Augmenting The Performance Of Mobile Devices Through The Use of Dynamic Partition Offloading With Heterogeneous Mobile Clouds

MSc Research Project
Cloud Computing

Emmanuel Okechukwu Weje
Student ID: 19122411

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

Student Name:	Emmanuel Okechukwu Weje
Student ID:	19122411
Programme:	Cloud Computing
Year:	2019
Module:	MSc Research Project
Supervisor:	Manuel Tova-Izquierdo
Submission Due Date:	17/08/2020
Project Title:	Augmenting The Performance Of Mobile Devices Through The Use of Dynamic Partition Offloading With Heterogeneous Mobile Clouds
Word Count:	7344
Page Count:	20

Signature:	
Date:	27th September 2020

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Augmenting The Performance Of Mobile Devices Through The Use of Dynamic Partition Offloading With Heterogeneous Mobile Clouds

Emmanuel Okechukwu Weje
19122411

Abstract

Smart mobile devices (SMD) are a part of our daily lives and sometimes we cannot do without them. With this, they are expected to have high functionalities and resources which is sometimes not the case i.e they are resource constrained. The Mobile Cloud Computing (MCC) paradigm was introduced to solve this issue and previous works have been implemented but are not without their flaws. Most works do not consider the context of the SMD when making offloading decisions and utilize only one offloading resource option which is not particularly ideal and limits the usage of the paradigm. In this paper, we present a context aware heterogeneous approach to the MCC paradigm. We apply heterogeneity by using serverless computing functions and remote mobile clients as our offloading options. We make offloading decisions based on the SMDs context (battery, network & memory) during runtime by applying dynamic partitioning. The core of our system is the decision making engine and remote configurations. Remote configurations have been implemented to change parameters in our approach without having to modify source code. This provides flexibility. Results gotten from experiments carried out showed our system was able to make offloading decisions based on changing context and also up to 66% reduced execution time (Zhou et al. produced around 65%) for offloaded tasks as compared to local executions thus reducing power utilization. Our solution also proved to be cost efficient when utilizing serverless computing as opposed to using cloud VMs as an offload resource.

1 Introduction

These days, the smart mobile device (SMD) is almost a necessity for every individual. With the vast growing trend in technology, the SMD has positioned itself as a major player in improving the life of people. Various applications are being developed to run on these devices with the aim of providing some form of value to the mobile device user. With this, SMDs are expected to provide an almost perfect platform for mobile application developers to capitalize on and produce applications that will be beneficial to users but this is not the case at all times. SMDs have limited resources such as battery life, CPU power, memory, network etc. This puts them in the category of resource constrained devices and as such limits the range of applications that should be able to run on them. Sometimes, manufacturers of SMDs try to take this into consideration by manufacturing high end devices that will often suit the needs of users. This is a good approach but often times does not benefit the SMD user mostly basing on cost. Not all users would be able to afford high end devices thereby maintaining the issue of constrained resources. In a bid to solve this issue, the Mobile Cloud Computing (MCC) paradigm was introduced.

According to Buyya et al. (2018), the MCC is a paradigm that aims to augment resource constrained mobile devices by bringing the vast amount of cloud resources to the SMD. There are many Cloud Service Providers (CSP) such as Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, IBM Cloud etc. These CSPs make available resources such as compute power, storage, network, proprietary solutions etc that are offered as services and are readily made available on the cloud. With the Pay-As-You-Use model of the cloud, cloud resources can be rapidly provisioned and integrated to be consumed by applications running on SMDs to make them more efficient. With the vast amount of positives the MCC introduces, it is not without its flaws. The MCC also offers challenges that can potentially hinder a wider adoption of the paradigm. According to Noor et al. (2018); Akherfi et al. (2018), some of these issues can be identified as singular offloading choice, high costs, inadequate task profiling, power utilization, low flexibility. These are issues that have to be considered before taking up the MCC and they also have to be tackled.

In research papers by Buyya et al. (2018); Zhou and Buyya (2018), the MCC has been identified to be made up of two models which are code offloading and task delegation. In task delegation, the SMD uses cloud services from different CSPs and introduces a potential interoperability issue (Buyya et al.; 2018) while code offloading has to do with profiling the mobile application to identify task(s) that can be deemed resource intensive and then be offloaded to external platforms with available resources. For this research study, we aim to apply the code offloading model to augment SMDs.

In mobile code offloading, which can also be termed as computational offloading (the two will be used interchangeably in this report), code is generally profiled and partitioned with set objectives in mind. These objectives could include but not limited to reduction in power utilization, increase in performance and scalability. Resource intensive tasks are identified and a decision making system is employed to determine if a task should be offloaded to a resource rich service or not (Zhou and Buyya; 2018). As discussed by Flores et al. (2018), there is still an issue of correctly profiling and partitioning compute intensive task(s) to offload. It is important to tackle this issue as wrong profiling and partitioning will be detrimental to the SMD user. Partitioning in offloading can either be static or dynamic as stated by Gu et al. (2018) with the difference being that dynamic partitioning occurs during application runtime.

Adopting a heterogeneous approach to the MCC is a potential solution to some issues identified and it is a field that has not been really taken into account by previous research works as they regularly make use of only one offload resource (Zhou et al.; 2017) and rarely consider the context of the SMD before offloading. Some of these previous works include frameworks such as mRARSA (Islam et al.; 2020), Mildip (Lu et al.; 2020), MobiCop (Benedetto et al.; 2018) etc. The heterogeneous mobile cloud (HMC) is one that can be made up of public clouds, mobile cloudlets and mobile ad-hoc networks (MANET). Research papers by Alonso-Monsalve et al. (2018) and Zhou et al. (2017) have presented the HMC but did not seem to tackle all issues presented by the MCC. Alonso-Monsalve et al. (2018) does not properly define how to adequately profile intensive tasks for offloading and while Zhou et al. (2017) does this, the HMC presented by authors is met with a power utilization problem during a device discovery service implemented and also cost incurred utilizing cloud VMs as offloading resources.

These issues and challenges identified have therefore led us to our research question and research objectives. Our research question is given below:
RQ: Can dynamic partitioning of a mobile application running compute intensive task(s) be made to function more efficiently by applying the code offloading model in a context aware heterogeneous mobile cloud environment while also reducing costs that might be incurred on users?

In a bid to answer the given research question, our research work has the following objectives:

- Perform research on the current state of the art in mobile code offloading.
- Design and implement a code offloading approach that will tackle issues identified and thus answering the research question.
- Evaluate the derived code offloading approach using appropriate test measures in order to determine performance.

