastructure. In *Proceedings of DS-RT'08*, pages 61–68. IEEE Computer Society, 2008.

[8] Ian F Akyildiz and Wenye Wang. The predictive user mobility profile framework for wireless multimedia networks. *IEEE/ACM Transactions On Networking*, 12(6):1021–1035, 2004.

[9] Thanasis Antoniou, Ioannis Chatzigiannakis, George Mylonas, Sotiris Nikoletseas, and Azzedine Boukerche. A new energy efficient and fault-tolerant protocol for data propagation in smart dust networks using varying transmission range. In *Proceedings of ANSS'04*, page 43. IEEE Computer Society, 2004.

[10] Tarek M Anwar, Howard W Beck, and Shamkant B Navathe. Knowledge mining by imprecise querying: A classification-based approach. In *Proceedings of IEEE 8th International Conference on Data Engineering*, ICDE, pages 622–630, 1992.

[11] Abdalla Artail, Karim Frenn, Haidar Safa, and Hassan Artail. A framework of mobile cloudlet centers based on the use of mobile devices as cloudlets. In *Proceedings of IEEE 29th International Conference on Advanced Information Networking and Applications*, AINA, pages 777–784, 2015.

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

[13] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of ACM Sigmod Record*, volume 19, pages 322–331. ACM, 1990.

[14] Mukta Bhatele and Anshita Khare. Ad-hoc mobile cloud computing modeling using cloudlet approach with secure data transfer. *International Journal of Modern Engineering & Management Research*, 5(2):39–46, 2017.

[15] Azzedine Boukerche and Kaouther Abrougui. An efficient leader election protocol for mobile networks. In *Proceedings of IWCMC'06*, pages 1129–1134. ACM, 2006.

[16] Azzedine Boukerche, Regina B Araujo, and Leandro Villas. A wireless actor and sensor networks qos-aware routing protocol for the emergency preparedness class of applications. In *Proceedings of LCN'06*, pages 832–839. IEEE, 2006.

[17] Azzedine Boukerche and Amir Darehshoorzadeh. Opportunistic routing in wireless networks: Models, algorithms, and classifications. *ACM Computing Surveys*, 47(2):1–32, 2015.

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

[19] Azzedine Boukerche, Sungbum Hong, and Tom Jacob. An efficient synchronization scheme of multimedia streams in wireless and mobile systems. *IEEE transactions on Parallel and Distributed Systems*, 13(9):911–923, 2002.

[20] Azzedine Boukerche, Anahit Martirosyan, and Richard Pazzi. An inter-cluster communication based energy aware and fault tolerant protocol for wireless sensor networks. *Mobile Networks and Applications*, 13(6):614–626, 2008.

[21] Azzedine Boukerche, Nathan J McGraw, Caron Dzermajko, and Kaiyuan Lu. Grid-filtered region-based data distribution management in large-scale distributed simulation systems. In *Proceedings of ANSS'05*, pages 259–266. IEEE, 2005.

[22] Azzedine Boukerche, Horacio ABF Oliveira, Eduardo F Nakamura, and Antonio AF Loureiro. Localization systems for wireless sensor networks. *IEEE wireless Communications*, 14(6), 2007.

[23] Azzedine Boukerche, Richard Werner N Pazzi, and Regina B Araujo. Hpeq a hierarchical periodic, event-driven and query-based wireless sensor network protocol. In *Proceedings of LCN'05*, pages 560–567. IEEE, 2005.

[24] Azzedine Boukerche, Richard WN Pazzi, and Jing Feng. An end-to-end virtual environment streaming technique for thin mobile devices over heterogeneous networks. *Computer Communications*, 31(11):2716–2725, 2008.

[25] Azzedine Boukerche and Yonglin Ren. A security management scheme using a novel computational reputation model for wireless and mobile ad hoc networks. In *Proceedings of PE-WASUN'08*, pages 88–95. ACM, 2008.

[26] Azzedine Boukerche, Cristiano Rezende, and Richard W Pazzi. Improving neighbor localization in vehicular ad hoc networks to avoid overhead from periodic messages. In *Proceedings of GLOBECOM'09*, pages 1–6. IEEE, 2009.

