Developing Offloading-enabled Application Development Frameworks for Android Mobile Devices

Hui Xia
School of Information Science and Engineering
Hunan University
ChangSha, China
xiahui@hnu.edu.cn

Ligang He[1]
Department of Computer Science
University of Warwick
Coventry, United Kingdom
ligang.he@warwick.ac.uk

Bin Wang
School of Information Science and Engineering
Hunan University
ChangSha, China
binvang@hnu.edu.cn

Cheng Chang
School of Information Science and Engineering
Hunan University
ChangSha, China
chengchangx@gmail.com

Xie Han
School of Electronics and Computer Science and Technology
North University of China
TaiYuan, China
hanxie@nuc.edu.cn

Carsten Maple
WMG
University of Warwick
Coventry, United Kingdom
CM@warwick.ac.uk

***Abstract*—Mobile devices, such as smartphones, offer people great convenience in accessing information and computation resources. However, mobile devices remain relatively limited in terms of computing, memory and energy capacity when compared with desktop machines. A promising solution to mitigate these limitations is to enhance the services mobile devices can provide by utilizing powerful cloud platforms through offloading mechanisms, i.e., offloading the heavy information processing tasks from mobile devices to the Cloud. This paper addresses this issue by developing two offloading-enabled application development frameworks by adapting certain Android OS interfaces. The applications developed using these frameworks will be equipped with offloading capability. In the first framework, each application is selfish and makes offloading decisions independently, whereas in the second, a central offloading manager resides in the mobile device and is responsible for making the offloading decisions for all applications. The two frameworks are designed in a way that application developers only need to make minimal changes to their programming behavior. Experiments have been conducted that verify the feasibility and effectiveness of the offloading mechanisms that are proposed.**

Keywords—mobile computing, computational offloading, information processing

I. INTRODUCTION

Mobile devices, such as smartphones with microprocessor, memory, Internet access and various applications offer people great convenience in accessing the information and using computational resources. However, mobile devices are still limited in computing, memory and energy capacity, compared with desktop servers. A promising solution to mitigate the limitations is to integrate the mobile devices with powerful cloud platforms through offloading mechanisms, i.e., offloading the heavy information processing tasks from mobile devices to the Cloud [1]. For this purpose, this paper develops two offloading mechanisms: Selfish Offloading Mechanism (SOM) and Global Offloading Mechanism (GOM). SOM is utilized by each individual application on the mobile, with each application assuming it has exclusive usage of the resources in the mobile device, making offloading decisions independently. In contrast, GOM is adopted by the mobile device and acts as a central offloading manager for all applications.

These two offloading mechanisms are developed for mobile devices with the Android Operating System [2] (OS) and adapt certain functions in Android OS. Application developers can integrate SOM and GOM allowing applications developed to have an offloading capability. Although developers need to change their programming behaviors to some extent, SOM and GOM are designed in the way that the changes in the application development procedure is minimal. In particular, SOM offers a Java Class to encapsulate the offloading, while GOM is packaged as a separate application that make the offloading decisions for all applications in the device.

Overall, this paper make the following contributions: i) we design a Selfish Offloading Mechanism and implement as a Java class; ii) we design a Global Offloading Mechanism and implement as an independent application; iii) we demonstrate that SOM and GOM are effective by conducting experiments on a real cloud platform.

The rest of this paper is organized as follows. Section 2 presents related work, before the design of SOM and GOM are presented in Section 3. Section 4 presents the evaluation of experiments and the paper is concluded in Sections 5.

II. RELATED WORK

There are the existing research studies [3,4,5] to investigate offloading strategies and frameworks. The research on the first aspect mainly focuses on the theoretical analysis of better offloading strategies [6,7,8,9], while the research on the latter aspect focuses on developing the application development frameworks that enable the applications to have an offloading capability.