Our main contributions are:

- A HMC system consisting of local code execution, public cloud resources and remote mobile client resources.
- Design and implement a context aware offloading resource decision making algorithm using dynamic partitioning.
- Apply the serverless computing approach in order to reduce costs that might be incurred from utilizing cloud VMs.
- Integrate remote configuration to enable flexibility during offloading decision making

The main report is divided in the following sections. Section 2 describes related works, Section 3 introduces the methodology we employ, Section 4 gives the design specification, Section 5 describes the implementation, Section 6 shows our evaluation and lastly in Section 7 we bring the research to a conclusion and define any future work.

2 Related Work

In this section, we discuss related works as regards to the research topic. Many works have been presented as regards to code offloading in the MCC paradigm. This section is structured as follows. First, we review papers on context aware offloading, heterogeneous mobile cloud computing, existing code offloading frameworks and lastly, we compare frameworks.

2.1 Context Aware Offloading

The context of mobile device is beneficial during a code offloading decision making process. This context can be in terms of its current battery life, available network, location, memory etc. Correctly applying these factors would lead to an efficient offloading process as the context of a mobile device tends to change continuously.

Basing on this, Chen et al. (2017); Islam et al. (2020) proposed context aware computational offloading frameworks that apply a dynamic approach in selecting the most suitable remote resource to offload tasks to. The framework by Chen et al. (2017) comes with an estimation model that chooses what cloud resource to offload a resource intensive task to. The idea behind this is that the available cloud resources offer varying performance and network quality. The estimation model works with tasks meant to be offloaded, the network quality and performance power of cloud resources. Islam et al. (2020) mainly focused on the network context (e.g signal strength and range) to make offload decisions. Both frameworks make their decision making during the application runtime through dynamic partitioning. This process ensures that the current context of the device is used in the decision making and therefore an adequate resource is selected. Result from Chen et al. (2017) showed a reduced execution time of about 6% to 96% and power utilization was down to about 60% to 90% while results Islam et al. (2020) demonstrated appropriate offloading decisions while also offering better performance and power utilization. Even with these good results, the frameworks are not robust enough to different offloading options. This is what we aim to achieve with our HMC approach.

Also, a research work by Xu et al. (2020) studied the inclusion of deep learning in wearable mobile devices. The motivation behind the study is that wearable devices take in lots of information and require processing in order to gain meaningful insights. Similar to Chen et al. (2017) and Islam et al. (2020), a context aware offloading scheduling process was implemented. Device context that was considered include latency, network signal strength, battery life etc. The study focused on offloading compute tasks from the wearable to a handheld device. One issue with this approach is that handheld devices are as well restricted in resources and therefore issues might arise if a task needs to be offloaded and the handheld device does not have enough resources to allocate. An adequate approach to this would have been to implement alternative offloading options such as a remote cloud or other nearby devices.

Understanding the current context of device is beneficial to offloading decision making algorithms. This helps to ensure that an adequate offloading resource is being selected because as stated before, the context of a mobile device changes continuously. One major disadvantage noticed in the papers above is that they only considered offloading to a remote cloud or a nearby device and not creating an option for both. Having multiple offloading options would have made the proposed frameworks presented more robust and efficient.

2.2 Heterogeneous Mobile Cloud Computing

As discussed, adopting the HMC can be a potential solution to some of the problems or challenges identified in the MCC. The HMC often involves having multiple offloading options in order to make better offloading decisions. This makes offloading decisions more robust and efficient when applied with the context of the device. In a paper by Alonso-Monsalve et al. (2018), a proposed model for the HMC in hybrid clouds was introduced. The model involved volunteer cloud system where people can offer resources of their mobile devices to act as part of the cloud resources that would be utilized. The BOINC [1] open-source software was used to make both the volunteer devices and the cloud resources interact with each other. Zhou et al. (2017) also proposed a HMC framework which uses a multi-criteria decision making (MCDM) algorithm and cost estimation modelling to

[1]BOINC: `https://boinc.berkeley.edu/`

determine a suitable resource to offload tasks to. The framework used VM migration and Java Reflection to offload tasks. One issue with VM migration is that it might introduce latency overheads to the offloading process. The MCDM in their approach is based on the Technique for Order of Preference by Similarity (TOPSIS). TOPSIS is chosen here because it is assumed to be lightweight and can be modified to accept other criteria if the need arises.

Both frameworks above presented good results when implemented. The Alonso-Monsalve et al. (2018) framework offered advantages in low costs, load distribution, better performance and scalability. One main disadvantage of the framework is that it is mainly dependent on its volunteer device model. If there are no volunteer devices, the framework might not be able to function optimally and falls back to using provisioned cloud VMs. Also, they did not consider device context during offloading. System evaluation from Zhou et al. (2017) also presented good results. The MCDM implemented was able to provide adequate medium at which compute tasks will be offloaded. The system also implemented a discovery process to look for other close by devices to make offloading decisions. This discovery seemed to provide some power utilization overhead which will not be good for devices.

Lee and Lee (2018) presented a HMC system that mainly focused on the chances of having outages during task offloading. The HMC comprised of external resources such as public clouds and cloudlets. The system works by initially defining task offloading outage possibilities with just a remote cloud then the same is done with the cloudlets. When both definitions are done, they are then used to arrive at a general outage probability involving both offload resources. Secondly, the system involved research on the right choice on deployment of cloudlets. This research was focused on maximising profit for CSPs. The downside to this is that maximizing profits for the CSPs is synonymous to incurring more costs on users. The approach we apply aims to reduce costs for users through the application of serverless computing as the cloud offload choice.

Zhou et al. (2019) presented a paper that provides a model to incentivize users that offer their mobile device resources as part of the HMC. Issue of providing a mechanism to incentivize users has before been raised by Buyya et al. (2018). In that paper, it was said that this mechanism would lead to more users offering their unused resources in their mobile devices. This is the issue Zhou et al. (2019) aimed to solve. A market place was designed for users to sell their unused resources for offloading tasks in a HMC scenario. The process was more of an auction strategy and users get to bid for resources at a price with the winning user getting to pay the bid price. A penalty system was also implemented where a user whose mobile device wins an offloading bid and cannot complete the offloading request for whatever reason is made to pay any extra costs incurred from pushing the failed offload request to a CSP which acts as a backup for failed requests. In as much as this is good, there are possibilities that offload requests might fail and is caused by issues beyond the mobile device owner's control. In such as situation, it is unfair to make such a user incur unwarranted cost for the failed offload request and this might discourage users from participating. Overall, the system displayed performance efficiency and a clear bidding process. Our approach does not currently involve incentivizing users but can be considered for a future work as it is a feature that would drive acceptance for the model.

