An Android-based Application for Computation Offloading in Mobile Cloud Computing

Mohit Bansal
Department of Computer Science and Engineering
Motilal Nehru National Institute of Technology Allahabad
Prayagraj, India
bansalmohitwss@gmail.com

Aman Jaiswal
Department of Computer Science and Engineering
Motilal Nehru National Institute of Technology Allahabad
Prayagraj, India
srtaj456@gmail.com

Mohd. Asif Ansari
Department of Computer Science and Engineering
Motilal Nehru National Institute of Technology Allahabad
Prayagraj, India
aansari4599@gmail.com

Dr. Dinesh Kumar
Department of Computer Science and Engineering
Motilal Nehru National Institute of Technology, Allahabad
Prayagraj, India
dinesh.kumar@mnnit.ac.in

***Abstract*—Computation Offloading is a fundamental problem in mobile cloud computing in which tasks are offloaded from resource-constrained devices to some computationally rich mobile devices. In this direction, an Android-based application *ClientFramework*, is developed that offloads the client device's tasks. On the other side, an application *OffloadingServer* is developed that provides an interface between Service Provider and Service Receiver and processes the offloaded requests and sends back the results to client devices. Auction mechanism and greedy heuristics are used for resource allocation in each auction round in which tasks are allocated to available mobile devices while considering their computational and storage resources. At last, the conducted experiments verify the offloading benefits in the MCC environment.**

***Keywords*— Mobile Cloud Computing, Computational Offloading, IoT, Auction**

I. INTRODUCTION

Mobile Cloud Computing (MCC) enables the efficient offloading of applications and services from mobile devices to remote resource providers such as cloud, fog, or cloudlet [1]. In MCC, the cloud is shifted close to the mobile users for computation offloading [2]. It is like an ad-hoc cloudlet and different from static and elastic cloudlets. In MCC, there may or may not be a traditional cloud. A cloud may be formed by any ad-hoc entity close to a mobile user, whether it is a server or a combination of mobile devices, or any other infra. It is not static. In MCC, mobile (edge) devices are connected using the cloud directly without any other interleaving hop, i.e., a direct connection between the mobile edge device and cloud servers. For each application, some part of it runs on its device and some part on the cloud. Therefore, there is always a trade-off between offloading and not offloading computation to the cloud. Also, unlike traditional clouds, MCC is not based on SOA. Incorporating general and universal computing has altered the behavior of everyday objects in the presence of humans. Mobile phones, sensors, watches, and other objects are no longer limited to their simple features and upgraded to smart devices thanks to the Internet of Things (IoT) paradigm. Mobile devices will now attach to various Internet of Things (IoT) devices to sense and identify situations and make intelligent decisions [3].

Despite their many advantages, the end-user devices are characterized by limited battery and computation capacity. Consequently, running computation-intensive and delay-sensitive applications, e.g., such as face detection, fingerprint processing, voice processing, video streaming, image processing, healthcare monitoring, etc., is not always profitable. To address these issues, mobile cloud computing appears to be a promising strategy that enables the efficient offloading of applications and services from mobile devices to remote resource providers such as cloud, fog, or cloudlet.

Mobile cloud computing is a computing model that allows mobile devices to offload computationally intensive tasks to cloud computing resources for execution, thus saving battery life and enhancing the performance of mobile cloud applications. The basic concept of MCC is to offload massive and complex tasks to remote resource providers. Mobile Cloud Computing (MCC) is based on cloud computing and mobile computing principles, and it uses wireless networks to provide rich computational services to mobile users. MCC's mission is to make it possible to run rich mobile apps on various mobile devices while providing a rich user experience. Since Mobile phones have become a critical necessity for human life, Mobile cloud computing (MCC) has seen faster growth in research. Mobile clients use mobile apps or embedded browser applications to communicate with cloud service providers.