Aimed at energy management, a compile-time framework supporting remote task execution was first introduced in [10]. Based on the same approach, a more detailed cost graph was used in [11], with a parametric analysis on its effect at runtime presented in [12]. Another compiler-assisted approach was introduced in [13], which turns the focus to reducing the application's overall execution

[1]Corresponding author.

time. Spectra, [14], adds application fidelity (a run-time QoS measurement) into the decision making process and uses it to leverage execution time and energy usage in its utility function. Spectra monitors the hardware environment at run-time and choose between programmer pre-defined execution plans. Chroma [15] builds on Spectra but constructs the utility function externally in a more automated fashion. The offload decision engine applies an integer programming technique to produce allocation schemes. Aimed at reducing the communication costs, [16] proposes the concept of cloudlets, which bring the distant Cloud to the more commonly accessible WiFi hotspots. A dynamic VM synthesis approach is proposed in [16].

Cuckoo [17] integrates the popular open source Android framework with the Eclipse development tools [18]. It provides a simple programming model, allowing an interface for implementing the computation tasks to be executed locally in the mobile device and remotely in the Cloud; Cuckoo makes the offloading decisions at runtime. It is also equipped with a common remote server, which is used to run the computation-intensive services offloaded from the mobile.

MAUI [19] provides a programming environment in which developers use a "remoteable" keyword to declare the methods that can be run remotely in the Cloud. MAUI uses the four features of the .NET common language runtime to perform computation offloading. It uses code portability to produce two versions of smartphone applications, one running locally and the other running on a remote server. In addition, MAUI uses the type security feature to send the application state to the remote execution method. Each method of the application is analyzed and the serialization characteristics are used to determine its network transportation costs.

III. Design and Implementation of SOM and GOM

A. *The underlying mechanisms and functions for developing SOM and GOM*

There are four types of components in an Android application: Activity, Service, Content Provider, and Broadcast Receiver. Of these, Activity and Service are the two components that relate to this work. The Activity component provides an interface that interacts with the users, e.g., takes data input by users. The Service component runs in the background, performing the actual computations of an application.

When a user launches an application on a mobile device, it invokes an Activity component that provides the user's graphical interface. The Activity component then binds to the service currently running or start a new service. Services can be shared between multiple activities. Once the Activity is bound to the running service, it can invoke service operations through Inter-Process Communication (IPC). In particular, the Activity issues a request to invoke the Service. The request is passed to the proxy subcomponent residing within the Activity. The request is further passed, by the Android kernel, to the stub sub-component within the Service. Finally, the stub invokes the actual service operations implemented in the Service component.

The underlying IPC mechanism in Android is implemented through the IBinder interface. IBinder is a base interface for calling a remote object and the core part of the lightweight remote procedure call mechanism. This interface describes the abstract protocol for interacting with a remote object. The frameworks developed in this work do not implement this interface directly, but rather extend it. The key method in the IBinder interface is transact(), which allows the programmer to send a call to an Ibinder object. The transact() method is matched by the Binder.onTransact() method in the Binder interface in the Service component, i.e., the call sent by transact() will be received by onTransact().

B. *The Design of SOM*

The architecture and the execution flow of the SOM is illustrated in Figure 1. In SOM, the MainActivity method calls the proxy. In the proxy, XBinder, which extends from IBinder, is invoked. The internal structure of XBinder is illustrated in Figure 2. The offloading decision (by calling the makeDecision method shown in Figure 2) is made inXBinder. If the offloading decision is yes, the service implemented in the Cloud is invoked (by invoking the service specified by "http" in Figure 2). Otherwise, the local service is called (by calling iBinder shown in Figure 2), in which the stub in the Service is called and then the local service operation (KMeansService in this example) is called.

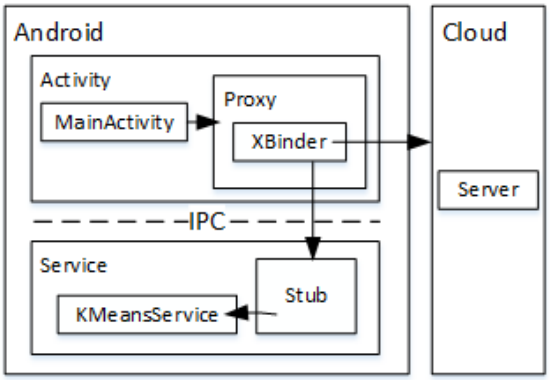

Fig. 1. The architecture of SOM

The internal code of XBinder is outlined in Figure 2. Line 1 shows that XBinder extends IBinder (Line 1).The original transact method is overwritten to insert the function of making offloading decisions (Line 13-19).

