2012 10th IEEE International Symposium on Parallel and Distributed Processing with Applications

Adaptive Computation Offloading from Mobile Devices into the Cloud

Dejan Kovachev, Tian Yu and Ralf Klamma
Information Systems and Database Technologies
RWTH Aachen University
Ahornstr. 55, 52056 Aachen Germany
{kovachev|tianyu|klamma}@dbis.rwth-aachen.de

***Abstract*—The inherently limited processing power and battery lifetime of mobile phones hinder the possible execution of computationally intensive applications like content-based video analysis or 3D modeling. Offloading of computationally intensive application parts from the mobile platform into a remote cloud infrastructure or nearby idle computers addresses this problem. This paper presents our Mobile Augmentation Cloud Services (MACS) middleware which enables adaptive extension of Android application execution from a mobile client into the cloud. Applications are developed by using the standard Android development pattern. The middleware does the heavy lifting of adaptive application partitioning, resource monitoring and computation offloading. These elastic mobile applications can run as usual mobile application, but they can also reach transparently remote computing resources. Two prototype applications using the MACS middleware demonstrate the benefits of the approach. The evaluation shows that applications, which involve complicated algorithms and large computations, can benefit from offloading with around 95% energy savings and significant performance gains compared to local execution only.**

I. Introduction

Resource-demanding multimedia applications such as 3D video games are being increasingly demanded on mobile phones. Even if mobile devices' hardware and mobile networks continue to evolve and to improve, mobile devices will always be resource-poor, less secure, with unstable connectivity, and with constrained energy. Resource poverty is major obstacle for many applications [1]. Therefore, computation on mobile devices will always involve a compromise. For example, on-the-fly editing of video clips on a mobile phone is prohibited by the energy and time consumption. Same performance and functionalities on mobile devices still cannot be obtained as on their desktop PCs' or even notebooks' when dealing with tasks containing resource-demanding tasks.

Recently, the combination of cloud computing [2], wireless communication infrastructure, ubiquitous computing devices, location-based services, mobile Web, etc., has laid the foundation for a novel computing model, called *mobile cloud computing* [3]. It provides to users an online access to unlimited computing power and storage space. The cloud abstracts the complexities of provisioning computation and storage infrastructure, which the end user uses them as utility and in reality they can be far away data center or nearby idle hardware.

Offloading has gained big attention in mobile cloud computing research, because it has similar aims as the emerging cloud computing paradigm, i.e. to surmount mobile devices' shortcomings by augmenting their capabilities with external resources. Offloading or augmented execution refers to a technique used to overcome the limitations of mobile phones in terms of computation, memory and battery. Such applications, which can adaptively be split and parts offloaded [4], [5], are called *elastic mobile applications*. Basically, this model of elastic mobile applications enable the developers the illusion as if he/she is programming virtually much more powerful mobile devices than the actual capacities. Moreover, elastic mobile application can run as stand-alone mobile application but also use external resources adaptively. Which portions of the application are executed remotely is decided at runtime based on resource availability. In contrast, client/server applications have static partitioning of code, data and business logic between the server and client, which is done in development phase.

Our contributions include an integration with the established Android application model for development of "offloadable" applications, a lightweight application partitioning and a mechanism for seamless adaptive computation offloading. We propose Mobile Augmentation Cloud Services (MACS), a services-based mobile cloud computing middleware. Android applications that use the MACS middleware benefit from seamless offloading of computation-intensive parts of the application into nearby or remote clouds. First, from developer's perspective, the application model stays the same as on the Android platform. The only requirement is that computation-intensive parts are developed as Android services, each of which encapsulates specific functionality. Second, according to different conditions/parameters, the modules of program are divided into two groups; one group runs locally, the other group is runs on cloud side. The decision for partitioning is done as an optimization problem according to the input on the conditions of cloud side and devices, such as CPU load, available memory, remaining battery power on devices, bandwidth between the cloud and devices. Third, based on the solution of the optimization problem, our middleware offloads parts to the remote clouds and returns the corresponding results back. Two Android applications on top of MACS demonstrate the potential of our approach.