In summary, the HMC should be a consideration when adopting the MCC. It delivers more offloading options for tasks. In addition with the mobile device context and an appropriate decision making system, the right choice of offloading will be made. This is what we aim to achieve while taking into consideration issues identified in papers above.

2.3 Code Offloading Frameworks

In this section, we review various existing offloading frameworks. Over the years, various studies and research works have introduced different frameworks to achieve code offloading in SMDs. These frameworks usually come with different methodologies and approaches which they use to achieve offloading.

In Chen et al. (2019), authors introduced an offloading solution called AndroidOff which focused on cost estimation for the process of offloading tasks. This was done through a prediction process which worked by applying test cases to some methods in the application to determine how long it will take them to run. After this was done, a weighted call graph for other methods was then created and their execution time predicted basing on the previous test cases ran. Results gotten showed execution time saved up to 8% - 49% and lower power utilization of up to 12% - 49%. One limitation of this approach is that the test cases used might not in all cases produce results suitable in a real world situation. In Mahmoodi et al. (2019), authors proposed a joint scheduling approach for offloading components of a mobile application and also scheduling the order in which the offloading should be performed. Their approach depended on network availability and used parallel execution to run tasks both locally and on the provisioned AWS EC2 instances. It was mainly targeted at mobile applications that had sequential component dependency graphs. This is a limitation in their approach as it reduces the range of applications which it can be applied to. Overall, significant results were achieved with reduced power utilization for applications with lengthy runtimes.

In Benedetto et al. (2018), authors proposed a framework called MobiCOP which mainly focused on reducing the execution time of resource intensive methods that run for a long time as opposed to methods with short execution time. The framework comprised of an offloading decision engine, public cloud execution environment and a communication handler. The decision engine focused on quality of service monitoring and profiling of code which it used to make decisions on offloading tasks or not while the communication handler focused reducing power usage during poor network scenarios. Results from the approach showed that for longer running tasks there was about 17 times increased efficiency and about 25 times better power saving. The framework integrated AWS EC2 which might pose higher costs when compared to using serverless computing.

An environment friendly cloudlet based MCC approach was proposed by Gai et al. (2016) which focused on saving energy and reducing latency when utilizing wireless communications in unstable network conditions. The approach used dynamic partitioning to make decisions based on changing context or environment (Dynamic partitioning is a key part of our approach). Offloading was achieved by first sending requests to nearby cloudlets which then search for a suitable cloud resource to offload the task to. This process is limiting because it will not be suitable for applications that are latency sensitive and will therefore have less applications. However, this is due to the focus on only energy savings which produced expected results from experiments. More recently, Lu et al. (2020) also proposed an energy efficient framework for cloudlet based offloading in microprocessors and mobile process architecture called Mildip. The framework functioned by taking measurements of power utilization of CPU cores and its frequencies, and also the required performance and power utilization for mobile applications which was then used in performing analysis for energy efficiency. Simulations were used to carry out experiments which showed that the approach offered about 77% of lower power utilization for offloading tasks as compared to running the task locally. One limitation of this approach is that the authors did not consider device context when making offloading decisions.

2.4 Comparison of Code Offloading Frameworks

In this section, we present comparisons between various code offloading frameworks and approaches and this is displayed in Table 1 below.

Table 1: Comparison of code offloading frameworks.

Framework	**Goal**	**Offloading Mechanism**	**Partitioning**	**Context Aware**	**Heterogeneous**
mCloud (Zhou et al.; 2017)	Energy & Performance	VM migration	Dynamic	Yes	Yes
Circa (Lin et al.; 2015)	Performance	Code	Static	No	No
MilDip (Lu et al.; 2020)	Energy	Code	Dynamic	No	No
Nawrocki et al. (2019)	Energy & Performance	Code	Dynamic	No	No
Gai et al. (2016)	Energy	Code	Dynamic	Yes	No

In conclusion of the related works section, basing on reviewed papers, we can see that there are still identifiable issues that need to be solved. Most frameworks produce good results but are limited in their approach and therefore can lead to minimal application. In this research paper, we aim to apply a context aware heterogeneous approach to our solution which will reduce offloading costs with serverless functions and remote mobile clients as offloading options while also adding flexibility with remote configurations.

3 Methodology

In this section, we discuss the steps we aim to apply to achieve our research objectives. As stated earlier, this research work aims to apply a context aware heterogeneous approach to the MCC paradigm through the use of dynamic partitioning. We aim to use serverless computing functions and remote mobile clients as a our offloading resource options. Both options are chosen to improve on the drawbacks found in other mobile code offloading frameworks in our related works section. We also apply realtime monitoring events to know when task(s) have been completed by the external resources so the mobile application can be updated accordingly. In order to decide whether a resource intensive task should be offloaded or not, we use an offload decision engine which takes the current device context to make offloading decisions.

3.1 Dynamic Partitioning

Application partitioning can either be static or dynamic. In static partitioning, the decision to offload a task is made during development stage while in dynamic, the decision is made during runtime. The dynamic approach has an advantage over static due to the continuous change in the context of the SMD. This is why we apply the dynamic approach and also it has been used by other related works such as Zhou et al. (2017). In order to achieve dynamic partitioning, we use the Java Reflection API [2]. Java Reflection can be used to monitor classes, methods etc during runtime. With the application of this, we can call our offloading decision engine to decide whether to offload a task or not. Before we can offload tasks, we have to properly identify resource intensive tasks so correct offloading decisions can be made. This is done during development stage through method annotations. Resource intensive methods are annotated with a *@PushRemote* annotation. Only methods with this annotation are considered for offloading. If no annotations are found during runtime using Java Reflection, all tasks are performed locally on the SMD.

3.2 Offloading Decision Engine

The offloading decision engine is called during runtime to determine whether to offload a task or not. It uses the SMD's context during runtime to make this decision. The engine has the following components NetworkProfiler, MemoryProfiler and BatteryProfiler. The NetworkProfiler monitors the SMD for available network such as mobile networks (2G/3G/4G) and WiFi. If a network is found, the NetworkProfiler also checks the signal strength to determine if it is suitable for offloading task(s). The MemoryProfiler is used to check available RAM on the SMD. This component is needed to know if the SMD has enough memory to perform a task or not. Lastly, the BatteryProfiler monitors the SMD's battery level and also if the SMD is in a charging state. These three components make up the engine and are the core of the offloading decision making process.

