MAMoC-Android: Multisite Adaptive Computation Offloading for Android Applications

Dawand Sulaiman and Adam Barker
School of Computer Science
University of St Andrews
St Andrews, UK
Email: {djs21, adam.barker}@st-andrews.ac.uk

***Abstract*—Computational offloading has been widely used to improve the performance of mobile applications and conserve the energy of mobile devices. Prior studies have primarily focused on a form of offloading where only a single server is considered as the offloading site. However, mobile devices now have access to a range of nearby mobile and fixed devices and multiple cloud providers. This paper proposes a method for multisite computation offloading in dynamic mobile cloud environments, in order to save energy and improve application execution time. Our proposed dynamic offloading decision algorithm takes into consideration the offloading score and records of past offloading executions to select the best candidate(s) for offloading. Multisite offloading execution achieves a greater reduction with respect to the completion time and energy consumption of mobiles when compared to local execution or a single-site offloading execution on a public cloud instance.**

I. Introduction

Mobile devices have become an essential part of modern life in this new era of mobile computing and Internet of Things. The number of connected devices is estimated to reach 50 billion by the year 2020 [1], [2]. We are facing a contradiction between inadequate processing capacity of mobile devices and the users' ever-growing need for better performance and longer battery life. A wide range of applications are now executed on mobile devices, many of which demand high computational power. Backed by the unbounded resources of cloud computing, Mobile Cloud Computing (MCC) can meet the demands of even the most computationally and resource-intensive applications. Although, MCC has helped many application developers to overcome the limited resources of mobile devices, it has also created a new set of challenges, such as the possibility of high network latency and low bandwidth availability between the mobile device and the cloud.

Mobile cloud computation offloading has become a promising method to reduce execution time and save battery life of mobile devices. The process involves augmenting execution through migrating heavy computation from mobile devices to resourceful cloud servers and then receive the results from them via wireless networks. The constraints of the mobile devices in terms of execution power and battery life makes the idea of offloading attractive. Unfortunately, offloading to public cloud infrastructure is not always guaranteed to be time efficient and energy conserving [3]. When the network bandwidth is fairly limited, it may be too slow to transmit data between mobile devices and remote servers; when the network status is highly unstable, maintaining a connection to a cloud might consume more energy than local computation.

This paper presents *Multisite Adaptive Mobile Cloud (MAMoC)*-Android, a mobile client framework which allows an Android mobile device within the shared environment to offload its tasks (classes or methods) to other external platforms including nearby mobile devices running Android OS, fixed edge devices (also called cloudlets[4]) such as laptops and desktops or public cloud instances such as AWS and Microsoft Azure. The objectives of our proposed solution include performance enhancement in terms of computational time by offloading resource intensive computations to more powerful external resources, energy efficiency by reducing the computational overhead on the mobile device, context awareness by making smart offloading decisions considering the associated cost of computation and offloading delay, code reusability by following a highly modular approach, and high adoptability by keeping the adoption of the application model easy for the application providers.

The main contributions of this work are:

1) We present an enabling mechanism for context-adaptive computation offloading to support resource-constrained mobile devices with multiple destination clouds.
2) A dynamic offloading decision making algorithm taking into consideration calculated offloading scores of the mobile device and connected nodes and past local and remote executions.
3) The design and development of an Android offloading-enabled framework that can be adopted by developers to build Mobile Edge Cloud (MEC) [5] applications. Moreover, we have established a lightweight runtime environment which is under 200MB in size to serve the offloading requests from the mobile applications.

The rest of this paper is organized as follows: in Section II we present related work and what differentiates our work from the current mobile computation offloading frameworks. The design and implementation of our framework is presented in Section III. Section IV presents the evaluation of the proposed framework through running three demo applications. Finally, conclusion and future works are discussed in Section V.

II. Related Work

The goal of offloading computation from a mobile device to more powerful surrogates has been looked at for many years now for performance gains and reducing energy consumption of the mobile devices [6], [7], [8]. In a recent comprehensive survey [9], authors focus on Mobile Cloud Computing augmentation frameworks with a discussion of pertinent research challenges in MCC augmentation such as service heterogeneity and service context-awareness among others. Other research efforts investigated frameworks which allow the cloud to move closer to the user in the form of cloudlets [4], [10]. Other approaches include a group of nearby mobile devices to leverage lower end devices thus the formation of a local mobile cloud [11] also referred as *Mobile Device Clouds* [12] and *Mobile Edge Clouds* [13]. Authors at [14] proposed a context-aware offloading scheme and considered all the cloud resource types (nearby mobile devices, cloudlets, and public cloud) as candidate offloading destinations.