[27] Azzedine Boukerche and Steve Rogers. Gps query optimization in mobile and wireless networks. In *Proceedings of ISCC'01*, pages 198–203. IEEE, 2001.

[28] Azzedine Boukerche, Amber Roy, and Neville Thomas. Dynamic grid-based multicast group assignment in data distribution management. In *Proceedings of DS-RT'00*, page 47. IEEE, 2000.

[29] Azzedine Boukerche and Samer Samarah. A novel algorithm for mining association rules in wireless ad hoc sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 19(7):865–877, 2008.

[30] Azzedine Boukerche and Damla Turgut. Secure time synchronization protocols for wireless sensor networks. *IEEE Wireless Communications*, 14(5), 2007.

[31] Mengchu Cai and Peter Revesz. Parametric r-tree: An index structure for moving objects. In *Proceedings of the 10th International Conference on Management of Data*, COMAD, pages 57–64. Tata McGraw-Hill, 2000.

[32] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[33] Vinay Chamola, Chen-Khong Tham, and GS S Chalapathi. Latency aware mobile task assignment and load balancing for edge cloudlets. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications Workshops*, PerCom Workshops, pages 587–592, 2017.

[34] Hong Tai Chou and David J. DeWitt. An evaluation of buffer management strategies for relational database systems. *Algorithmica*, 1(1):311–336, 1986.

[35] Chi-Yin Chow, Hong Va Leong, and Alvin Chan. Peer-to-peer cooperative caching in mobile environments. In *Proceedings of IEEE 24th International Conference on Distributed Computing Systems Workshops*, ICDCSW, pages 528–533, 2004.

[36] Chi-Yin Chow, Hong Va Leong, et al. Grococa: Group-based peer-to-peer cooperative caching in mobile environment. *Journal on Selected Areas in Communications*, 25(1):179–191, 2007.

[37] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of ACM 6th Conference on Computer Systems*, EuroSys, pages 301–314, 2011.

[38] Byung-Gon Chun and Petros Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS '10, page 7, 2010.

[39] Sarah Clinch, Jan Harkes, Adrian Friday, Nigel Davies, and Mahadev Satyanarayanan. How close is close enough? understanding the role of cloudlets in supporting display appropriation by mobile users. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications*, PerCom, pages 122–127. IEEE, 2012.

[40] LAN/MAN Standards Committee et al. Ieee standard for local and metropolitan area networks: Overview and architecture (en línea). *New York-NY-USA. The Institute of Electrical and Electronics Engineers Inc*, 2002.

[41] Rodolfo WL Coutinho, Azzedine Boukerche, Luiz FM Vieira, and Antonio AF Loureiro. Geographic and opportunistic routing for underwater sensor networks. *IEEE Transactions on Computers*, 65(2):548–561, 2016.

[42] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of ACM 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys, pages 49–62, 2010.

[43] Felipe Cunha, Leandro Villas, Azzedine Boukerche, Guilherme Maia, Aline Viana, Raquel AF Mini, and Antonio AF Loureiro. Data communication in vanets: Protocols, applications and challenges. *Ad Hoc Networks*, 44:90–103, 2016.

[44] Michael D Dahlin, Randolph Y Wang, Thomas E Anderson, and David A Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI, page 19, 1994.

[45] Shaul Dar, Michael J Franklin, Bjorn T Jonsson, Divesh Srivastava, Michael Tan, et al. Semantic data caching and replacement. In *VLDB*, volume 96, pages 330–341, 1996.

[46] Horacio Antonio Braga Fernandes De Oliveira, Azzedine Boukerche, Eduardo Freire Nakamura, and Antonio Alfredo Ferreira Loureiro. An efficient directed localization recursion protocol for wireless sensor networks. *IEEE Transactions on Computers*, 14(5):677–691, 2008.

[47] Nikos Dimokas, Dimitrios Katsaros, and Yannis Manolopoulos. Cooperative caching in wireless multimedia sensor networks. *Mobile Networks and Applications*, 13(3-4):337–356, 2008.

[48] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18):1587–1611, 2013.