3.3 Offloading Resources

Heterogeneity in the MCC involves having multiple offloading options. Here, we use serverless computing and remote mobile clients to execute offloaded tasks. We apply serverless computing here to reduce costs that are attributed with using a public cloud VM. This is one of the objectives of this research. Serverless computing involves writing functions that perform specific tasks. In serverless computing, we do not have to bother about server setup or maintenance. These are taken care of by the CSP and costing is mainly done by amount of resources consumed through function calls. The second offload resource option used are remote mobile clients. These mobile clients are other SMDs that want to offer their idle resources to perform offloaded tasks. This is done by installing an offloading client application on the SMD client. The client application at intervals monitors the client SMD's context to determine if it is suitable to perform an offloaded task or not. If the client SMD has adequate resources, it can be chosen to perform offloaded tasks. An approach like this usually involves incentivizing the owners of the participating client devices but this is not being studied in this research and is considered for future research work.

In order to get best results from the offloading options, we apply method replication. This involves replicating profiled intensive methods on the different offloading resources. This is used to ensure consistent results are gotten when compute tasks are offloaded. One method that has been used by other related works is VM migration. The VM migration involves transferring a VM with the computational task to another external resource to

[2] Java Reflection API: `https://docs.oracle.com/javase/tutorial/reflect/index.html`

execute. As good as this approach is, it might be heavy on the users network and can also introduce latency overheads before tasks can be executed externally, the VM has to be migrated first and set up. With the method replication approach, functions already exists with needed code and only need to be called. One possible issue that might be associated with the method replication is code maintenance. Any code change must be implemented across all resources.

Also, the default offload resource option in our approach is the serverless computing. This is chosen because maintenance is being done by the CSPs and they are highly scalable. This default option can be changed at anytime using remote configuration. Remote configuration can be used to change variables in a mobile application without the need to modify source code. This gives the application owner complete control over what offload resource is chosen by their application thereby introducing flexibility.

3.4 Realtime Monitoring

In order for our approach to function optimally, we apply realtime monitoring to know if offloaded task(s) have been completed by external resources to update the requesting mobile application with results. This is important as some applications are latency sensitive and require results as soon as they are produced. In our approach, we do this by sending results produced by remote resources to Firebase Cloud Firestore [3] which is a NoSQL database that can be listened to in realtime. We make this work by creating a unique document on Firestore for each task offloaded. The SMD which made the offloading request then listens to this document for updates. As soon an update is received, an update event is sent to the SMD which can then update itself with the appropriate result.

3.5 Case Studies

For this research paper, we aim to apply two research case studies. We use an Optical Character Recognition (OCR) application and an N-Queens application both built on the android platform using Java. OCR involves reading and generating text from documents such as images while N-Queens is a chess style of application but uses only the queens. N-Queens involves placing N number of queens on a chessboard in such a way that the available queens will not be able to take out each other. These applications are used to determine performance of our approach and will be built on the Android platform using Android Studio. Performance metrics being monitored and compared against local execution are battery utilization, task completion time and performance. In order to capture these metrics, an android profiler monitor is employed which captures resource usage by the case applications.

4 Design Specification

Our overall system architecture is illustrated in figure 1. It shows a general flow of how task offloading is performed. Our approach uses serverless computing (GCP Cloud Functions) and remote mobile clients as code offloading resources. Firstly, we have our SMD which wants to offload a compute task. The decision to offload a task or perform

[3] Firebase: `https://firebase.google.com/docs/firestore`

a local execution is done by our offloading decision engine. The engine uses the SMD's current context to make this decision. Tasks can either be offloaded to the public cloud through either cellular networks (2G/3G/4G) or WiFi network depending on which is currently available. Lastly, depending on the default offloading option, the offloaded task can either be performed on the cloud functions or available mobile clients. As stated before, the default offloading option can be changed at anytime with a remote configuration implemented and does not require source code modification.

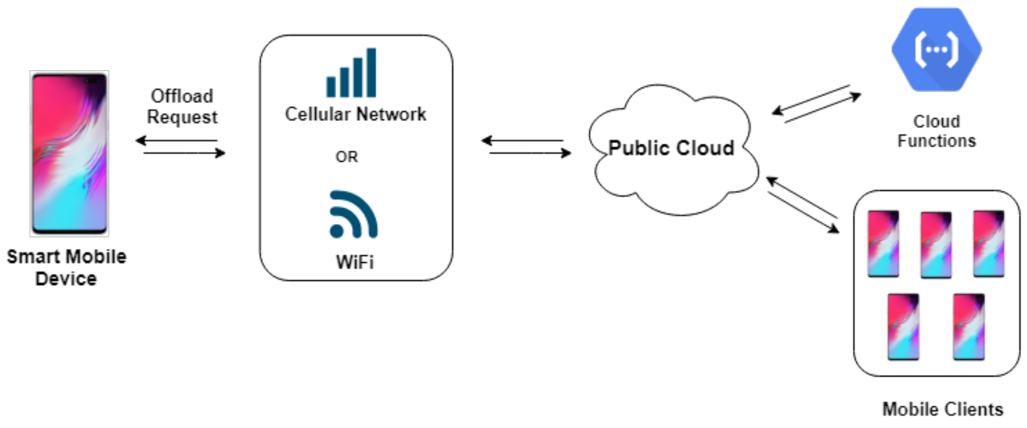

Figure 1: Overview of System Architecture

4.1 System Components

Figure 2 illustrates the various components of our system. Our system can be broken down into two major components which are the SMD and remote task execution. Offloading requests are made by the mobile application on the SMD. After this request has been made, we use Java Reflection during runtime to get the SMD's current context and make our offloading decision. The device context in our approach is currently made up of a BatteryProfiler, NetworkProfiler and MemoryProfiler. We used these components to determine if the SMD has enough resources to perform a compute intensive task or not. This helps the decision engine to know if the task execution will happen locally or be offloaded to available cloud resources.

4.2 Context-Aware Offloading Decision Engine

For our algorithm illustrated in Algorithm 1, the default offload choice is local execution. This is chosen because there is a possibility that the SMD might not have network access and will not be able to offload tasks to remote cloud resources. This default is also maintained if the SMD has adequate resources to run the computational task. We use the getRemoteConfig() method to get default context values from remote configuration. If the SMD does not have adequate resources, then a check is done with the NetworkProfiler to determine if network is available and also the strength of the network. Depending on the strength of the available network, the OFFLOAD_CHOICE can either be CLOUD OR LOCAL execution.