Many early work in the area of Mobile Cloud Computing only considered offloading decision between mobile devices and public cloud services. Most existing MCC proposals concentrate on single-site offloading [6] i.e., offloading application's parts from the mobile device to a single server. However, as the number of surrounding devices and cloud computing and storage increases, it is more common that an application can be executed on multiple servers [15]. It is shown that we can obtain better performance from multisite offloading. Therefore, multisite offloading is considered as a generally realistic model in this work.

Most of the research work in the area of mobile computation offloading involves designing intelligent decision engines to predict the execution time and energy consumption of an Android mobile computation and decide whether to offload to external resources based on the measurements [16][17]. There are also research attempts in reducing the overhead of setting up the external servers to enable offloading for mobile devices. Researchers of [18] use LXC containers to reduce the offloading Virtual Machine sizes and startup time of runtime environments. Unikernels are also used in [19], it is demonstrated that the runtime is more lightweight in bootup time, memory footprint, image size and energy consumption compared with traditional runtime like Android VM or Android container.

The differences between our framework and other frameworks in the literature is in the fact that most of the offloading systems in the literature are built with single-site offloading in mind. We leverage the constrained mobile devices with a wide range of external resource providers including nearby devices, edge devices, and remote cloud servers. Moreover, we do not require a specialized Android x-86 [1] server to be run in the server side. We can use any kind of general-purpose server that is capable of installing Python and openJDK environments.

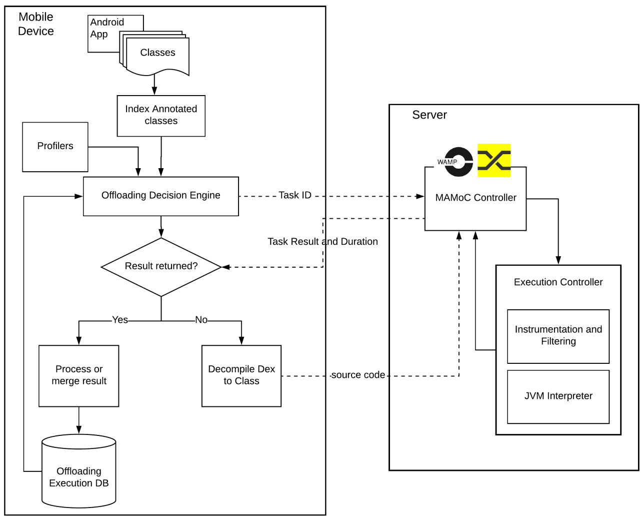


Figure 1: MAMoC task offloading execution workflow

We have provided two lightweight Docker images [2] [3] that only take few seconds to build and run to receive requests from mobile devices. Another advantage of using a general-purpose server would be to allow serving other clients which are not running Android OS such as iOS and Windows Phone mobile users.

III. Design & Implementation

Our objectives for MAMoC include improving the running time of the compute-intensive tasks on the mobile devices as well as saving energy consumption. MAMoC is designed to allow mobile devices to discover other surrounding devices over standard Local Area Network in infrastructural Wi-Fi using an access point or peer-to-peer Wi-Fi. Figure 1 demonstrates the offloading execution workflow in our system. When an offloadable task is invoked in the mobile application, the offloading decision engine has to decide whether to execute it locally or offload it to remote resources, we will further discuss the decision making process in Section III-A3. A more detailed explanation of offloading procedure in our framework will be described in Section III-B.

A. Components

1) Service Discovery: After the framework is initialized, service discovery is performed. Each mobile device can advertise services and discover what services other nearby devices on the local network are offering. A browser object in a host device searches for peers which have an advertiser object. Android provides helper libraries for managing Device-to-Device (D2D) communications. This can be done using infrastructural Wi-Fi, where the devices are connected to the

[1]http://www.android-x86.org/

[2]https://hub.docker.com/r/dawan/mamoc_server/

[3]https://hub.docker.com/r/dawan/mamoc_router/

same Access Point using Network Service Discovery (NSD) [4] or Wi-Fi Peer-to-peer [5] for direct D2D communications.