Mobile Cloud Computing (MCC) is a hybrid of mobile computing and cloud computing that uses offloading techniques to expand the capabilities of mobile devices. Computation offloading addresses the shortcomings of Mobile Devices, such as battery life, computing capacities, and storage space, by offloading the execution and workload to other resource-rich systems with better performance and resources. These systems can be mobile devices, cloudlets, or private/public clouds, having an adequate amount of computational and storage resources [4]. In this work, a client-server offloading mechanism is implemented in which mobile users who want to offload their mobile tasks to remote resources are referred to as Service Receivers who send offloading requests to the server, and mobile users with redundant resources are referred to as Service Providers. The following points illustrate our vital contribution to this work.

1. We develop an Android-based application *ClientFramework*, for placing the offloading request for a computation-intensive task and sharing the redundant

resources for offloading. A user can be a Service Provider or Service Receiver but not both at the same time.

2. We develop a Java application *OffloadingServer* to provide an interface between Service Provider and Service Receiver and offload the task requests to the selected Service Providers.
3. We adopt the greedy approach for allocating the offloading task requests to the Service Providers. The proposed algorithm takes available RAM, CPU frequency, and available time of device into consideration during allocation.

The rest of the paper is organized as follows. The system model is described in Section II. The application's functionality is demonstrated in Section III. Finally, section IV concludes the work.

II. SYSTEM MODEL

This section explains the proposed system's key components and how they perform computation offloading in the MCC environment. The device model, which abstracts the various characteristics of the proposed system, is then presented. After that, the *ClientFramework* is demonstrated in action.

A. Key Components

Mobile devices are the primary participants in the scheme we have proposed. We introduce a prototype of the architecture on the Android platform to verify the functioning of the proposed offloading mechanism on real applications. The system is based on a client-server architecture, with the client layer running in Android apps and the server running in a Java application. Mobile devices have been further categorized into two parts: Service Providers and Service Receivers.

1) Service Provider

A Service Provider is identified as a device having a lot of computing capacity but is not being used right now. As a result, it can make the most of its resources by performing the Service Receiver's computationally intensive tasks. Any device with many computing resources, such as mobile phones, computers, routers, or cloudlets, can serve as a Service Provider. However, for our purposes, mobile phones have been considered Service Providers.

2) Service Receiver

Devices with low computing capacity and battery power are classified as Service Providers. It also necessitates completing specific computationally intensive tasks that would be inefficient or impossible to complete locally on this device. To perform computationally intensive tasks requires additional computation power. For our work, we consider mobile devices as Service Receiver.

3) Offloading Server

Offloading Server provides an interface between Service Provider and Service Receiver. All Service Providers will be connected to the offloading server. And it allocates task requests from Service Receiver to Service Provider.

B. ClientFramework App

It provides an interface for users to act either as Service Provider or Service Receiver. It is an Android application written in the Java programming language. Android 9 is the minimum operating system needed to run this app.

ClientFramework uses a Wi-Fi hotspot in order to connect with Offloading Server. The J2SE Networking APIs are then used to establish a communication link. It also uses Google Vision API to perform Optical Character Recognition (OCR). The Google Cloud Vision API makes it easy for developers to incorporate vision recognition functionality into their apps.

C. System model

In MCC, task offloading is viewed as an offloading environment in which a mobile user can run an application entirely on his or her local mobile device or offload computation-intensive tasks to other computing resources in the network that share their redundant resources. Other mobile devices' local resources can also be used to replace redundant resources [5].

1) A user can send a task offloading request to the OffloadingServer if they want to offload tasks to other devices. The request includes the requested amount of computation and task completion deadline :

$$r_i = (c_i,\ d_i)$$

Where c_i denotes the necessary amount of computation for the offloading task and d_i denotes the offloading task's deadline.

2) Similarly, users who want to sell their excess resources will make a service proposal to the OffloadingServer in the network. The resource requirements are included in the service proposal.

$$s_j = (w_j, t_j, b_j)$$

Where w_j represents the processing speed of the service proposal s_j, t_j is the available time of s_j, and b_j denotes the service offer's bid.

3) Finally, the cost of task execution is calculated in terms of offloading task execution time. Assuming a large number of i task offloading requests and a large number of j devices providing offloading services, the task i execution period on system j's leased resource is :

$$T_{ij} = \frac{c_i}{w_j} + \frac{d_{in} + d_{out}}{v_{ij}}$$

Where d_{in} and d_{out} denote task i's input and output data sizes , respectively, and v_{ij} denotes bandwidth. c_i is the amount of computation needed for task i.