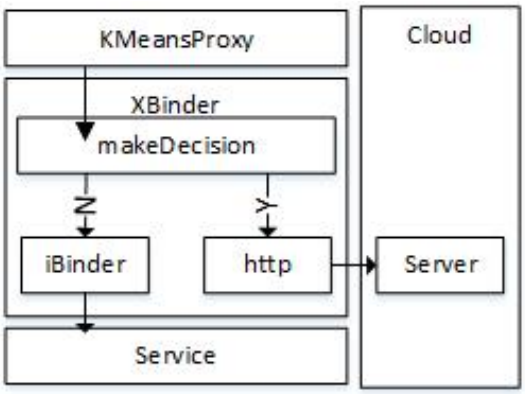

Fig. 2. Internal structure of XBinder in SOM

```
1  public class XBinder implements IBinder {
2    private IBinder iBinder = null;
3
4    public XBinder(IBinder iBinder) {
5      this.iBinder = iBinder;
6    }
7
8    private boolean makeDecision(long bitmapSize) {
9      // Code for computation offloading decision
10   }
11
12   @Override
13   public boolean transact(int code, Parcel data, Parcel reply, int flags)
14     throws RemoteException {
15     if(makeDecision(bitmapSize)) {
16       // Cloud server execution
17     } else {
18       // Local service execution
19   }
20
21   @Override
22   // Other override method
23 }
```

Fig. 3. Internal code of XBinder in SOM

Using SOM, the application developer needs to change his programming behavior to some extent. All changes are in the proxy code. When the proxy subcomponent in the Activity calls the stub in the Service, the developer uses the Java Class provided by SOM (i.e., XBinder) instead of the default Java Class iBinder. XBinder implements Android's IBinder and override its methods. The function for making offloading decisions and the specific offloaded operations are implemented in in the transact method of XBinder. Figure 4 gives an example to show the program segments that are different from the default program. Figure 4a is the default program while 4b is the program that is coded utilizing SOM. There are 3 changes to the default way of writing the proxy code: i) using our own Java Class XBinder to declare the variable instead of using Android's own default Java Class iBinder (Line 2); ii) using the constructor of XBinder to take in the variables passed from the outside (the default iBinder) (Line 4-6); iii) using the transact method of XBinder to perform IPC (Line 16).

C. The Design of GOM

SOM is used to enable an individual application to have offloading capability. Each application makes the offloading decisions independently, assuming it has the exclusive use of resources in the mobile device. In contrast, GOM is used to develop an application (which we call the Offloading Manager) that makes offloading decisions for all applications on the mobile device. In GOM, all applications send their own data to the offloading manager, which can make a device-wide optimal offloading decision based on the data collected, such as the global status of the resources and other variables in the mobile device. The architecture of GOM is shown in Figure 5. In this case the operation is different from SOM, and XBinder in the proxy of every application calls the service (DecisionService) in the offloading manager to make offloading decisions.

```
1  public class KmeansProxy implements IKmeans {
2    private IBinder iBinder;
3
4    public KmeansProxy(IBinder iBinder) {
5      this.iBinder = iBinder;
6    }
7
8    @Override
9    public Bitmap getResultGraphics(Bitmap bitmapData, int k, int m) {
10     Parcel data = Parcel.obtain();
11     Parcel reply = Parcel.obtain();
12     data.writeParcelable(bitmapData, 0);
13     data.writeInt(k);
14     data.writeInt(m);
15     try {
16       iBinder.transact(1, data, reply, 0);
17       return reply.readParcelable(null);
18     } catch (RemoteException e) {
19       e.printStackTrace();
20     }
21     return null;
22   }
23 }
```

(a)

```
1  public class KmeansProxy implements IKmeans {
2    private XBinder xBinder;
3
4    public KmeansProxy(IBinder iBinder) {
5      this.xBinder = new XBinder(iBinder);
6    }
7
8    @Override
9    public Bitmap getResultGraphics(Bitmap bitmapData, int k, int m) {
10     Parcel data = Parcel.obtain();
11     Parcel reply = Parcel.obtain();
12     data.writeParcelable(bitmapData, 0);
13     data.writeInt(k);
14     data.writeInt(k);
15     try {
16       xBinder.transact(1, data, reply, 0);
17       return reply.readParcelable(null);
18     } catch (RemoteException e) {
19       e.printStackTrace();
20     }
21     return null;
22   }
23 }
```