2) Profilers: This component collects real time information of the devices including hardware, software, and networking related context information. This allows the framework to have sufficient information about the connected devices. This eases the process of offloading decision making. Once a new mobile device or a remote server join the framework, it goes through a set of profiling mechanisms using the following profiling subcomponents:

- Device Profiler: Profiling hardware status of mobile devices is an essential step towards taking suitable offloading decisions. We collect the number of CPU cores, The current and maximum frequency of CPU, available and total RAM. For the host device and the discovered nearby devices, we monitor changes in battery of the device. We check the state of the battery (charging or not charging) and the level of battery (0-100).
- Network Profiler: The objective of this profiler is to measure the quality of the network connection between the mobile device and connected remote nodes. Unpredictable communication link is the main bottleneck in making adaptive computation offloading decisions, since the offloading delay and energy consumption vary based on the quality (bandwidth, latency) of the communication link. Consequently, whether to offload or execute locally decision may change due to inconsistent communication link. On the mobile device, we continuously monitor the network status changes in the mobile device using ConnectivityManager [6]. To aid the offloading decision making, network profiler also measures the Round Trip Time (RTT) between the host device and the connected nodes. All the RTT values are recorded along with other execution details in a private database in the mobile device for future references.

3) Offloading Decision Engine: The engine is fed with the metrics collected from profilers and it is used by the framework to check whether to offload a task to other devices or execute it locally. The engine also depends on the past offloaded and local executions which are saved in a private database in the mobile device. The decision is made according to the steps shown in Algorithm 1. We have statically defined the values for *maxLocalExecutions* and *maxRemoteExecutions* to be 5. These values are used against the five most recent consecutive local or remote executions. We calculate normalized offloading scores for each of the connected nodes in our framework using an Analytical Hierarchy Process (AHP) multi-criteria method [20]. We use price, speed, bandwidth, availability, and security as criteria for our alternatives (offloading sites). The price is a cost criterion whereas the others are benefit criteria. We assume the priory of importance is ranked as: bandwidth > speed > availability > security > price in performing the pair-wise comparisons. However, the priority of these five criteria can be various in other situations. After calculating the weights of each criterion (Bandwidth: 0.4072, Speed: 0.3885, Availability: 0.1083, Security: 0.05729, Price: 0.0384), the final ranking of the offloading sites will be generated using our evaluation matrix considering all the five criteria simultaneously.

We have created an Android service to make offloading decision based on run-time information collected by the aforementioned profilers. We use a combination of network connectivity information and historical data of task offloading to make an offloading decision, Algorithm 1 demonstrates the steps involved in offloading decision making. Every time an offloadable task is initiated, the engine determines if it is beneficial to offload it. Because of the uncertainties inherent in the mobile environment, the offloading decision takes risk into consideration. In case a bad decision has been made, it will also adjust its strategy with new information available.

Algorithm 1 Task Offloading Decision Algorithm

```
Input: taskName, localResults, remoteResults, nodes
Output: ExecutionLocation
 1: selfNode = getSelfNode()
 2: localScore = getSelfOffloadingScore()
 3: nodeScores = getOffloadingScoresforNodes(nodes)
 4: maxScore = getMaxOffloadingScore(nodeScores)
 5: for result in localResults do
 6:     if result.taskName is taskName then
 7:         localExecutions.add(result)
 8:     end if
 9: end for
10: if localExecutions > maxLocalExecutions then
11:     if remoteResults is empty OR localScore < maxScore then
12:         return nodeScores.objectFor[maxScore]
13:     end if
14: end if
15: for result in remoteResults do
16:     if result.taskName is taskName then
17:         remoteExecutions.add(result)
18:     end if
19: end for
20: if remoteExecutions > maxRemoteExecutions then
21:     if localResults is empty OR localScore > maxScore then
22:         return selfNode
23:     end if
24: end if
```

B. Implementation

In MAMoC-enabled mobile applications, Each task is identified by a unique ID which will be looked up in the remote server to check if it is previously been offloaded. All the annotated (offloadable) tasks are indexed and saved in a

[4]https://developer.android.com/training/connect-devices-wirelessly/nsd

[5]https://developer.android.com/training/connect-devices-wirelessly/nsd-wifi-direct

[6]https://developer.android.com/reference/android/net/ConnectivityManager

Table I: Comparing previous and current work

Aspect	previous work[21]	current work
Mobile OS	iOS	Android
Goal	Time saving	Time and energy saving
Server	Simple	Sophisticated
Client language	Swift	Java
Server language	Swift	Python

metafile during the launch of the application. This allows for an easy retrieval of the source code of the task when it is needed to be sent over to the remote server. This procedure depends on the offloading service provider. If the offloading execution location is a nearby mobile device, we will simply use Java Reflect to execute the task in the connected mobile device. However, if the location selected by the offloading decision engine is an edge device or a public cloud instance, we need to retrieve and send over the Java source code of the offloaded task.