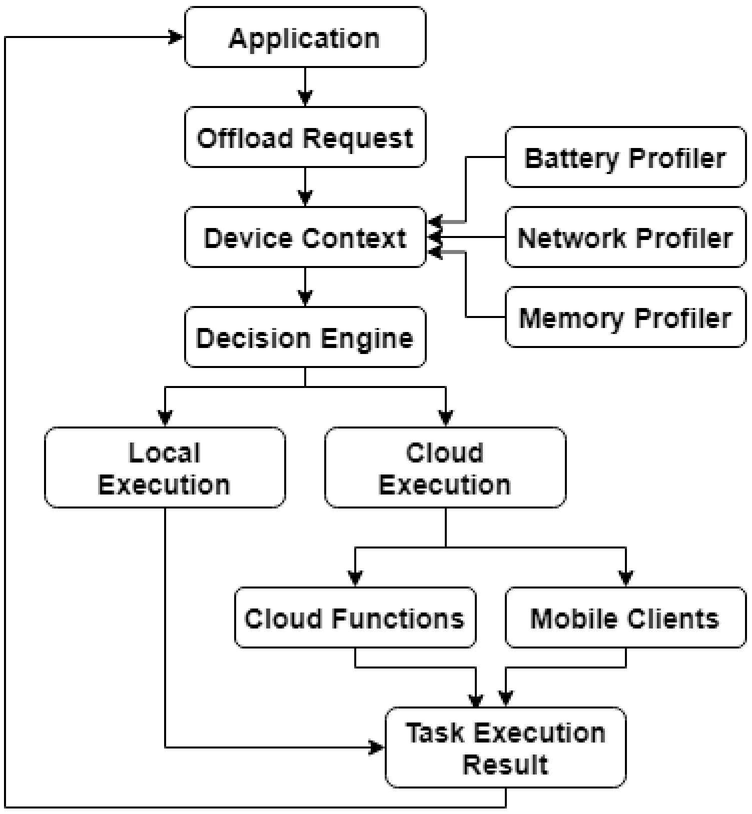

Figure 2: Overview of System Components

5 Implementation

In this section, we discuss the implementation of our approach. The system is entirely written using the Java SE 8 [4]. The mobile applications used for our case studies are built for the Android platform using Java and Android Studio IDE. For our offloading resources, we use GCP as our CSP for our serverless computing. The service used in GCP is called Google Cloud Functions. For the remote mobile clients, we built a mobile application also based on the Android platform which users (volunteers) willing to participate will install on their devices. Both resources have replicated methods of tasks that can be offloaded.

5.1 Method Annotation

In our system, we start by first identifying resource intensive methods that can potentially be offloaded during development stage of the mobile application and annotate them using Java annotations. This is done so that during runtime, using Java Reflection we can know what method we plan to offload. We have created a custom Java annotation class called PushRemote with retention policy of runtime. This is done using @Retention(RetentionPolicy.RUNTIME). This will ensure that any method annotated with the @PushRemote annotation will remain accessible during runtime. The annotation use is also restricted to only methods as we only plan to offload only methods. This is achieved using @Target(ElementType.METHOD).

[4]Java SE 8: `https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html`

Algorithm 1 Context-Aware Offloading Decision Algorithm

```
 1: function makeOffloadDecision($context$)
 2:     $OFFLOAD\_CHOICE \leftarrow LOCAL$
 3:     $batteryProfiler \leftarrow new\ BatterProfiler(context)$
 4:     $memoryProfiler \leftarrow new\ MemoryProfiler(context)$
 5:     $networkProfiler \leftarrow new\ MemoryProfiler(context)$
 6:     if ($batteryProfiler.getBatteryLevel() >= getRemoteConfig(BATTERY)$ OR
        $batteryProfiler.phoneIsCharging()$) AND $memoryProfiler.getAvailableRAM() >=$
        $getRemoteConfig(RAM)$ then
 7:         return $LOCAL$
 8:     end if
 9:     if $networkProfiler.getNetworkIsConnected() == true$ then
10:         if $networkProfiler.getAvailableNetwork() == WIFI\ OR\ MOBILE$ then
11:             switch ($networkProfiler.getConnectionStatus()$)
12:                 case EXCELLENT:
13:                 case GOOD:
14:                 case MODERATE:
15:                     $OFFLOAD\_CHOICE \leftarrow CLOUD$
16:                 case POOR:
17:                 case UNKNOWN:
18:                 default:
19:                     $OFFLOAD\_CHOICE \leftarrow LOCAL$
20:         end if
21:     end if
22:     return $OFFLOAD\_CHOICE$
```

5.2 Dynamic Partitioning

We perform dynamic partitioning using Java Reflection to monitor classes for annotated methods that can be offloaded. When this method is discovered, we run our offloading decision engine to determine where to perform the task. If the result gotten from the engine is CLOUD, we reassign the 'pushRemote' and 'offloadDecision' variables with necessary information to perform the task offload. As stated before, the engine uses the device's current context and this happens during runtime thereby making it a dynamic approach. Step wise series of this approach is given below:

1. Initialize reflection class with device context
2. Pass context to decision engine
3. Get an array of all methods from class object
4. Check if @PushRemote annotation is present in class
5. If present, run decision engine to get offload choice
6. If offload decision is cloud, set pushRemote = true and offloadDecision = CLOUD else, leave default option which is local execution

5.3 Offloading Decision Engine

As discussed before, before we can make use of this engine, we have to pass the SMD's context to it. This context is then used by our BatteryProfiler, NetworkProfiler and MemoryProfiler in algorithm 1 to make adequate offloading decision. These three profilers are discussed below:

- **BatteryProfiler:** This class contains two methods (getBatteryLevel() & phoneIsCharging()) which we use to get the current battery level of the SMD and also if the SMD is currently in a charging state. The phoneIsCharging() method returns a boolean value of 'true' if the phone is currently charging or has battery level of 100%. These two methods are called by algorithm 1.

- **NetworkProfiler:** This class determines the network availability of the SMD. We first define using the SMD's context if there is connectivity and if there is, we run a function to define what type of network is available. Available network connectivity is in two options which are WIFI and MOBILE. Depending on which is available, we run a function that determines the strength of the network and then assign a value that matches the strength. The available values for network strength are EXCELLENT, GOOD, MODERATE, POOR AND UNKNOWN. This result can be gotten by calling the getConnectionStatus() method the NetworkProfiler exposes.

- **MemoryProfiler:** This class is basically used to get the current available RAM of the SMD. It exposes a method called getAvailableRAM() which uses the device's context to return RAM that is currently not utilized.

After these classes have been initialized in the engine, the first thing we do is to check if the SMD has available resources to run the compute task. We do this by checking battery life and available memory. If the values from this check equal default values set in our remote configuration, we run the task locally else, we check for network availability to determine if we can offload the task to the cloud. Depending on network availability and as shown in algorithm 1, if this result is EXCELLENT, GOOD OR MODERATE, we can then offload to the cloud. If the network availability is POOR OR UNKNOWN, then we have to remain with the default and run the task locally since offloading will not be possible.