(b)

Fig. 4. The comparison between the default programming behaviors and those in SOM

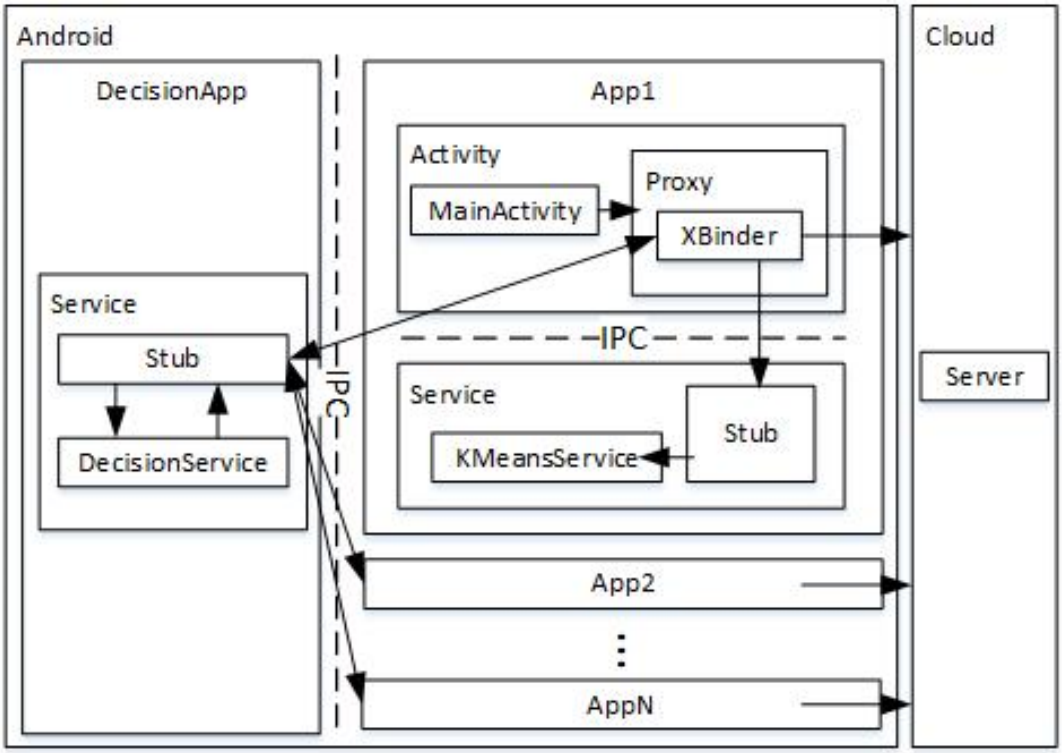

Fig. 5. The architecture of GOM

```
1  public class XBinder implements IBinder {
2     private IBinder iBinder = null;
3     private IDecision iDecision = null;
4
5     public XBinder(IBinder iBinder) {
6        this.iBinder = iBinder;
7     }
8
9     @Override
10    public boolean transact(int code, Parcel data, Parcel reply, int flags)
11       throws RemoteException {
12       if(iDecision.makeDecision(bitmapSize)) {
13          // Cloud server execution
14       } else {
15          // Local service execution
16    }
17
18    @Override
19    // Other override method
20 }
```

Fig. 6. The code structure of the XBinder in GOM

The application developer also needs to change the default way of programming to use GOM. As with SOM, the developers need to use XBinder to provide IPC communications in GOM. In addition, the developers need to use the other Java Class provided in GOM to bind to the offloading decision application. The XBinder in GOM is more complex than the XBinder in SOM. XBinder in GOM also needs to pass the IBinder interface object. In the transact method, the decision making algorithms are invoked to use the IBinder interface to make decisions. The code in the GOM XBinder is outlined in Figure 6.