978-0-7695-4701-5/12 $26.00 © 2012 IEEE
DOI 10.1109/ISPA.2012.115

In the rest of the paper, we first review related research work mobile cloud computing in Section II. Then we describe our MACS middleware with detailed descriptions of the implementation (Section III). We explain the offloading model in our middleware in Section IV. In Section V we introduce the two use case applications, the setup of the evaluation and the corresponding evaluation results based on those two applications. After that, we discuss about the results in Section VI. Finally, draw conclusions and refer to the future work.

II. Related Work

Previous work has proposed many mechanisms that address the challenges of seamless offloaded execution from a device to a computational infrastructure (cloud). A "brute force" approach to offloading can be considered the encapsulation of mobile device's software stack into a virtual machine image and executing it on a more powerful hardware, such as proposed by Chun and Maniatis [6] or Satyanarayanan et al. [1]. More recently, Kosta et al. [7] have further improved this idea. Although such virtualized offloading can be considered as simple and general solution, it lacks flexibility and control over offloadable components. Therefore, we consider that application developers can better organize their application logic using the established Android service design patterns and benefit from the MACS middleware.

Ou et al. [8] propose class instrumenting technique, i.e. a process to transform code classes into a form which is suitable for remote execution. Two new classes are generated from the original class, one is an instrumented class which has the real implementation and the same functionality as the original class, the other is a proxy class, whose responsibility is only to call the function written in the instrumented class. Then, the instrumented class can be offloaded to remote cloud, and the call will be invoked from the instrumented in the remote cloud. In MACS we adopt similar idea, but unlike Ou et al. [8] we use standardized language for the proxy interfaces which is already widespread in the Android platform. The Cuckoo framework [4] and MAUI system in [9] implement a similar idea. Our MACS middleware is inspired by these solutions. However, MACS middleware does extra profiling and resource monitoring of applications and adapts the paritioning decision at runtime.

An important challenge in partitioned elastic applications is how to determine which parts of code should be pushed to the remote clouds. The graph based approach to model the application has been used in several works. Giurgiu et al. [10] use "consumption" graph and decide which part should be running locally or remotely by finding a cut of the consumption graph with a goal function, which minimizes the total sum of communication cost, transmitting cost and the cost of building local proxies. The AIDE platform [11] uses a component-based offloading algorithm, which mainly focuses on minimum historical transmission between two partitions. The $(k+1)$ partitioning algorithm, introduced by Ou et al. [8], is applied to a multi-cost graph to represent the class-based components. A similar approach is done by Gu et al. [11], [12]. Zhang et al. [13], [14] use a general Bayesian inference to make the partitioning decision. However, executing constantly executing graph or inference algorithms on the mobile device takes significant resources on the constrained device. We use integer linear optimization model to describe the offloading so that it is not only easy to implement, but it can also be independently solved if the remote clouds are temporarily not available.

III. Mobile Augmentation Cloud Services

The goal of our MACS middleware is to enable the execution of elastic mobile applications. Zhang et al. [14] consider elastic applications to have two distinct properties. First, an elastic application execution is divided partially on the device and partially on the cloud. Second, this division is not static, but is dynamically adjusted during application runtime. The benefits of having such application model are that the mobile applications can still run independently on mobile platforms, but can also reach cloud resources on demand and availability. Thus, mobile applications are not limited by the constraints of the existing device capacities.

MACS architecture is depicted on Figure 1. In order to use MACS middleware, the application should be structured using established Android services pattern. Android is already established as the most prominent mobile phone platform. Additionally, its application architecture model allow decomposition of applications into service components which can be shared between applications. A MACS application consists of an application core (Android activities, GUI, access to devices sensors) which can not be offloaded, and multiples services (S_i) that encapsulate separate application functionality (usually resource-demanding components) which can be offloaded (S_{Ri}). The services communicate with the application through an interface defined by the developer in the Android interface definition language (AIDL).

As service-based implementation is adopted, for each service we can profile following metadata:

- type: whether can be offloaded or not
- memory cost: the memory consumption of the service on the mobile device
- code size: size of compiled code of the service
- dependency information on other services, for each related module, we collect following:
  - transfer size: amount of data to be transferred
  - send size: amount of data to be sent
  - receive size: amount of data to be received

Metadata is obtained by monitoring the application execution and environment.