5.4 Remote Mobile Clients

Having remote clients as part of system is to apply heterogeneity to the system. We have implemented this by building a separate mobile application with method replication of methods that can be offloaded. When a user volunteers to offer their free resources, we create a unique record for them on Firebase Cloud Firestore. The mobile application can then listen in realtime to that record to know if it has been assigned a task to perform and also what type of task it was assigned. After performing the task, it updates that same record with result gotten so the requesting device can get it also in realtime. In order to ensure participating devices have adequate resources to offer, the mobile application at regular intervals sends device information such as battery, network status and available memory to Firebase Cloud Firestore. This ensures that best unassigned device is selected to perform a task.

5.5 Remote Configuration

This option was implemented to give the ability to change configurations in our system without the need to modify source code. We store all remote configurations on Firebase Remote Config [5] and these can be changed at anytime on the cloud platform and will reflect on the mobile application. We implement this by using the remote config library provided. In a scenario that there is no network availability, the library uses default values hardcoded in the mobile application. In our system, we use remote config for default values for battery life and available RAM in our decision engine. This makes it easy for the developer to change depending on requirements. We also use it for our offloading resource options. By default, our system is set to offload tasks to our serverless functions and using remote config, we can change this to offload to available remote clients depending on requirements.

6 Evaluation

In this section, we present analysis of results of various experiments carried out using our approach. We also discuss findings and how they support the objectives of this research work. For experiments carried out, we have built two android applications which implement our approach. Also, we have set up various virtual android devices with different context parameters (battery, network and RAM) in order to properly test our context-aware decision making algorithm. Results are given below.

6.1 Experiment 1 / N-Queens Problem

The N-Queens Problem involves placing N number of queens on a chessboard in such a way that no queen will be able to attack another. We were able to implement this problem using the backtracking algorithm which is a recursive approach to finding a solution step by step. For this experiment, we have set up three scenarios where the task will run locally (i.e resources are available for local execution), run using serverless functions and run using the remote mobile clients. To obtain results, we run each scenario five times and get the average execution time. For $N = \{8, 16, 24, 32\}$, we achieved the following results for the different scenarios illustrated in figure 3. Each number for N also represents the chessboard size i.e $N = 16$ will represent a 16 x 16 chessboard with 16 queens.

From figure 3, we can observe that for $N = \{8, 16\}$, running the task locally seems more efficient than offloading to cloud resources due to lesser execution time. This is in the scenario that the SMD has adequate resources to run the task. Still looking at $N = \{8, 16\}$, the difference in execution time for running the task using serverless functions as compared to running locally is not too much. The serverless function has a slightly higher execution time due to latency overhead. This is also displayed when running tasks using the remote mobile clients. As the value of N begins to grow, we can observe that running the task using serverless functions is more beneficial than running local or using the remote mobile clients. This happens because the SMD has limited resources and the serverless function can scale up to properly handle the increasing value for N. Running the task using remote mobile clients has the highest execution time for all values of N. This is due to the overhead of looking for a suitable client to run the task coupled with the task execution and return of results gotten to the offloading device. This is

[5] Firebase Remote Config: `https://firebase.google.com/docs/remote-config`

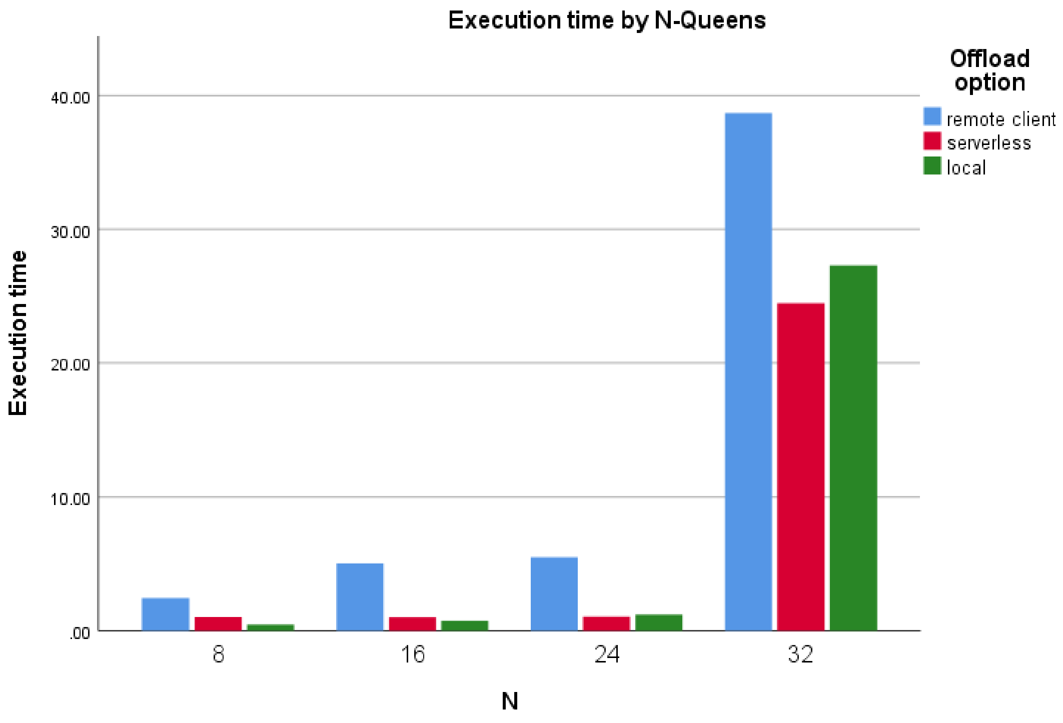

Figure 3: Overall Execution Time for $N = \{8, 16, 24, 32\}$

the major reason why we chose executing offloaded tasks on serverless functions as our default option.

6.2 Experiment 2 / Optical Character Recognition (OCR)

Here we present results from an OCR application with our approach implemented. For this experiment, we use two images with different sizes (large and small). The large image contains lots of text while the small one does not. We use the same scenarios identified in section 6.1 above. Results are shown in figure 4. For the smaller image with less text, local execution was faster followed by using the serverless function and remote clients. The serverless execution has higher execution than local due to the overhead of uploading the image before we can perform the OCR task on it. For the large image, we can see that the serverless function has execution time of almost 10 seconds while local is almost at 25 seconds. This is a huge difference considering the overhead attached to offloading to serverless function. All task executions on the remote clients have higher execution times. As discussed in section 6.1 above, it is due to overhead of looking for a suitable device then performing the task and returning the results to the requesting device.