[49] Cong Du and Suozhu Wang. Research on mobile web cache prefetching technology based on user interest degree. In *Proceedings of the 3rd International Conference on Logistics, Informatics and Service Science*, LISS, pages 1253–1258. Springer, 2015.

[50] Mourad Elhadef, Azzedine Boukerche, and Hisham Elkadiki. A distributed fault identification protocol for wireless and mobile ad hoc networks. *Journal of Parallel and Distributed Computing*, 68(3):321–335, 2008.

[51] Elias C Eze, Sijing Zhang, and Enjie Liu. Vehicular ad hoc networks (vanets): Current state, challenges, potentials and way forward. In *Proceedings of IEEE 20th International Conference on Automation and Computing*, ICAC, pages 176–181, 2014.

[52] Qiang Fan and Nirwan Ansari. Workload allocation in hierarchical cloudlet networks. *Communications Letters*, 22:820–823, 2018.

[53] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings of IEEE Conference on Computer Communications*, volume 3 of *INFOCOM*, pages 1548–1557. IEEE, 2001.

[54] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[55] Mouna Garai, Slim Rekhis, and Noureddine Boudriga. Communication as a service for cloud vanets. In *Proceedings of IEEE Symposium on Computers and Communication*, ISCC, pages 371–377, 2015.

[56] Mark S Gordon, Davoud Anoushe Jamshidi, Scott A Mahlke, Zhuoqing Morley Mao, and Xu Chen. Comet: Code offload by migrating execution transparently. In *10th USENIX Symposium on Operating Systems Design and Implementation*, volume 12 of *OSDI*, pages 93–106, 2012.

[57] Lin Gu, Deze Zeng, and Song Guo. Vehicular cloud computing: A survey. In *Proceedings of IEEE Globecom Workshops*, GC Wkshps, pages 403–407, 2013.

[58] Shichao Guan, Robson Eduardo De Grande, and Azzedine Boukerche. A novel energy efficient platform based model to enable mobile cloud applications. In *Proceedings of IEEE Symposium on Computers and Communication*, ISCC, pages 914–919, 2016.

[59] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of ACM International Conference on Management of Data*, SIGMOD, pages 47–57. ACM, 1984.

[60] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[61] Takahiro Hara. Effective replica allocation in ad hoc networks for improving data accessibility. In *Proceedings of IEEE Conference on Computer Communications*, volume 3 of *INFOCOM*, pages 1568–1576, 2001.

[62] Takahiro Hara. Cooperative caching by mobile clients in push-based information systems. In *Proceedings of ACM 11th International Conference on Information and Knowledge management*, CIKM, pages 186–193, 2002.

[63] Yifeng He, Guobin Shen, Yongqiang Xiong, and Ling Guan. Optimal prefetching scheme in p2p vod applications with guided seeks. *Transactions on Multimedia*, 11(1):138–151, 2009.

[64] Zhijun Hou, E Robson, and Azzedine Boukerche. Towards efficient data access in mobile cloud computing using pre-fetching and caching. In *Proceedings of IEEE International Conference on Communications*, ICC, pages 1–6, 2017.

[65] Mitsutaka Itoh, Eric Y Chen, and Tetsuya Kusumoto. Virtual smartphone over ip. *NTT Technical Review*, 8(7), 2010.

[66] Yaser Jararweh, Loai Tawalbeh, Fadi Ababneh, and Fahd Dosari. Resource efficient mobile computing using cloudlet infrastructure. In *Proceedings of IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, MSN, pages 373–377, 2013.

[67] Sofiene Jelassi, Amna Bouzid, and Habib Youssef. Qoe-driven video streaming system over cloud-based vanet. In *Proceedings of International Workshop on Communication Technologies for Vehicles*, pages 84–93. Springer International Publishing, 2015.

[68] Mike Jia, Jiannong Cao, and Weifa Liang. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. *Transactions on Cloud Computing*, 2015.

[69] Mike Jia, Weifa Liang, Zichuan Xu, and Meitian Huang. Cloudlet load balancing in wireless metropolitan area networks. In *Proceedings of IEEE 35th International Conference on Computer Communications*, INFOCOM, pages 1–9, 2016.