Android services are using Android interprocess communication (IPC) channels to do RPC. The services are registered in the Service Manager, and a binder maintains a handle for each service. Then an application, that wants to use a service, can query the service in the Service Manager. Upon service discovery, the Android platform will create a service proxy for the client application. All the requests to access the service will

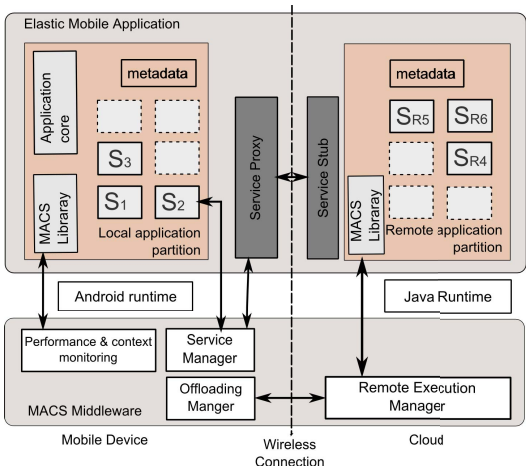


Fig. 1. MACS architecture. Application logic is structured from multiple Android services (S_i). Some of them can be offloaded into the cloud (S_{Ri}).

be sent through the service proxy, and then forwarded to the service by the binder. After processing the requests, results are sent back to the service proxy on the client application through the binder. Finally, the client gets the result from the service proxy. From client's point of view, there is no difference between calling a remote service or calling a local function.

The offload manager determines the execution plan, and then the services to be offloaded are pushed to the cloud. The results are sent back to the application upon completion. Our approach is similar to the Cuckoo framework [4], however, MACS allows dynamic application partitioning at runtime, where Cuckoo only enables static partitioning at compile time. MACS monitors the execution of the services and the environment parameters. Whenever the situation changes, the middleware can adapt the offloading and partitioning.

The main goal of MACS is to enable transparent computation offloading for mobile applications. Therefore, our middleware tries to fit the usual Android development process and bring the developers an easier way to offload parts of their applications to remote clouds in a transparent way. MACS hooks into the Android compile system, makes modifications of generated Java files from AIDL in the pre-compile stage. Developers need to include MACS SDK libraries into their Android project.

At the cloud side, the MACS middleware handles the offload requests from the clients, installs of offloaded services, their initialization and method invokes. The cloud-side MACS middleware is written with pure J2SE so that it can run on any machines with installed J2SE.

MACS middleware monitors the resources on the mobile execution environment and available clouds. It then forms an optimization problem whose solution is used to decide whether the service which contains the called function should be offloaded or not. When the service is determined to be offloaded to the remote cloud, our middleware tries first to execute the service remotely. If there is no such service on the remote clouds, our framework transmits the service code (jar file) to the cloud, and the corresponding results after the service execution are returned to the mobile device. The cloud caches the jar files for subsequent execution.

Except for the computation offloading, our framework also features simple data offloading. If files are needed to be accessed on the remote cloud, MACS file transmission (MACS-FTM) transfers automatically the non-existed files from the local device and vice versa.

IV. Adaptive Computation Offloading

The proposed model and corresponding algorithm are supposed to be applied for scenario which is computation-intensive [15], and the requirements for the developer is that the code should be structured in a model in advance.

Let us suppose that we have n number of modules which can be offloaded, $S_1, S_2...S_n$. Each of modules has several properties described as metadata, i.e. for specific module i, its memory cost mem_i, code size $code_i$. Let us consider the k number of related module which can be offloaded. For each of them, we denote the transfer size $tr_1, tr_2...tr_k$, send size $send_1, send_2...send_k$, receive size $rec_1, rec_2...rec_k$, where $\{1..k\} \subseteq \{1..n\}$ and $send_k + rec_k = tr_k$. Meanwhile, we introduce x_i for module i, which indicates whether the module i is executed locally ($x_i = 0$) or remotely ($x_i = 1$). The solution $x_1, x_2...x_n$ represents the required offloading partitioning of the application.