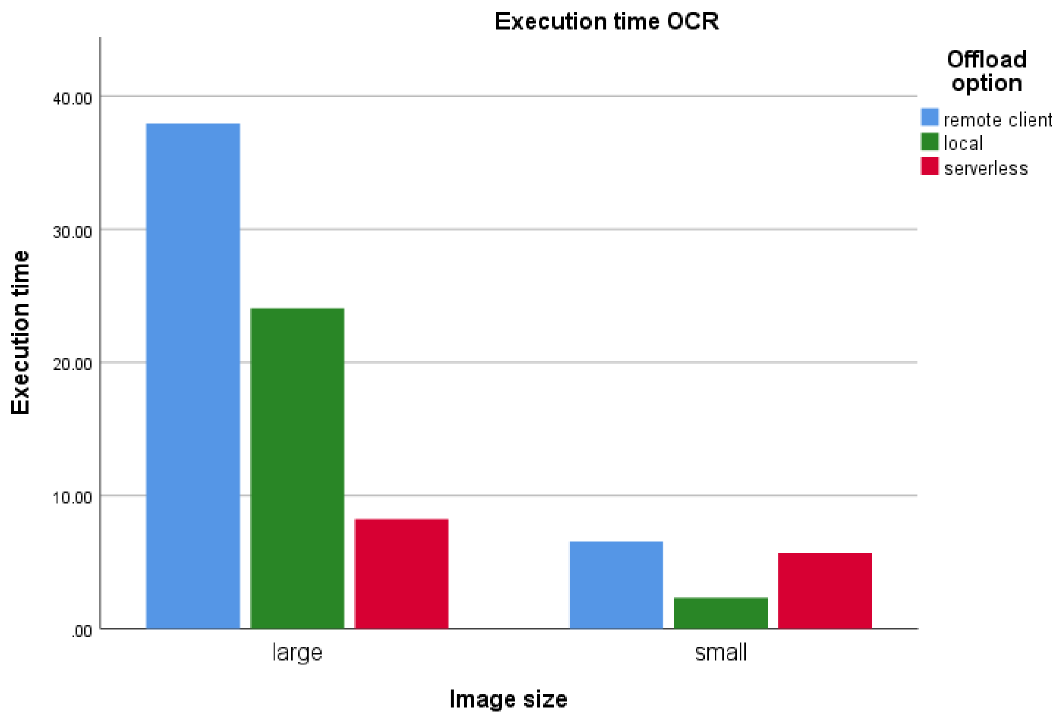

Figure 4: Overall Execution Time for OCR

6.3 Experiment 3 / Offloading Decision Based on Context

In this section, we set up different virtual android devices with different context parameters in order to determine if our context aware decision algorithm is functioning properly i.e choosing the right offloading decision. For this experiment, the default battery level is set at 90% and default RAM level is set at 1GB. This result is presented in the table 2 below.

Table 2: Offloading Decision Engine Test

Device	Context	Expected Decision	Gotten Decision
#1	Battery - 100% Network - GOOD RAM - 1 GB	LOCAL	LOCAL
#2	Battery - 100% Network - GOOD RAM - 500 MB	CLOUD	CLOUD
#3	Battery - 80% Network - GOOD RAM - 1 GB	CLOUD	CLOUD
#4	Battery - 80% Network - POOR RAM - 1 GB	LOCAL	LOCAL
#5	Battery - 20% Network - EXCELLENT RAM - 600 MB	CLOUD	CLOUD

From table 2 above, we can see that our decision engine is functioning properly and making correct offloading decisions based on changing device context.

6.4 Discussion

Results from our experiments have shown that our approach is functioning properly. Our decision making algorithm makes the right offloading decision based on the SMD's changing context as shown in table 2. Results from experiment 6.1 and 6.2 have shown that the SMD was able to handle low compute intensive task properly locally in a time efficient manner thereby utilizing less resources. This is not the case when the task became more intensive. Execution time became higher thereby consuming more resources of the SMD such as battery life, CPU and RAM. When compared to running offloaded tasks using the serverless functions, execution time was reduced when the task became more intensive as shown in figures 3 and 4. This result was achieved even with the network overhead associated with offloading the task and returning the results. As regards to offloading with remote mobile clients, this option might not be the best for latency sensitive applications. This is due to the overheads attached to it such as looking for an appropriate device, assigning the task to the device and lastly waiting for the task to be completed and result returned. Also, it is dependent on devices being available and have adequate resources to perform offloaded tasks. With results obtained in general, we have been able to achieve the objectives of this research paper.

Our results achieved have shown to be in line with results gotten from previous related works and when comparing our approach to other related works, our approach is better in the sense that we have applied remote configurations which can be used to change parameters used in our offloading decision algorithm without the need to modify source code. Also, most papers implemented their approach using cloud VMs while we apply ours using serverless computing. When using cloud VMs, in most scenarios, the cost is always higher when compared to using serverless computing. This cost difference is shown in table 3 for some related works. For this comparison, we assume that each task runs for 300 ms every five minutes for an hour. Total cost of execution will be cost of 12 executions in an hour x 300 ms x the unit cost.
**Note: UC = Unit Cost, TC = Total Cost

Table 3: Price Comparison for running task for 300 ms every 5 minutes / hour

	Instance	**Billed at**	**UC**	**TC (300ms)**
Our approach	GCP Cloud Functions (2 GB)	100 ms	\$0.000002900	\$0.01044
Zhou et al. (2017)	AWS EC2 t2.medium (4 GB)	1 Hour	\$0.0464	\$0.0464
Islam et al. (2020)	AWS EC2 t3.xlarge (16 GB)	1 Hour	\$0.1664	\$0.1664

From table 3 above, we can see that using cloud functions in our approach offered the lowest cost when compared to instances used by other related works. With this result, we can state that our approach is cost efficient and is one of the objectives we aimed to achieve with this research paper. One limitation of using the serverless approach is the problem of cold starting. Before functions can be executed, the execution environment has to be set up which can then lead to additional execution time. This problem usually occurs when a function is deployed and used for the first time. Subsequent function invocations and steady load will make the impact of the cold start negligible.

This research paper has been able to contribute to the field of cloud computing by exposing cloud services that can be implemented in mobile application development and enhance the usage of the MCC. Cloud computing offers various services by different CSPs. We have been able to implement three of such services being offered by GCP. These services include Remote Config, Cloud Firestore Realtime database and Cloud Functions (serverless). These services offer lots of advantages and we have shown how it can be utilised in mobile cloud computing thus leading to a potential better acceptance of the model.