D. The greedy offloading mechanism

This section introduces a greedy auction algorithm for allocating task offloading requests from the Service Receiver to Service Provider's service offers. When allocating task offloading requests, the proposed algorithm takes into account the benefits of offloading in the MCC environment [5].

The proposed offloading mechanism's greedy auction can be defined as a series of auctions at discrete time intervals. Offloading requests r_i and service offers s_i arrive at random times. It sends service details to the offloading server for each s_i, including processing power, bid, and available time. For each request r_i , it contains the requested amount of computation and the user-specified task completion deadline. The greedy auction-based algorithm performs resource

allocation in each auction time interval to allocate computational resources to execute task requests received from the Service Receiver.

In resource allocation, all offloading requests are placed in a queue in the order in which they were sent. The greedy auction algorithm processes the queued requests by fetching one and allocating it to available service offers depending on the task requirement and offloading benefits.

1. To process requests with extensive computation or urgent deadlines, the algorithm sorts the offloading requests in descending order of the value of c_i/d_i.
2. Similarly, service offers are sorted in ascending order based on the value of b_j/w_j, with the service with the lower bid and higher processing speed appearing first.
3. Then the algorithm takes the sorted requests and offers to start the auction process. For each sorted task request, the algorithm iterates through sorted service offers and check for the following conditions:
    a. To ensure the completion of offloading request before its deadline, the completion time of offloading request r_i on service offer s_j does not exceed the completion deadline of request r_i.
    b. To ensure the completion of offloading requests before the service offer's available time ends, the completion time of offloading request r_i on service offer s_j does not exceed the available time of service offer s_j.
    c. To ensure the offloading, the time it takes to complete offloading request r_i locally must be greater than the time it takes to complete it on service offer s_j using the offloading process.
4. After all three conditions are met, service offer s_j will be allocated to offloading request r_i and labeled as busy to other offloading requests. After all of the offloading requests have been processed, the algorithm returns the output.

III. A REAL INSIGHT

In this section, we introduce a prototype of the architecture on the Android platform to verify the functioning of the proposed offloading mechanism on real applications. The system is based on a client-server architecture, with the client layer running in Android apps and the server running in a Java application. In this section, the implementation for the client and the server is explained first, followed by the interaction between various entities of the offloading mechanism. Then some experiments are carried out on the implemented prototype using a mobile OCR application to evaluate the performance of the offloading framework.

A. Implementation of ClientFramework App

The client-side of the framework is implemented on the Android platform. The client framework is divided into three modules depending on various functionalities: a Connection module, a Resource Provider module, and an Offloading Controller module. Fig. 1 shows the starting activity of the ClientFramework App. The connection module handles communication with the offloading server as well as data encryption and decryption. All network operations in the connection module have been performed on a separate thread to provide a better user experience. The Resource Provider module submits the service offer to the offloading server and executes the offloading task requests from other mobile users. The Offloading Controller module deals with offloading tasks to the server and receives results back from the server.

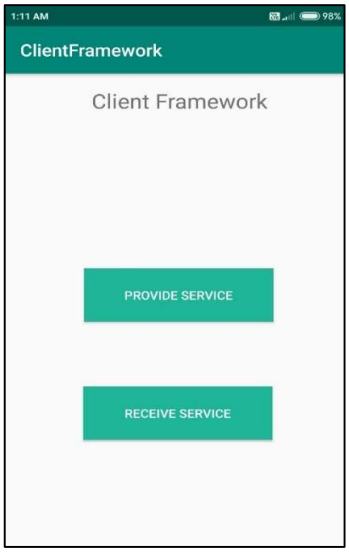


Fig. 1. Starting activity of ClientFramework App

B. Implementation of Offloading Server

The server side of the Offloading framework is implemented as a Java application on the computer. On the server-side, four modules are implemented to meet the needs of various functionalities: a Connection module, a Service Handler module, a Task Handler module, and a Resource Allocator module. The connection module deals with communication with clients. The Service Handler module keeps track of currently available service offers. The Task Handler module keeps track of available offloading task requests. The Resource Allocator module assigns offloading task requests to the available service offers using the greedy auction algorithm explained earlier.