As can be seen from Figure 6, The IDecision variable is created in the XBinder constructor (Line 3). The iDecision variable refers to the Service in the offloading manager application. The makeDecision method in iDecision is called to make offloading decisions (Line 12).

IV. Experiments

In this section we use a smartphone application to showcase the applicability and effectiveness of SOM and GOM. We measure the energy consumption and execution time of the mobile application in both native execution and offloading-enabled execution.

A. The Offloading Strategy

The purpose of this work is to develop an application development framework to enable the applications to have the offloading ability, not to develop better offloading strategies. In the exemplar application, we adopt an existing offloading strategy from the literature [20, 21], as shown in Eq. 1, where C is Computation size; D is Communication size; M is the Computation speed of the mobile device; Pc is the energy consumed for computation in a time unit; Ptr is energy consumed for communication in a time unit; Pi is energy used when idle for a time unit; S is the computation speed of the cloud server; B is the network bandwidth. The values of all these parameters can be benchmarked. When the output of the equation, which is the energy saved when the computation is offloaded, is positive, it is regarded beneficial to offload the computation to the cloud. Otherwise, the computation is run locally on the mobile device.

$$Energy\ Saving = P_r \times \frac{C}{M} - \left(P_i \times \frac{C}{S} + P_{tr} \times \frac{D}{B}\right) \quad (1)$$

B. Mobile Device and Cloud Resources

In the experiment, the smartphone we use is a Huawei Honor 8. The following is the specification of this smartphone: Android OS version 7.0, the Hisilicon Kirin 950 processor with 4GB memory.

There are a number of cloud service providers such as Ali cloud, Baidu cloud, Amazon cloud [22], Qing cloud [23] and others. In our experiment, the Qing cloud is used as the cloud server. We rented a server in the Qing cloud, and created a router and switch, and acquired a public network IP and other resources. Figure 7 shows the main resources we rented on the Qing cloud and its cost. The specification of the cloud server is as follows: Ubuntu Server 16.04 LTS 64bit, dual core, 2G memory. Figure 8 lists the specification of the Cloud server.

Resource	Total:¥ 122.7599	
	Count	Total Consumption(¥)
Instance	2	58.3065
Image	0	0
Router/VPC Network	3	40.0977
Load Balancer	0	0
EIP	2	24.3557

Fig. 7. The resources rented in the Qing cloud in the experiment

We use the Trepn Profiler to test the power consumption of smartphone in order to calculate the energy consumption of the phone. Figure 8 shows the result of the Trepn Profiler.

Configuration	
Image	Ubuntu Server 16.04 LTS 64bit
Instance Class	High Performance
CPU Count	2
CPU Topology	2.1.1
CPU Model	Default
Memory	2G
Keypair	(ssh1)

Fig. 8. The server configuration in Qing cloud

C. Mobile application in the experiment

The mobile application we use in the experiments is an image segmentation application based on the K-means clustering algorithm. The main steps of the algorithm are as follows:

Input: the number of clusters k, the number of iterations m and the input image.

Output: A segmented image.

1: Select the initial center of k clusters;

2: For a sample, find the distance to the k clusters and place the sample in the cluster with the shortest distance;

3: Using the mean and other methods to update the center of the clusters;

4: Repeat Step 2 and 3 for m times.