In our previous work [21], we used Swift programming language to develop a multisite offloading framework for iOS mobile devices. The differences between our previous and current work are shown in Table I.

1) MAMoC Client: We have implemented our client library in Android Studio and distributed it to Maven-Central repository[7], which can be added to any Android development application or plugin using Android Studio's Gradle mechanism. Once added to an application's build file, the developer has full access to our API public methods (e.g. mamocFramework.start(), mamocFramework.execute(ExecutionLocation.EDGE), and many others).

Application developers can use Java Pluggable Annotation Processing API to annotate the heavy tasks with @*Offloadable* annotation. This can be seen as a metadata added to the source code and is assigned to any class or method that can be offloaded to external candidate nodes without depending on any native library components in the mobile OS. An example class is KMP [22] class which is purely dependent on Java calls and can be executed on any node with a JVM interpreter. If the computation of text search problem is done in a method which is invoked in an Android activity then the method is annotated instead. The @*Offloadable* annotation has two boolean optionals:

- *parallelizable*: The tasks of some embarrassingly parallel programs can be run independently. There are no dependencies between the subtasks of the task. An example is a text search task which can be split across a number of computing nodes without any data exchanges between them. The offloading node can partition the task (the text file in this instance) and send it over to external resources. After the results are returned, they are merged and presented as one result the same way it is presented as if the execution were to happen locally.
- *resourceDependent*: The tasks which are dependent on resources need to be available at the time of processing. Examples of resources used in mobile apps can be in the form of text files (word search and sorting workloads in word processing apps) , images (face detection and recognition apps), audio files (translation apps). The resource files in Android apps are statically added to Assets folder or the assigned resources directory which include XML files for layout design and global values. Any @*Offloadable* class or method, which has set this optional element to true, needs the data to be present at the remote site before being processed.

[7]https://search.maven.org/

2) MAMoC Server: Our server is written in Python with Java code execution support using a JVM interpreter. The two components of the server side include a router and a custom server. For routing client requests, we use Crossbar [8] while Autobahn [9] is used for handling Remote Procedure Calls and Publish/Subscribe events. Routers are the core facilities of Crossbar, responsible for routing Web Application Messaging Protocol (WAMP) [10] Remote Procedure Calls (RPC) between callers and callees, as well as routing WAMP Publish-Subscribe (PubSub) events between publishers and subscribers. We allow a node to interact with the local infrastructure available. This mechanism is implemented by establishing a control channel for command streams and monitoring services based on WebSocket. WAMP, our choice of asynchronous transport and delivery system for message-encapsulated commands, is a sub-protocol of WebSocket, in its turn a standard HTTP-based protocol providing a full duplex TCP communication channel over a single HTTP-based persistent connection. We use psutil [11] for profiling the remote servers for CPU power, available memory and network information and publish it to the subscribed mobile devices.

The server component is capable of transforming Android offloadable classes to Java source code. Algorithm 2 shows the necessary steps taken when a new request arrives.

Algorithm 2 Decompiled Android code transformation in the server

```
Input: sourceCode, resourceName, parameters
Output: result, duration
 1: start ← startTimer()
 2: code ← removePackageName(sourceCode)
 3: code ← removeAnnotations(code)
 4: className ← findClassName(code)
 5: code ← addMainMethod(code)
 6: if resourceName is not empty then
 7:     code ← addResourceCode(code)
 8: end if
 9:
10: result ← executeCode(className, code, parameters)
11: duration ← endTimer() - start
12: return result, duration
```

[8]https://crossbar.io

[9]https://crossbar.io/autobahn/

[10]https://wampws.org

[11]https://pypi.org/project/psutil/

An array of method parameters are sent over and their types are dynamically invoked to be called in the main method of the class. As an example, we demonstrate KMP class which is in one of our demo Android applications. Listing **??** shows the original Android code.