The cost function is represented as follows:

$$\min_{x \in 0,1} \left(c_{transfer} * w_{tr} + c_{memory} * w_{mem} + c_{CPU} * w_{CPU}\right) \tag{1}$$

where

$$c_{transfer} = \sum_{i=1}^{n} code_i * x_i + \sum_{i=1}^{n} \sum_{j=1}^{k} tr_j * (x_j \ XOR \ x_i) \tag{2}$$

$$c_{memory} = \sum_{i=1}^{n} mem_i * (1 - x_i) \tag{3}$$

$$c_{CPU} = \sum_{i=1}^{n} code_i * \alpha * (1 - x_i) \tag{4}$$

There are three parts in the cost function. The first part depicts the transfer cost for remote execution of services, including the transfer cost of its related services which are not at the same execution location. The latter part of Eq.(2) implicitly includes the dependency relationship between modules, i.e. if the output of one module is an input of another. The c_{memory} contains the memory cost on the mobile device, and c_{CPU} the CPU cost on the mobile device is, where α is the convert factor mapping the relationship between code size and CPU instructions, which is taken to be 10 based on [16]. w_{tr}, w_{mem}, w_{CPU} are the weights of each cost, which can lead to different objectives, for example lowest memory cost, lowest CPU load or lowest interaction latency.

The three constraints are expressed as the following.

Minimized memory usage. First, the memory cost of resident service can not be more than available memory on the mobile device, i.e.

$$\sum_{i=1}^{n} mem_i * (1 - x_i) \leq avail_{mem} * f_1 \tag{5}$$

where $avail_{mem}$ can be obtained from the mobile device, f_1 is the factor to determine the memory threshold to be used, because the application can not occupy the whole free memory on the mobile device.

Minimized energy usage. Second, for the offloaded services, the energy consumption of offloading should not be greater than not offloading [17], i.e.

$$E_{local} - E_{offload} > 0 \tag{6}$$

The local energy consumption can be expressed using the number of local instructions to be executed I_{local}, local execution speed S_{local} and the power used of local execution P_{local} [17]. At the first decision time, I_{local} is estimated according to the code size. After the first decision, this number is collected from our framework (by calling the statistic method provided from Android API). Obviously, there is relationship between instruction number and the power used for that instruction while doing power profiling.

$$E_{local} = \frac{P_{local} * I_{local}}{S_{local}} \tag{7}$$

The energy cost of offloading some parts to remote cloud can be expressed as the sum of energy consumption during waiting for the results from the cloud E_{idle}, transferring (including sending E_s and receiving E_s) the services to be offloaded [17] and also the additional data which may be needed on the remote cloud E_{extra}.

$$\begin{aligned} E_{offload} &= E_s + E_{idle} + E_r + E_{extra} \\ &= P_s * (t_s + t_{extra}) + P_{idle} * t_{idle} + P_r * t_r \end{aligned} \tag{8}$$

The idle time of the mobile device waiting for the result from cloud can be treated as the execution time of remote cloud, so the formula becomes

$$\begin{aligned} E_{local} - E_{offload} = \frac{P_{local} * I_{local}}{S_{local}} - \frac{P_{local} * I_{local}}{S_{cloud}} \\ - \frac{P_s * (D_s + D_{extra})}{B_s} - \frac{P_r * D_r}{B_r} \end{aligned} \tag{9}$$

where D_s and D_r are the total data sizes to be sent and received, D_{extra} is the size of extra data needed because of offloading, which is determined at runtime, B_s and B_r are the bandwidths of sending and receiving data, and S_{cloud} is the remote execution speed. Additionally,

$$I_{local} = \sum_{i=1}^{n} code_i * type_i * x_i \tag{10}$$

$$D_s = \sum_{i=1}^{n} send_i * type_i * x_i \tag{11}$$

$$D_r = \sum_{i=1}^{n} rec_i * type_i * x_i \tag{12}$$

where $type_i \in \{0, 1\}$ represents whether a service is offloadable or not.