C. Interaction between ClientFramework and OffloadingServer

This section explains the interaction between the client-side of framework (ClientFramework App) and the server-side of the framework (OffloadingServer). Since we have categorized the mobile devices into two categories earlier: Service Provider and Service Receiver. Service Provider is the device that provides or sells the resources to execute offloading task requests. Service Receiver is identified by low computing power; that is why it offloads task requests for execution. In this section, the interaction of the Service Provider with the offloading mechanism is explained first, followed by the interaction of the Service Receiver with offloading mechanism.

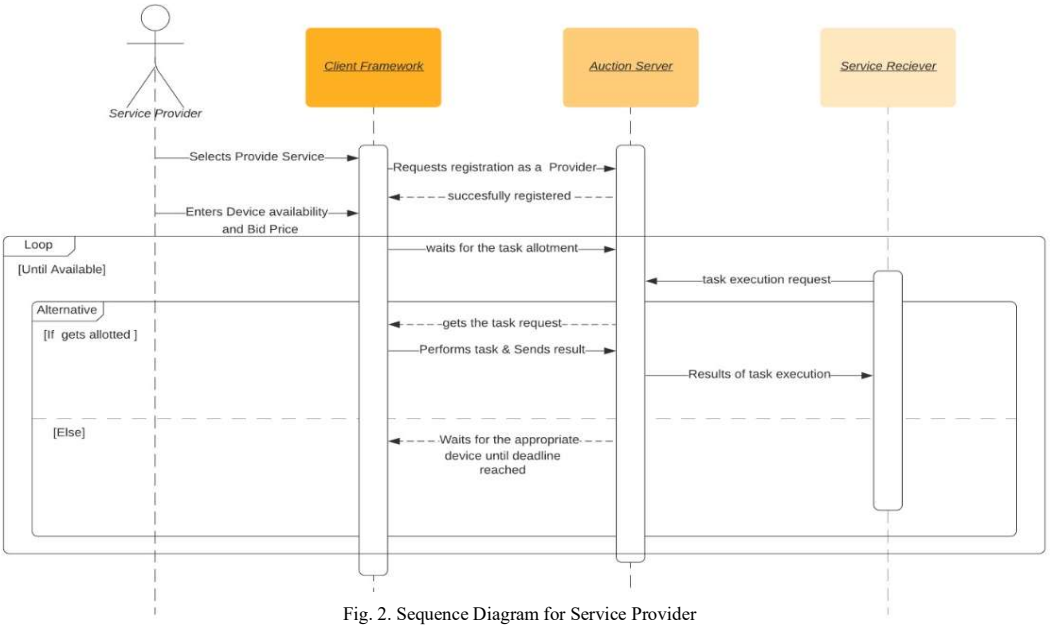


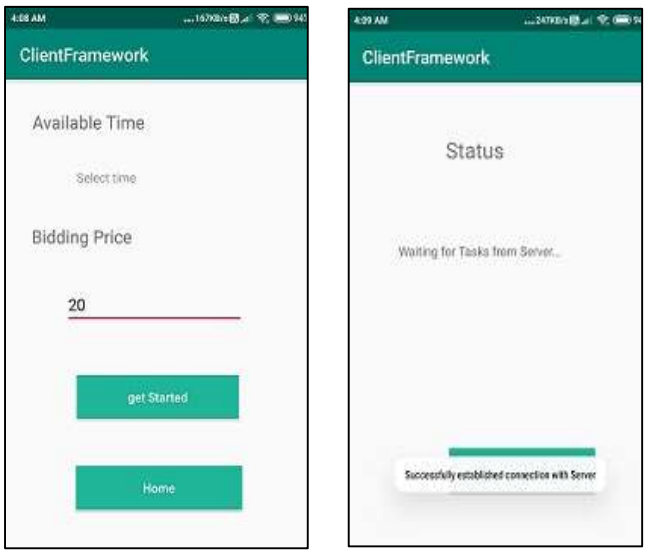
Fig. 2. Sequence Diagram for Service Provider