When the task is executed on the Android app, it is first checked whether the remote procedure is already registered by fetching the list of registered procedures in our server by calling *wamp.registration.list*. If the procedure exists, we only send the parameters and resource names (if any). Otherwise, the following event will be published which the server is subscribed to:

publish("uk.ac.standrews.cs.mamoc.offloading", "Android", "uk.ac.standrews.cs.mamoc.SearchText.KMP", Decompiled KMP SourceCode, "large.txt", searchKeyword)

After the server receives it, the Java code in Listing **??** will be generated.

The result of execution and duration in milliseconds are then published to the device which have subscribed to *uk.ac.standrews.cs.mamoc.offloadingresult* topic.

Listing 1: NQueens on Android

```
package uk.ac.standrews.cs.mamoc.NQueens;
@Offloadable
public class NQueens {
     int n;
     public NQueens(int N){
          this.n = N;
     }
     public void run() {
      ...
}
```

Listing 2: NQueens on server side

```
public class NQueens {
 public static void main(String[] args){
  new NQueens(Integer.parseInt(args[0])).run();
 }
     int n;
     public NQueens(int N) {
      ....
}
```

The complete source code of MAMoC and a short documentation for setting up the different components in the framework is publicly available online at *https://github.com/dawand/MAMoC-Android.* The server component which can also be pulled from Docker hub is also available at *https://github.com/dawand/MAMoC-Server.*

IV. Results & Discussion

The main goals of the framework are to allow mobile application developers to achieve a transparent automated offloading to multiple destination clouds (mobile clouds and public clouds) and device dynamic changes over the lifecycle of execution of an application. MAMoC has been designed to improve the execution time of mobile apps through better offload decision making. This results in reduced energy consumption and improved responsiveness. This section evaluates MAMoC, demonstrating that it achieves its design goals. Our main experiments have been carried out through a real-world testbed deployment with three mobile applications.

A. Experimental Setup and Testbed

Our main experiment considers a scenario where a mobile app is associated with a nearby mobile device, an edge device, and a public cloud instance to improve energy and performance on the users' devices. Testing was done on two stock Android mobile devices. For a slightly older and a lower-end device, we used a Nexus 7 tablet released in 2013 with a Quad-core 1.5 GHz CPU and 2GB of RAM; for a higher-end mobile device, we used a Google pixel phone with a Quad-core CPU (2x2.15 GHz Kryo & 2x1.6 GHz Kryo) and 4GB of RAM. A laptop is used as an edge device running the containers on Docker Engine for Mac [12]. The server component was deployed on a AWS c4.4xlarge instance type. We chose the AWS region (London) with the minimum latency (23 ms) from our school network to deploy the remote cloud instance [13]. The detailed specifications of our testbed devices are shown in Table II.

B. Demo Applications

We have developed three demo applications to measure their completion time and energy consumption when executed locally or offloaded to external resources.

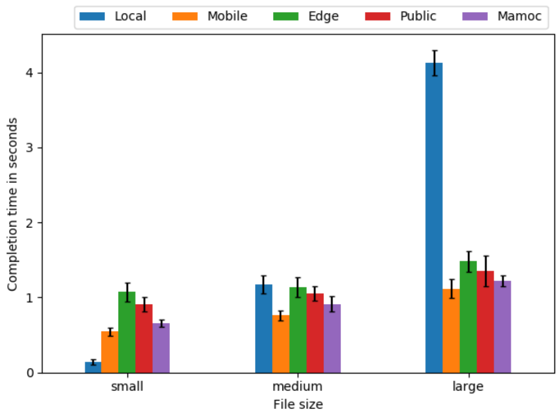
- Text Search: It allows a user to enter a keyword and select a file size from (small, medium, and large) to find the occurrences of the word in the file. Knuth-Morris-Pratt string searching algorithm [22] is used. This is an example of an embarrassingly parallel task since it can be independently run on multiple nodes hence the *parallelizable* annotation optional value is set to true. The external node that performs a full or partial search should have access to the text file so we need to send the file over hence the *resourceDependent* is also set to true. The entered keyword in the mobile device by the user needs to be sent over as a parameter to the remote resource.
- Quick sort: Quicksort Algorithm is used as the sorting algorithm. The mobile device or the service provider needs to fetch the content of the text file and apply the sorting algorithm to the list of words. It does not have any required parameters.
- N-Queens: The task is to enumerate the placements of all N valid queens on an N x N chess board such that no queen is in range of another. We have run 6 different N value tests from $N = 8$ to $N = 13$. It does not depend on any resource so we only need to send the required N (number of queens) parameter over in offloading scenarios.