[70] Bernard Kamsu-Foguem, Fabien Rigal, and Félix Mauget. Mining association rules for the quality improvement of the production process. *Expert Systems With Applications*, 40(4):1034–1045, 2013.

[71] Won Kim. Cloud computing: Today and tomorrow. *Journal of Object Technology*, 8(1):65–72, 2009.

[72] Emmanouil Koukoumidis, Dimitrios Lymberopoulos, Karin Strauss, Jie Liu, and Doug Burger. Pocket cloudlets. In *Proceedings of ACM 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, pages 171–184, 2011.

[73] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, 2010.

[74] Lingxia Liao, Meikang Qiu, and Victor CM Leung. Software defined mobile cloudlet. *Mobile Networks and Applications*, 20(3):337–347, 2015.

[75] Nianbo Liu, Ming Liu, Guihai Chen, and Jiannong Cao. The sharing at roadside: Vehicular content distribution using parked vehicles. In *Proceedings of IEEE 31st IEEE International Conference on Computer Communications*, INFOCOM, pages 2641–2645, 2012.

[76] Nianbo Liu, Ming Liu, Wei Lou, Guihai Chen, and Jiannong Cao. Pva in vanets: Stopped cars are not silent. In *Proceedings of IEEE 30th International Conference on Computer Communications*, INFOCOM, pages 431–435, 2011.

[77] Gang Lu and Wen Hua Zeng. Cloud computing survey. In *Advances in Measurements and Information Technologies*, volume 530 of *Applied Mechanics and Materials*, pages 650–661. Trans Tech Publications, 5 2014.

[78] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N Papadopoulos, and Yannis Theodoridis. *R-trees: Theory and Applications*. Springer Science & Business Media, 2010.

[79] Bamshad Mobasher, Honghua Dai, Tao Luo, and Miki Nakagawa. Effective personalization based on association rule discovery from web usage data. In *Proceedings of ACM 3rd International Workshop on Web Information and Data Management*, WIDM, pages 9–15, 2001.

[80] Anthony J Nicholson and Brian D Noble. Breadcrumbs: forecasting mobile connectivity. In *Proceedings of ACM 14th International Conference on Mobile Computing and Networking*, MobiCom, pages 46–57, 2008.

[81] Pavan Nuggehalli, Vikram Srinivasan, and Carla-Fabiana Chiasserini. Energy-efficient caching strategies in ad hoc wireless networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, MobiHoc, pages 25–34. ACM, 2003.

[82] Stephan Olariu and Michele C Weigle. *Vehicular networks: from theory to practice*. Crc Press, 2009.

[83] Horacio ABF Oliveira, Eduardo F Nakamura, Antonio AF Loureiro, and Azzedine Boukerche. Error analysis of localization systems for sensor networks. In *Proceedings of GIS'05*, pages 71–78. ACM, 2005.

[84] Carlos Ordonez. Association rule discovery with the train and test approach for heart disease prediction. *IEEE Transactions on Information Technology in Biomedicine*, 10(2):334–343, 2006.

[85] George Pallis, Athena Vakali, and Jaroslav Pokorny. A clustering-based prefetching scheme on a web cache environment. *Computers & Electrical Engineering*, 34(4):309–323, 2008.

[86] Zhengyuan Pang, Lifeng Sun, Zhi Wang, Erfang Tian, and Shiqiang Yang. A survey of cloudlet based mobile computing. In *Proceedings of IEEE International Conference on Cloud Computing and Big Data*, CCBD, pages 268–275, 2015.

[87] Chhabi Rani Panigrahi, Bibudhendu Pati, Mayank Tiwary, and Joy Lal Sarkar. Eeoa: Improving energy efficiency of mobile cloudlets using efficient offloading approach. In *Proceedings of IEEE International Conference on Advanced Networks and Telecommuncations Systems*, ANTS, pages 1–6, 2015.

[88] Richard W Pazzi and Azzedine Boukerche. Propane: A progressive panorama streaming protocol to support interactive 3d virtual environment exploration on graphics-constrained devices. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 11(1):5, 2014.

[89] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys*, 35(4):374–398, 2003.