1) Interaction of Service Provider

Service Provider sends a service offer to the OffloadingServer through the Resource Provider module with its capacity, available time, and bid. Fig. 2 shows the Sequence diagram for the Service Provider. OffloadingServer adds it to Service Handler and sends a confirmation to the Service Provider. Fig. 3 Shows the user interface for the Service Provider. After getting the confirmation, Service Provider waits for the offloading task requests to execute. As soon as it gets assigned to any task request, the Resource Controller module executes it on shared resources. Furthermore, send back the results to the OffloadingServer. This process is repeated until its available time is over.

2) Interaction of Service Receiver

Service Receiver sends a offloading task request to OffloadingServer through Offloading Controller module with the task and its completion deadline. Fig. 5 shows the Sequence diagram for the Service Receiver. OffloadingServer adds it to the Task Handler. After each round of auction, OffloadingServer assigns offloading requests to available service offers. And sends the task to the assigned service offer. Fig.4 shows the user interface for the Service Receiver. After getting the result, OffloadingServer transmits the result to the Service Receiver and deletes the task request from the Task Handler.



(a) (b)

Fig. 3. A user interface for Service Provider

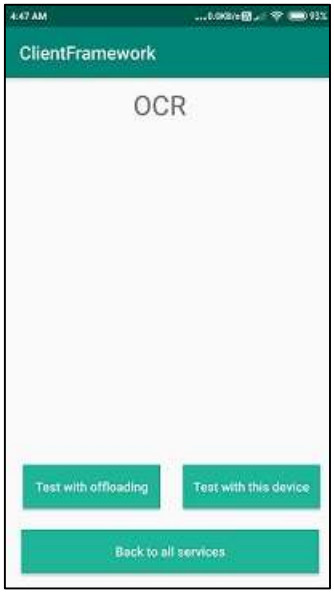


Fig. 4. A user interface for Service Receiver

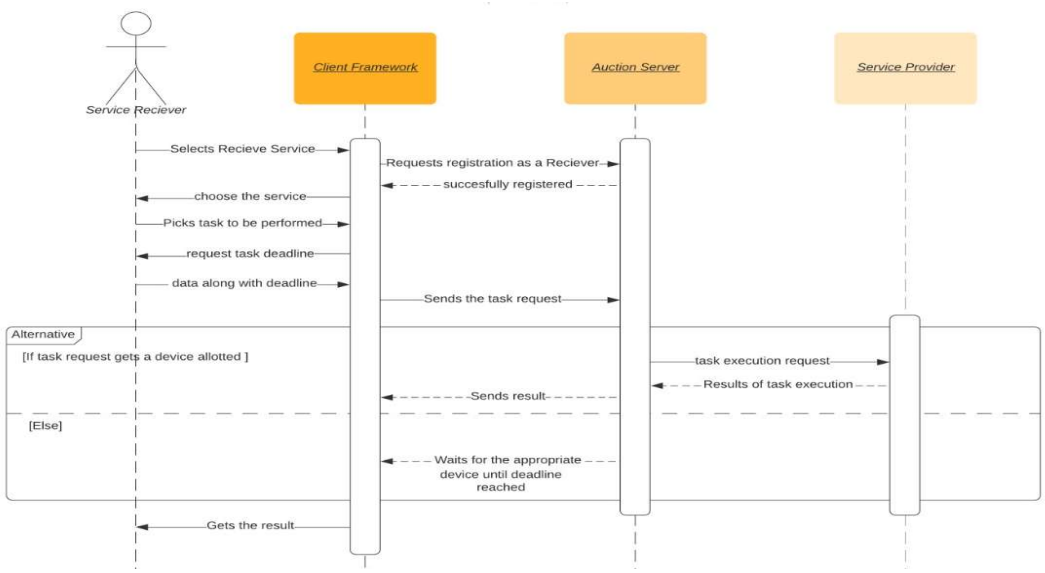


Fig. 5. Sequence Diagram for Service Receiver

D. Performance Analysis

In this section, some experiments will be carried out to test the performance of the proposed offloading mechanism. The application is developed using open-source Android OCR (Optical Character Recognition) google vision to test the framework. Using Google Vision API, it is easy to incorporate the vision features in the application. To provide better functionality of image editing for testing, we have also used the open-source uCrop library. uCrop is a library for Android that allows clip photos for later use.