Text search and Quick sort examples are data dependent. We have used three common files with different sizes. The large text file consists of 1,095,649 words, the medium text

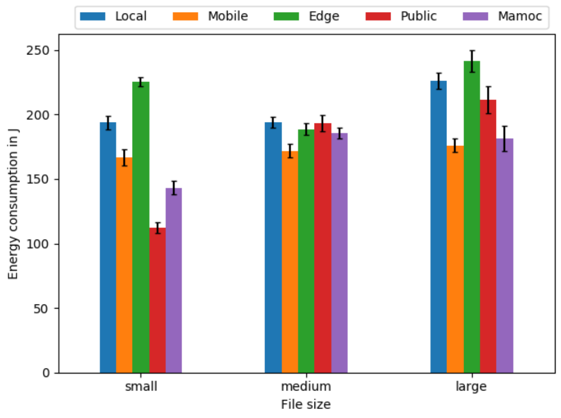
[12]https://github.com/docker/for-mac
[13]https://www.cloudping.info/

Table II: Experimental Environment Device Specifications

Node	CPU (in GHz)	RAM (in GB)	OS	Location
Nexus 7	1.3 (2-core)	1	Android 6.0 (Marshamallow)	St Andrews, UK
Pixel	2.15 (4-core)	2	Android 9.0 (Pie)	St Andrews, UK
Macbook Pro	2.5 (4-core)	16	Mac OS 10.13	St Andrews, UK
c5.4xlarge	3.0 (16-core)	64	Ubuntu Server 18.04 LTS	London, UK

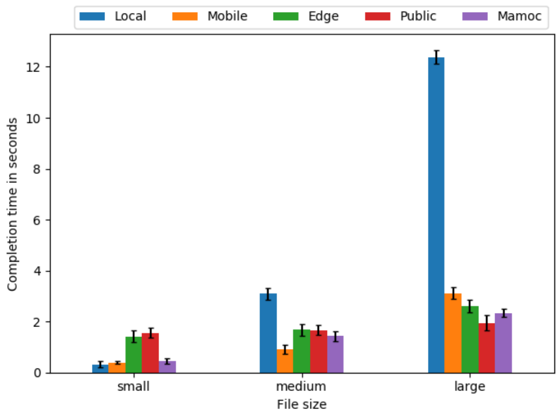


(a) Completion time of text search

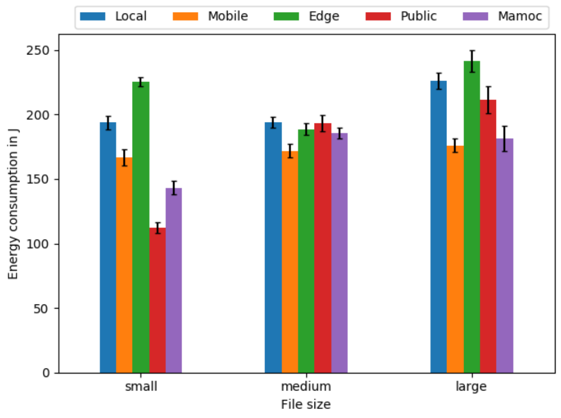


(b) Energy consumption of text search

Figure 2: Completion time and energy consumption for text search application



(a) Completion time of quick sort



(b) Energy consumption of quick sort

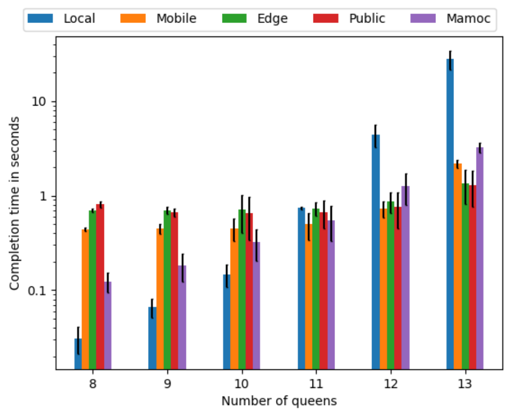
Figure 3: Completion time and energy consumption for quick sort application

file contains 316,323 words, and the small text file contains 39,799 words.