[90] Cecilia M Procopiuc, Pankaj K Agarwal, and Sariel Har-Peled. Star-tree: An efficient self-adjusting index for moving objects. In *Proceedings of Workshop on Algorithm Engineering and Experimentation*, pages 178–193. Springer Berlin Heidelberg, 2002.

[91] Wang Qing, Hu Zheng, Wang Ming, and Liu Haifeng. Cactse: Cloudlet aided cooperative terminals service environment for mobile proximity content delivery. *China Communications*, 10(6):47–59, 2013.

[92] Lakshmish Ramaswamy and Ling Liu. A new document placement scheme for cooperative caching on the internet. In *Proceedings of IEEE 22nd International Conference on Distributed Computing Systems*, ICDCS, pages 95–103, 2002.

[93] Cristiano G Rezende, Azzedine Boukerche, Heitor S Ramos, and Antonio AF Loureiro. A reactive and scalable unicast solution for video streaming over vanets. *IEEE Transactions on Computers*, 64(3):614–626, 2015.

[94] Hayat Routaib, Elarbi Badidi, Mouna Elmachkour, Essaid Sabir, and Mohammed ElKoutbi. Modeling and evaluating a cloudlet-based architecture for mobile cloud computing. In *Proceedings of IEEE 9th International Conference on Intelligent Systems: Theories and Applications*, SITA, pages 1–7, 2014.

[95] Françoise Sailhan and Valérie Issarny. Cooperative caching in ad hoc networks. In *Proceedings of International Conference on Mobile Data Management*, MDM, pages 13–28. Springer Berlin Heidelberg, 2003.

[96] Simonas Šaltenis. Indexing the positions of continuously moving objects. In *Encyclopedia of GIS*, pages 538–543. Springer, 2008.

[97] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4), 2009.

[98] Usman Shaukat, Ejaz Ahmed, Zahid Anwar, and Feng Xia. Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges. *Journal of Network and Computer Applications*, 62:18–40, 2016.

[99] Huaping Shen, Mohan Kumar, Sajal K Das, and Zhijun Wang. Energy-efficient data caching and prefetching for mobile devices based on utility. *Mobile Networks and Applications*, 10(4):475–486, 2005.

[100] Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.

[101] Hui Song and Guohong Cao. Cache-miss-initiated prefetch in mobile environments. *Computer Communications*, 28(7):741–753, 2005.

[102] Tolga Soyata, Rajani Muraleedharan, Colin Funai, Minseok Kwon, and Wendi Heinzelman. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *Proceedings of IEEE Symposium on Computers and Communications*, ISCC, pages 59–66, 2012.

[103] Tolga Soyata, Rajani Muraleedharan, Jonathan Langdon, Colin Funai, Scott Ames, Minseok Kwon, and Wendi Heinzelman. Combat: mobile-cloud-based compute/communications infrastructure for battlefield applications. In *Proceedings of SPIE 8403, Modeling and Simulation for Defense Systems and Applications VII*, volume 8403, page 84030K. International Society for Optics and Photonics, 2012.

[104] Xiang Sun and Nirwan Ansari. Green cloudlet network: A distributed green mobile cloud network. *IEEE Network*, 31(1):64–70, 2017.

[105] Yufei Tao, Dimitris Papadias, and Jimeng Sun. The tpr*-tree: an optimized spatio-temporal access method for predictive queries. In *Proceedings of the 29th International Conference on Very Large Data Bases*, volume 29 of *VLDB*, pages 790–801. VLDB Endowment, 2003.

[106] Nor Jaidi Tuah, Mohan Kumar, and Svetha Venkatesh. Resource-aware speculative prefetching in wireless networks. *Wireless Networks*, 9(1):61–72, 2003.

[107] Chaitanya Vemulapalli, Sanjay Kumar Madria, and Mark Linderman. Pre-distribution scheme for data sharing in mobile cloud computing. In *Proceedings of ACM 1st International Workshop on Mobile Cloud Computing & Networking*, MobileCloud, pages 11–18, 2013.

[108] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Adaptive application configuration and distribution in mobile cloudlet middleware. In *Proceedings of International Conference on Mobile Wireless Middleware, Operating Systems, and Applications*, MOBILWARE, pages 178–191. Springer Berlin Heidelberg, 2012.