Minimized execution time. Third, the third constraint is enabled when the user prefers fast execution, i.e.

$$t_{local} - t_{offload} > 0 \tag{13}$$

The local execution time can be expressed as the ratio of CPU instructions to local CPU frequency, meanwhile, the remote execution time consists of the time consumed by CPU, file transmission and the overhead of our middleware.

$$t_{local} = \frac{I_{local}}{S_{local}} * x_i \tag{14}$$

$$t_{offload} = (\frac{I_{local}}{S_{cloud}} + \frac{D_{extra}}{B_s} + t_{overhead}) * x_i \tag{15}$$

where $t_{overhead}$ is the overhead which our framework brings in.

According to the constraints above, we now transform the partitioning problem to an optimization problem. The solution of $x_1, x_2...x_n$, is the optimized partitioning strategy. By using integer liner programming (ILP) on the mobile device, MACS gets a global optimization result. Whenever the the parameters in the model change, such as available memory or network bandwidth, the partitioning is adapted by solving the new optimization problem.

Although MACS introduces the overhead because of using a proxy for communication between offloaded services and the mobile application, the overhead is relatively small, which is shown in the evaluation part. Our MACS also does the profiling for each offloaded module/service to dynamically change its execution plan and adjust the partitioning.

V. Experimental Evaluation of Offloading

We evaluate our MACS framework with two use case phone applications. The first application implements the well-known N-Queens problem. It is chosen because the performance bottleneck represents a pure computation problem. This use case can easily show the overhead introduced by MACS middleware. The second application involves face detection and recognition in video files. This use case involves lots of computation, but also requires much more memory resources to process and obtain results.

The second case can process a video file, and detect faces from the video file, cluster them and provide the time point cues for video navigation. The results can then be used for faster video navigation on small screen devices (Figure 2). The video file is processed with OpenCV [1] and FFmpeg [2] libraries.

[1] http://opencv.willowgarage.com
[2] http://ffmpeg.org

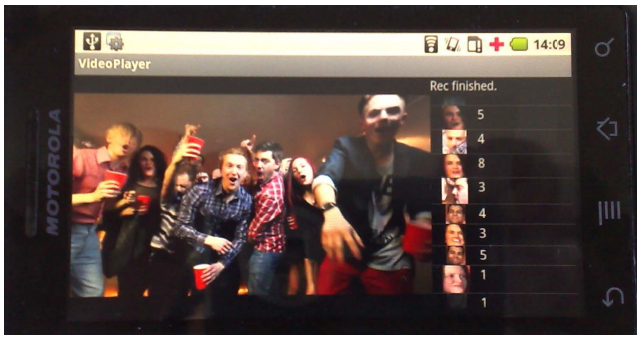


Fig. 2. Snapshot of prototype application of face detection and recognition using MACS middleware

In the processing, faces in the video file are detected by the existed implementation in OpenCV, and then the detected faces are recognized by the method proposed by Turk and Pentland [18], and after that, the faces are clustered.

In the implementation we used JavaCV [3] for video processing. When the application gets the results from the processing, it shows all detected faces as a clustered view. The user can select a cluster, and then navigate to the time points where that face occurs in the video. Thus, the application can accelerate navigation in a video based on persons that occur within.

A. Setup of the Evaluation

Hardware. The hardware we are using in the evaluation is as follows. A Motorola Milestone mobile phone based on Android platform 2.2 is used in the evaluation. A desktop computer which includes quad-core CPU acts as a cloud provider that can host the offloaded computation. The details about the hardware components for the mobile device and desktop computer are shown in Table I.

TABLE I
HARDWARE COMPONENTS OF MOBILE DEVICE AND DESKTOP COMPUTER

Hardware Component	Milestone	PC
Processor	ARM A8 600 MHz	Quad-Core 2.83GHz
Memory	256MB	8GB
WLAN	Wi-Fi 802.11 b/g	N/A
OS	Android 2.2	Windows XP x64

Network Topology. While offloading services to the remote cloud, the mobile phone connects to a nearby access point. Since the wireless local area network is encrypted with Wi-Fi Protected Access 2 (WPA2) security protocol, the data speed is not as fast as non-encrypted considering of the overhead introduced by the security protocol. The desktop computer is connected to the Internet directly by network cable, whose bandwidth is 100 Mbps.

Energy Estimation Model. We adopt a method as the one proposed by Zhang et al. [19], a power model for an Android phone and a measurement application for the energy consumption on the Android-based mobile device on the fly. Using their software, the energy consumption of each hardware component of the Motorola Milestone such as LCD, CPU and Wi-Fi can be measured separately (see Table II).