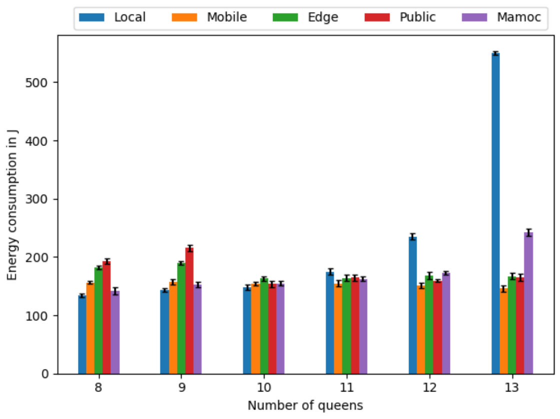
C. Evaluation

We evaluate the gains in response time and energy obtained by using MAMoC. We perform the evaluation by executing the apps locally on the device and offloading them into all available surrogates and measuring the associated response time and energy consumption. We then let our proposed offloading decision making algorithm select the surrogate for offloading. We execute each local and remote execution 30 times and calculate an average.

1) Completion Time: Completion time of offloading task to a remote server contains communication time in wireless network and computation time in the server. As the increasing of task size, the communication time and computation time are getting larger. From Figure 2(a) we can observe that in terms of total completion time, the local execution is preferable for a small text file but not in the case of medium or large text files. The minimum completion time for medium and large text file scenarios was when the task was offloaded to nearby mobile device due to low transmission overhead and high computation capabilities. Even though, the mobile device candidate had the highest offloading score, MAMoC checks if the task has been executed previously from the database entries of that task and its configurations. In this step, a simple heuristic is applied. If the method has been executed for 5 times in a row in the same location, e.g. edge, then MAMoC decides to run it on the



(a) Completion time of nqueens



(b) Energy consumption of nqueens

Figure 4: Completion time and energy consumption for nqueens application

other site, e.g. locally. By doing this, we have a mechanism to compare the executions and figure out if the task performs better locally for small input values and if for big input values it is more convenient to offload its execution on the remote site. Similarly, in Quicksort example in Figure 3(a), we can note that the edge device and public cloud instance have shorter completion times . We see a similar pattern in the N-Queens example in Figure 4(a).

2) Energy Consumption: There are essentially two methodologies for measuring the energy consumption of mobile devices: hardware and software solutions [23]. Monsoon [14] is a well-known power monitor used in many mobile computation offloading systems. There are many software based mobile device power modeling and analysis tools [24][25][26] that are used in the mobile computation offloading literature. Trepn Profiler [15] is an on device standalone profiling tool which displays an overlay UI with real-time graphs for CPU loads and battery data. App specific power consumption and utilization can be saved in a CSV file in the mobile device. The file can then be exported to a desktop computer for offline analysis. We generated the figures presented in this paper using this approach. One of the concerns of this profiler is that it only works on Snapdragon chipset-based Android devices powered with special component-wise sense resistors and power management IC. In a survey on software energy profilers [27], it is shown that Trepn profiler can achieve up to 99% accuracy against the power measurement results with the external devices.

Figures 2(b), 3(b), and 4(b) depict the energy consumption variance with the task size. It is observed that our proposed scheme outperforms local computing and full offloading to the public cloud server. In the meantime, the offloading method gets a better result especially when the task size becomes larger. Therefore, for large computing tasks, our method prefers to offload large partial computation tasks to the more powerful surrogates to reduce mobile consumption. The full offloading method is expected to have better performance than local computing on energy consumption. Our approach considers the trade-off between advantages of local computing and full offloading methods, hence our scheme reduces energy consumption in total.

It can also be noted that in some scenario executions, MAMoC has a longer average completion time and energy consumption than a particular offloading site. Nonetheless in the long run, it can adapt to the dynamic environment changes and make better decision making than always selecting local execution or full offloading to a single site.

V. Conclusions and Future Work

In this paper, we presented an adaptive multisite offloading framework that takes into consideration dynamic context changes in mobile cloud environment and offloads computation to multiple offloadees including nearby mobile devices, edge devices, and remote cloud servers. We designed and developed an Android library for mobile devices and a service provider component. We evaluated the proposed framework, and results showed that it can provide suitable offloading decisions based on the current context of the local device and past local and remote executions. We developed three demo applications and conducted experiments on different offloading scenarios and measure their completion time and energy consumption.