We have considered three Android devices for the experiment: Vivo 1606, Redmi Note 4, and POCO M2. Vivo 1606 (2 GB RAM, 1.4 GHz) is taken as Service Receiver, Redmi Note 4 (4GB, 2 GHz), and POCO M2 (4GB, 2.32 GHz) are taken as Service Provider. All three devices will be running the android application ClientFramework. The OffloadingServer is running on a laptop. The OffloadingServer listens passively for various requests from Service Provider and Service Receiver using a server socket. Similarly, Service Provider keeps a server socket open to listening for OffloadingServer's offloading task requests.

Fig. 6 shows the interface of the Service Provider receiving the offloading task requests from OffloadingServer, executing the task, and sending back the results to OffloadingServer after successful completion of the task. Fig. 7 shows the interface of the Service Receiver sending tasks to OffloadingServer and getting back the results after successful execution. Since OCR depends upon various factors like image quality, text content, etc. So, to get meaningful and comparable results, we only consider one aspect, such as the quantity of text in the image, which means we vary the number of characters in the image. We experimented on POCO M2 with a set of 6 images with varying quantities of text. Furthermore, local runtime for both the Service Provider and the Service Receiver, as well as offloaded runtime, are measured. The Offloaded runtime of the task is calculated as the sum of local execution time for Service Provider, offloading time of the task, and auction processing time.

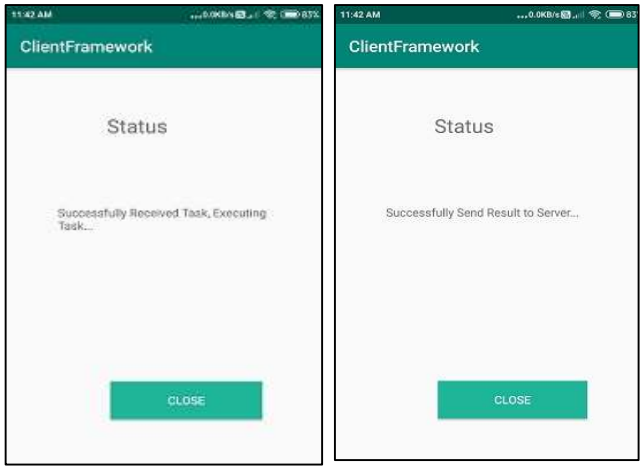


Fig 6. Service Provider receiving, executing the task, and sending back result

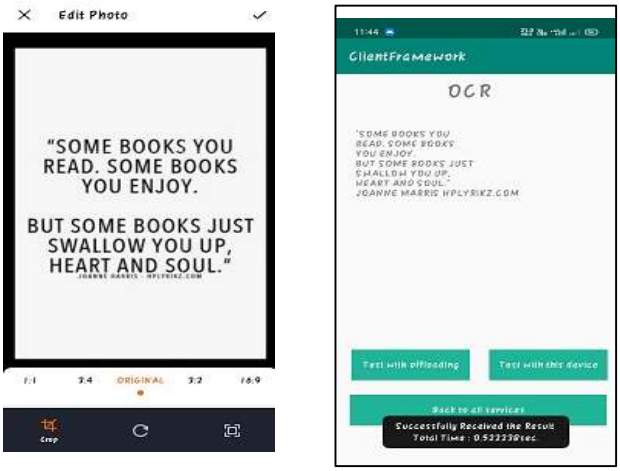


Fig 7. Service Receiver sending the task and receiving the result.