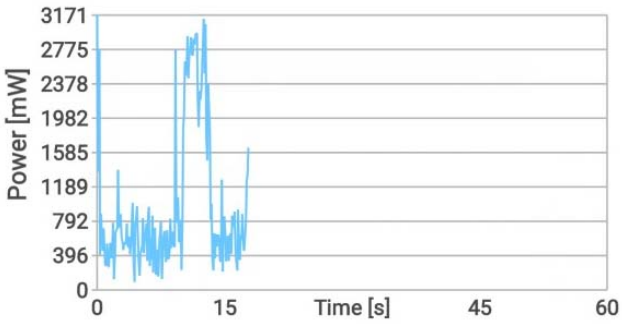

Fig. 9. Testing the power consumption with the Trepn Profiler

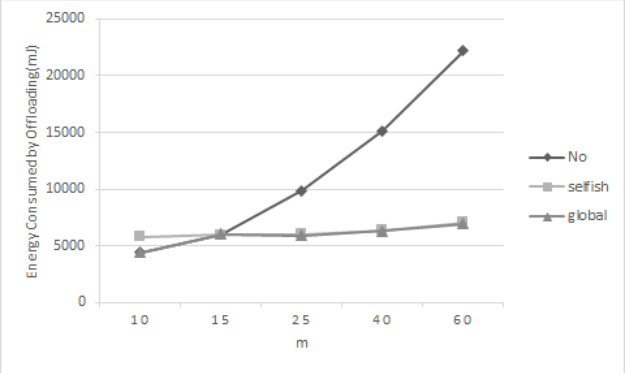

Fig. 10. The energy consumption of the native App, application developed by SOM and application by GOM

We developed the offloading-enabled image segmentation application using SOM and GOM. We tested the execution time and power consumption in three scenarios: i) native application, ii) application developed with SOM and iii) application with GOM.

D. Experimental Results

Figure 10 shows the energy consumption of the applications. It can be seen from Figure 10 that compared with the native application, SOM and GOM reduce the energy consumption when m is greater than 15. The gap increases as the number of iterations (m, i.e., the problem size) increases. This is because as m increases it becomes increasingly better off to offload the computation to the Cloud for energy savings. The reason why SOM is worse than the native application when m is less than 15 is because with SOM each application thinks it monopolizes the usage of resources in the mobile device. However, it is not the case in reality. So when all applications decide to offload the computation to the cloud, they compete for the

communication bandwidth. As a result, offloading is an unfavorable decision in reality for such situations. On the other hand, the application by GOM always makes the best decision. Another observation from the figure is that when m becomes bigger (than 15 in the figure), the energy consumed by the application with SOM is almost the same as that by GOM. This is because when m is big, offloading is always a better decision, no matter whether making a global decision or individual decision.

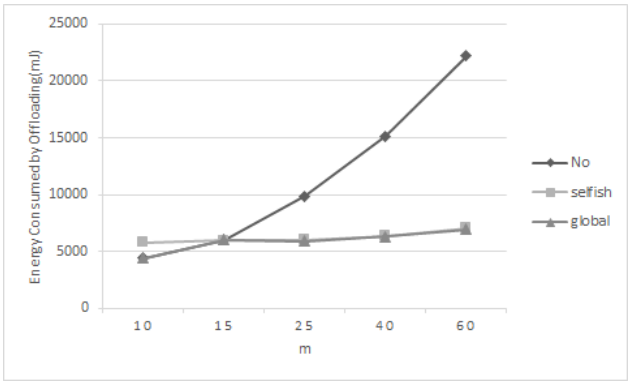

Fig. 11. The execution time of the native application, application developed with SOM and application with GOM

Figure 11 shows the execution time of the applications. The first observation from Figure 11 is that the native application achieves the shortest execution time among all three applications. This is because the objective function for the offloading decision used in GOM and SOM is energy saved. Therefore, GOM and SOM will offload the computations to the cloud as long as energy can be saved, even if this will increase the execution time. If this is an undesirable situation, it can be solved by adjusting the offloading strategy (e.g, also taking the execution time into account when making offloading decisions). When m becomes big (greater than 40 in this figure), the applications developed by SOM and GOM also have shorter execution times.

V. Conclusion

In this paper, we have developed two offloading-enable application frameworks, SOM and GOM, for Android mobile devices. With SOM, each application can make individual offloading decisions; with GOM, an application is developed as a central offloading manager, which makes offloading decisions for all applications on the mobile device. In these frameworks, a new Java class, XBinder, extends the original class IBinder in the Android OS. The two frameworks are tested with a real mobile application and on a real Cloud platform called Qing.

Acknowledgment

This work is partially sponsored by Shandong Worldwide Byte Security Information Technology, Co., Ltd, and by PETRAS, the UK Research Hub for Cyber Security of the Internet of Things, through the EPSRC Grant EP/N02334X/1.