7 Conclusion and Future Work

In this research paper, we aimed to determine if the dynamic partitioning of a mobile application running compute intensive task(s) can be made to function more efficiently by applying the code offloading model in a context aware HMC environment while also reducing costs that might be incurred on users. Our objectives included performing a study on the state of art, implementing a cost effective code offloading approach and perform system evaluation. We were able to achieve all these by building an offloading decision engine which used the SMDs context (i.e battery, network, memory) at runtime to make adequate offloading decisions. We profiled intensive methods by annotating them and was able to achieve offloading through method replication on our remote offload resources. Our offload resources included GCP serverless cloud functions and remote mobile clients which proved to be cheaper alternatives when compared to using cloud VMs.

Also, we were able to add flexibility to our approach by using remote configurations which we used to change system parameters without the need to modify source code. We performed evaluations of our system and results proved that our decision making algorithm provided adequate offloading decisions based on context and execution time was significantly reduced (up to 66%) for heavy computational tasks when offloaded.

One limitation identified in this paper is the cold starting of the serverless functions. This is due to the setting up of the execution environment when a function is being deployed and run for the first time. This limitation becomes negligible when the functions receive subsequent invocations and steady load. For a future work, we aim to propose an adequate incentive mechanism for users who want to be part of the remote mobile client system. If users are incentivized, there will be potential for increased participation therefore increasing the number of devices available to perform offloaded tasks.

References

Akherfi, K., Gerndt, M. and Harroud, H. (2018). Mobile cloud computing for computation offloading: Issues and challenges, *Applied Computing and Informatics* **14**(1): 1 – 16. doi: 10.1016/j.aci.2016.11.002.

Alonso-Monsalve, S., García-Carballeira, F. and Calderón, A. (2018). A heterogeneous mobile cloud computing model for hybrid clouds, *Future Generation Computer Systems* **87**: 651 – 666. doi: 10.1016/j.future.2018.04.005.

Benedetto, J., Valenzuela, G., Sanabria, P., Neyem, A., Navón, J. and Poellabauer, C. (2018). Mobicop: A scalable and reliable mobile code offloading solution, *Wireless Communications and Mobile Computing* **2018**. doi: 10.1155/2018/8715294.

Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A. S. and et al. (2018). A manifesto for future generation cloud computing: Research directions for the next decade, *ACM Comput. Surv.* **51**(5). doi: 10.1145/3241737.

Chen, X., Chen, J., Liu, B., Ma, Y., Zhang, Y. and Zhong, H. (2019). Android-off:offloading android application based on cost estimation, *Journal of Systems and Software* **158**: 110418. doi: 10.1016/j.jss.2019.110418.

Chen, X., Chen, S., Zeng, X., Zheng, X., Zhang, Y. and Rong, C. (2017). Framework for context-aware computation offloading in mobile cloud computing., *Journal of Cloud Computing (2192-113X)* **6**(1): 1. doi: 10.1186/s13677-016-0071-y.

Flores, H., Hui, P., Nurmi, P., Lagerspetz, E., Tarkoma, S., Manner, J., Kostakos, V., Li, Y. and Su, X. (2018). Evidence-aware mobile computational offloading, *IEEE Transactions on Mobile Computing* **17**(8): 1834–1850. doi: 10.1109/TMC.2017.2777491.

Gai, K., Qiu, M., Zhao, H., Tao, L. and Zong, Z. (2016). Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing, *Journal of Network and Computer Applications* **59**: 46 – 54. doi: 10.1016/j.jnca.2015.05.016.

Gu, F., Niu, J., Qi, Z. and Atiquzzaman, M. (2018). Partitioning and offloading in smart mobile devices for mobile cloud computing: State of the art and future directions, *Journal of Network and Computer Applications* **119**: 83 – 96. doi: 10.1016/j.jnca.2018.06.009.

Islam, A., Kumar, A., Mohiuddin, K., Yasmin, S., Khaleel, M. and Hussain, M. (2020). Efficient resourceful mobile cloud architecture (mrarsa) for resource demanding applications, *Journal of Cloud Computing* **9**(1). doi: 10.1186/s13677-020-0155-6.

Lee, H. and Lee, J. (2018). Task offloading in heterogeneous mobile cloud computing: Modeling, analysis, and cloudlet deployment, *IEEE Access* **6**: 14908–14925. doi: 10.1109/ACCESS.2018.2812144.

Lin, X., Jiang, J., Li, B. and Li, B. (2015). Circa: Offloading collaboratively in the same vicinity with ibeacons, *2015 IEEE International Conference on Communications (ICC)*, London, UK, pp. 3751–3756. [Online]. Available: IEEE Xplore, https://ieeexplore.ieee.org/document/7248908 [Accessed on: Mar. 23, 2020].

Lu, F., Gu, L., Yang, L., Shao, L. and Jin, H. (2020). Mildip: An energy efficient code offloading framework in mobile cloudlets, *Information Sciences* **513**: 84–97. doi: 10.1016/j.ins.2019.10.008.

Mahmoodi, S. E., Uma, R. N. and Subbalakshmi, K. P. (2019). Optimal joint scheduling and cloud offloading for mobile applications, *IEEE Transactions on Cloud Computing* **7**(2): 301–313. doi: 10.1109/TCC.2016.2560808.

Nawrocki, P., Sniezynski, B. and Slojewski, H. (2019). Adaptable mobile cloud computing environment with code transfer based on machine learning, *Pervasive and Mobile Computing* **57**: 49 – 63. doi: 10.1016/j.pmcj.2019.05.001.

Noor, T. H., Zeadally, S., Alfazi, A. and Sheng, Q. Z. (2018). Mobile cloud computing: Challenges and future research directions, *Journal of Network and Computer Applications* **115**: 70 – 85. doi: 10.1016/j.jnca.2018.04.018.

Xu, M., Qian, F., Zhu, M., Huang, F., Pushp, S. and Liu, X. (2020). Deepwear: Adaptive local offloading for on-wearable deep learning, *IEEE Transactions on Mobile Computing* **19**(2): 314–330. doi: 10.1109/TMC.2019.2893250.

Zhou, B. and Buyya, R. (2018). Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions., *ACM Computing Surveys* **51**(1): 13:2 – 38. doi: 10.1145/3152397.

Zhou, B., Dastjerdi, A. V., Calheiros, R. N., Srirama, S. N. and Buyya, R. (2017). mcloud: A context-aware offloading framework for heterogeneous mobile cloud, *IEEE Transactions on Services Computing* **10**(5): 797–810. doi: 10.1109/TSC.2015.2511002.

Zhou, B., Srirama, S. N. and Buyya, R. (2019). An auction-based incentive mechanism for heterogeneous mobile clouds, *Journal of Systems and Software* **152**: 151 – 164. doi: 10.1016/j.jss.2019.03.003.