Input	Local Execution Receiver (sec.)	Local Execution Provider (sec.)	Offloaded Runtime (sec.)
Input 1	1.2405	0.0995	1.5935
Input 2	1.4510	0.1081	1.8145
Input 3	2.4101	0.1414	2.3165
Input 4	3.9017	0.2221	2.6517
Input 5	4.9582	0.2881	2.9528
Input 6	8.4118	0.5381	3.2748

Fig. 8. Task execution results

Fig. 8 shows the results of the experiment we have carried out. It shows the local execution time of the Service Receiver, the local execution time of the Service Receiver, and the total offloaded runtime. Fig. 9 shows the comparison between the local runtime of Service Receiver and offloaded runtime for image OCR. The blue bar represents the Service Receiver's local runtime, and the red bar represents the Offloaded runtime when using offloading service. As can be observed from the figure, when the quantity of text in an image is too low, the offloaded runtime is greater than the local runtime. In this case, the task offloading time becomes the major factor here. However, as we increase the text content, the execution time of the task plays the leading role after a particular stage. It shows that the proposed offloading mechanism will improve task execution time by a considerable amount.

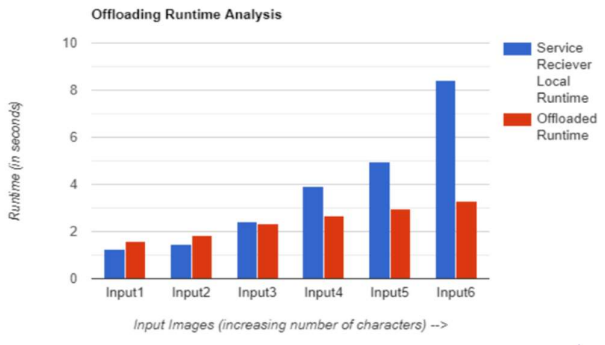


Fig. 9. Comparison of task execution between local and offloading services.

IV. Conclusions

In this paper, we present an MCC offloading mechanism that allows a device with limited computing power or battery life to offload its task request to a device with more processing power. Such mobile devices can be Service Provider or Service Receiver. Service Providers are characterized by their high processing capacity, while Service Receivers are distinguished by their low computing power or battery life. The greedy auction algorithm is then utilized, allocating Service Receiver task requests to Service Provider service offers. The greedy auction algorithm processes the offloading requests and allocates them to available service offers depending on the task requirement and offloading benefits.

Then we have presented the prototype of the proposed mechanism in which the client-side of the framework is implemented on the Android platform, and the server-side of the framework is implemented on the regular personal computer. We then divided the whole application into different modules according to their functionalities and presented the interaction between client and server modules. To assess the feasibility of the proposed mechanism, we conducted some experiments using OCR (Optical Character Recognition) as a task. The ClientFramework and OffloadingServer are available to download at https://github.com/bansalmohitwss/Mobile-cloud-offloading-framework.

References

[1] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, R. Buyya, Mobile code offloading: From concept to practice and beyond, IEEE Commun. Mag. (2015). https://doi.org/10.1109/MCOM.2015.7060486.

[2] B. Zhou, A.V. Dastjerdi, R.N. Calheiros, S.N. Srirama, R. Buyya, MCloud: A Context-Aware Offloading Framework for Heterogeneous Mobile Cloud, IEEE Trans. Serv. Comput. 10 (2017) 797–810. https://doi.org/10.1109/TSC.2015.2511002.

[3] A. Kumar, R. Yadav, G. Baranwal, NearBy-Offload: An Android based Application for Computation Offloading, in: 2021: pp. 357–362. https://doi.org/10.1109/iciis51140.2020.9342724.

[4] G. Orsini, D. Bade, W. Lamersdorf, Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading, in: Proc. - 2015 8th IFIP Wirel. Mob. Netw. Conf. WMNC 2015, 2016: pp. 112–119. https://doi.org/10.1109/WMNC.2015.10.

[5] B. Zhou, S.N. Srirama, R. Buyya, An auction-based incentive mechanism for heterogeneous mobile clouds, J. Syst. Softw. 152 (2019) 151–164. https://doi.org/10.1016/j.jss.2019.03.003.