References

[1] Kumar, K., Lu, Y.-H.: Cloud computing for mobile users. Computer 99 (2010)

[2] Android, http://developer.android.com/

[3] Chun, B.-G., Maniatis, P.: Augmented smart phone applications through clone cloud execution. In: Proceedings of the 12th Workshop on Hot Topics in Operating

[4] Kemp, R., Palmer, N., Kielmann, T., Seinstra, F., Drost, N., Maassen, J., Bal, H.E.:eyeDentify: Multimedia Cyber Foraging from a Smartphone. In: IEEE International Symposium on Multimedia (2009)

[5] C. Shin, J.-H. Hong, and A. K. Dey, ‘‘Understanding and prediction of mobile application usage for smart phones,’’ in Proc. ACM Conf. Ubiquitous Comput. (UbiComp), 2012, pp. 173–182

[6] B. Gao, L. He, "Modelling Energy-Aware Task Allocation in Mobile Workflows", The 10th Annual International Conference on in Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous), 2013

[7] B. Gao, L. He and C. Chen, "Modelling the Bandwidth Allocation Problem in Mobile Service-Oriented Networks", The 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'15), Cancun, Mexico, November 2-6, 2015

[8] B. Gao, L. He, L. Liu, K. Li and S. Jarvis, "From Mobiles to Clouds: Developing Energy-aware offloading Strategies for Workflows", The 13th IEEE/ACM International Conference on Grid Computing (Grid 2012)

[9] Bo Gao, Ligang He and S. A. Jarvis, "Offload Decision Models and the Price of Anarchy in Mobile Cloud Application Ecosystems," in IEEE Access, vol. 3, no. , pp. 3125-3137, 2015. doi: 10.1109/ACCESS.2016.2518179

[10] U. Kremer, J. Hicks, and J. M. Rehg, ‘‘Compiler-directed remote task execution for power management,’’ in Proc. Workshop Compil. Oper. Syst. Low Power (COLP), 2000, pp. 1–8.

[11] Z. Li, C. Wang, and R. Xu, ‘‘Computation offloading to save energy on handheld devices: A partition scheme,’’ in Proc. Int. Conf. Compil., Archit., Synthesis Embedded Syst. (CASES), 2001, pp. 238–246.

[12] C. Wang and Z. Li, ‘‘Parametric analysis for adaptive computation offloading,’’ in Proc. ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI), New York, NY, USA, Jun. 2004, pp. 119–130.

[13] S. Kim, H. Rim, and H. Han, ‘‘Distributed execution for resource constrained mobile consumer devices,’’ IEEE Trans. Consum. Electron.,vol. 55, no. 2, pp. 376–384, May 2009.

[14] J. Flinn and M. Satyanarayanan, ‘‘Balancing performance, energy, and quality in pervasive computing,’’ in Proc. 22nd Int. Conf. Distrib. Comput. Syst., 2002, pp. 217–226.

[15] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi, ‘‘Tactics-based remote execution for mobile computing,’’ in Proc. 1st Int. Conf. Mobile Syst., Appl. Services (MobiSys), New York, NY, USA, 2003, pp. 273–286.

[16] M. Satyanarayanan, P. Bahl, R. CÆceres, and N. Davies, ‘‘The case for VM-based cloudlets in mobile computing,’’ IEEE Pervasive Comput., vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009.

[17] Kemp R, Palmer N, Kielmann T, Bal H (2012) Cuckoo: a computation offloading framework for smartphones. In: Mobile computing application and service, vol 76 of LNCS. Springer, pp 59–79

[18] Eclipse, http://www.eclipse.org/

[19] E. Cuervo et al., ‘‘MAUI: Making smartphones last longer with code offload,’’ in Proc. 8th Int. Conf. Mobile Syst., Appl., Services (MobiSys), Jun. 2010, pp. 46–62.

[20] M. A. Khan, ‘‘A survey of computation offloading strategies for performance improvement of applications running on mobile devices,’’ J. Netw. Comput. Appl., vol. 56, pp. 28–40, Oct. 2015.

[21] C. Chekuri and M. Bender, ‘‘An efficient approximation algorithm for minimizing makespan on uniformly related machines,’’ in Proc. 6th Conf. Integer Program. Combinat. Optim.,1998, pp. 383–393.

[22] Amazon Elastic Computing, http://aws.amazon.com/ec2/

[23] QingCloud, https://www.qingcloud.com/