Our future work includes adding support for offloading intensive components of an application by performing dynamic application partitioning both locally in the mobile device and in the server side. In the client, the offloading framework implements automatic annotation by using the profiler to collect the necessary information and annotate the relevant component in the application as an indication of availability of partitioning. In the server, we will be using static analysis tools and binary code instrumentation to generate a method call graph of the application to identify the offloadable tasks. Moreover, for our current evaluation of MAMoC, we statically

[14]https://www.msoon.com/
[15]https://developer.qualcomm.com/software/trepn-power-profiler

assign both *maxLocalExecutions* and *maxRemoteExecutions* in our offloading decision making algorithm to 5. The maximum consecutive execution values could be updated dynamically through learning mechanisms e.g. using techniques from Reinforcement Learning.

VI. References

[1] Cisco. (2015) Cisco global cloud index: Forecast and methodology, 2015?2020. [Online]. Available: http://bit.ly/25wqkDN

[2] Gartner. (2017) Gartner says 8.4 billion connected things will be in use in 2017, up 31 percent from 2016. [Online]. Available: http://www.gartner.com/newsroom/id/3598917

[3] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 1285–1293.

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

[5] U. Drolia, R. Martins, J. Tan, A. Chheda, M. Sanghavi, R. Gandhi, and P. Narasimhan, "The case for mobile edge-clouds," in *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, Dec 2013, pp. 209–215.

[6] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814441

[7] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 301–314. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966473

[8] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, April 2010.

[9] B. Zhou and R. Buyya, "Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions," *ACM Comput. Surv.*, vol. 51, pp. 13:1–13:38, 2018.

[10] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, ser. MCS 12. New York, NY, USA: ACM, 2012, pp. 29–36. [Online]. Available: http://doi.acm.org/10.1145/2307849.2307858

[11] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '12. New York, NY, USA: ACM, 2012, pp. 145–154. [Online]. Available: http://doi.acm.org/10.1145/2248371.2248394

[12] A. Mtibaa, K. A. Harras, and A. Fahim, "Towards computational offloading in mobile device clouds," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 1, Dec 2013, pp. 331–338.

[13] N. Fernando, S. W. Loke, and W. Rahayu, "Computing with nearby mobile devices: a work sharing algorithm for mobile edge-clouds," *IEEE Transactions on Cloud Computing*, 2016.

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

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

[16] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "Cosmos: Computation offloading as a service for mobile devices," in *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '14. New York, NY, USA: ACM, 2014, pp. 287–296. [Online]. Available: http://doi.acm.org/10.1145/2632951.2632958

[17] J. L. D. Neto, S. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "Uloof: A user level online offloading framework for mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2660–2674, Nov 2018.

[18] S. Wu, C. Niu, J. Rao, H. Jin, and X. Dai, "Container-based cloud platform for mobile computation offloading," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2017, pp. 123–132.

[19] S. Wu, C. Mei, H. Jin, and D. Wang, "Android unikernel: Gearing mobile code offloading towards edge computing," *Future Generation Computer Systems*, 2018.

[20] N. Vafaei, R. A. Ribeiro, and L. M. Camarinha-Matos, "Normalization techniques for multi-criteria decision making: Analytical hierarchy process case study," in *Technological Innovation for Cyber-Physical Systems*. Cham: Springer International Publishing, 2016, pp. 261–269.

[21] D. Sulaiman and A. Barker, "Mamoc: Multisite adaptive offloading framework for mobile cloud applications," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec 2017, pp. 17–24.

[22] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt, "Fast pattern matching in strings," *SIAM journal on computing*, vol. 6, no. 2, pp. 323–350, 1977.

[23] S. Hao, D. Li, W. G. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 92–101.

[24] Google. Profile battery usage with batterystats and battery historian. [Online]. Available: https://developer.android.com/studio/profile/battery-historian

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

[26] D. D. Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. D. Lucia, "Software-based energy profiling of android apps: Simple, efficient and reliable?" in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb 2017, pp. 103–114.

[27] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma, "Modeling, profiling, and debugging the energy consumption of mobile devices," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, p. 39, 2016.