[109] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Cloudlets: Bringing the cloud to the mobile user. In *Proceedings of ACM 3rd Workshop on Mobile Cloud Computing and Services*, MCS, pages 29–36, 2012.

[110] Leandro Villas, Azzedine Boukerche, Regina Borges De Araujo, and Antonio AF Loureiro. Highly dynamic routing protocol for data aggregation in sensor networks. In *Proceedings of ISCC'10*, pages 496–502. IEEE, 2010.

[111] Leandro A Villas, Azzedine Boukerche, Daniel L Guidoni, Horacio ABF De Oliveira, Regina Borges De Araujo, and Antonio AF Loureiro. An energy-aware spatio-temporal correlation mechanism to perform efficient data collection in wireless sensor networks. *Computer Communications*, 36(9):1054–1066, 2013.

[112] Leandro A Villas, Daniel L Guidoni, Regina B Araújo, Azzedine Boukerche, and Antonio AF Loureiro. A scalable and dynamic data aggregation aware routing protocol for wireless sensor networks. In *Proceedings of MSWIM'10*, pages 110–117. ACM, 2010.

[113] Leandro Aparecido Villas, Azzedine Boukerche, Guilherme Maia, Richard Werner Pazzi, and Antonio AF Loureiro. Drive: An efficient and robust data dissemination protocol for highway and urban vehicular ad hoc networks. *Computer Networks*, 75:381–394, 2014.

[114] Leandro Aparecido Villas, Azzedine Boukerche, Heitor Soares Ramos, Horacio AB Fernandes de Oliveira, Regina Borges de Araujo, and Antonio Alfredo Ferreira Loureiro. Drina: A lightweight and reliable routing approach for in-network aggregation in wireless sensor networks. *IEEE Transactions on Computers*, 62(4):676–689, 2013.

[115] Qiufen Xia, Weifa Liang, Zichuan Xu, and Bingbing Zhou. Online algorithms for location-aware task offloading in two-tiered mobile cloud environments. In *Proceedings of IEEE/ACM 7th International Conference on Utility and Cloud Computing*, UCC, pages 109–116, 2014.

[116] YuanJian Xing, Zhi Yang, Chi Chen, and YaFei Dai. Beehive: low-cost content subscription service using cloudlets. *Science China Information Sciences*, 56(7):1–16, 2013.

[117] Jianliang Xu, Qinglong Hu, Dik Lun Lee, and Wang-Chien Lee. Saiu: An efficient cache replacement policy for wireless on-demand broadcasts. In *Proceedings of ACM 9th International Conference on Information and Knowledge Management*, KMO, pages 46–53, 2000.

[118] Tianyin Xu, Weiwei Wang, Baoliu Ye, Wenzhong Li, Sanglu Lu, and Yang Gao. Prediction-based prefetching to support vcr-like operations in gossip-based p2p vod systems. In *Proceedings of IEEE 15th International Conference on Parallel and Distributed Systems*, ICPADS, pages 1–8, 2009.

[119] Zichuan Xu, Weifa Liang, Wenzheng Xu, Mike Jia, and Song Guo. Efficient algorithms for capacitated cloudlet placements. *Transactions on Parallel and Distributed Systems*, 27(10):2866–2880, 2016.

[120] Dinesh Kumar Yadav, Prashant Kumar, and Amarjeet Kaur. Implementation of data center in vehicular cloud computing. *International Journal of Innovations & Advancement in Computer Science*, 6(6):318–322, 2017.

[121] Rong Yu, Yan Zhang, Stein Gjessing, Wenlong Xia, and Kun Yang. Toward cloud-based vehicular networks with efficient resource management. *IEEE Network*, 27(5):48–55, 2013.

[122] Zhenxia Zhang, Richard W Pazzi, and Azzedine Boukerche. A mobility management scheme for wireless mesh networks based on a hybrid routing protocol. *Computer Networks*, 54(4):558–572, 2010.

[123] Zhibin Zhou and Dijiang Huang. Efficient and secure data storage operations for mobile cloud computing. In *Proceedings of the 8th International Conference on Network and Service Management*, CNSM, pages 37–45. International Federation for Information Processing, 2012.