[3]http://code.google.com/p/javacv

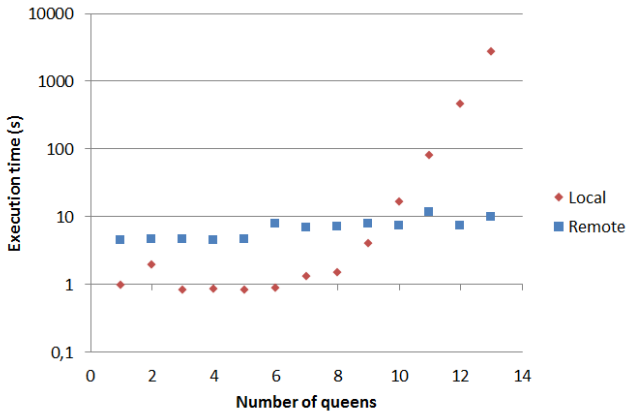


Fig. 3. Execution time of N-queens

TABLE II
ESTIMATED ENERGY CONSUMPTION OF MOBILE DEVICE

Hardware Component	Estimated Energy Consumption (W)
Processor	0.4 (Idle: 0.05)
Wi-Fi	0.75 (Low: 0.03)
LCD	0.9

B. Results of the Use Case 1

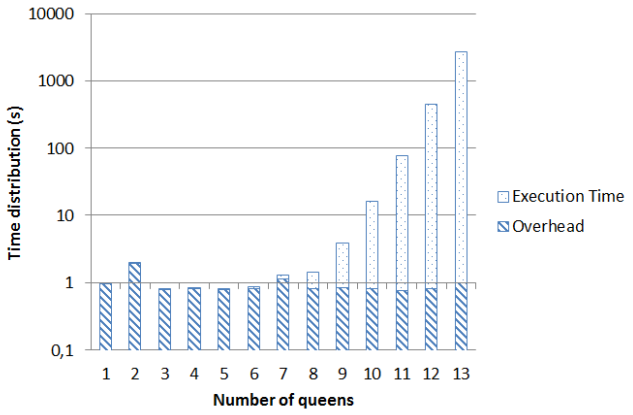
We use the algorithm by Sedgewick and Wayne[1]. The basic idea is to use recursion and back-tracking to enumerate all possible solutions. Although it is not the best algorithm, it is often used for solving the N-queens puzzle. It is clear that with the increase of N, much more steps are spent to find solutions, which is extremely time-consuming for the mobile device.

We run the N-Queens on the local device and offload to the remote cloud separately, for $N = 1$ to $N = 13$. As when $N = 14$, it will take hours to finish on the local device, it is not realistic not to be offloaded while doing computation after $N = 14$.

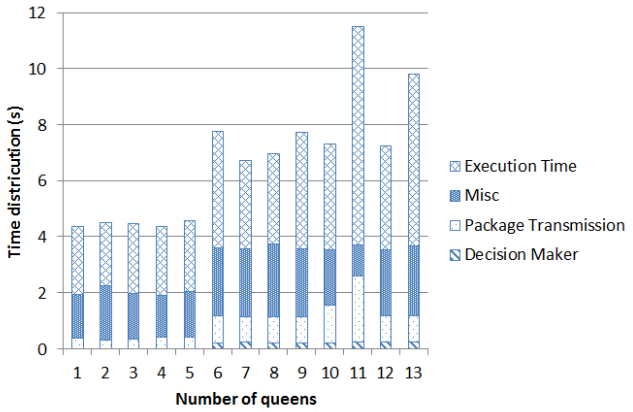
Figure 3 shows the time duration of execution of the specific calculation service. From $N = 1$ to $N = 9$, the execution speed on the local device is acceptable compared with the remote speed and to run the method locally is better, but after $N = 10$, the remote speed dominates to be the better option as the computation time dominates the total time in the rest cases, and the remote execution speed is also relative stable, there is no huge variation for remote speed.

Figure 4 shows the different time parts, which are made up of the total spent time. With the increase of the queens number, the local execution time increases outstandingly,

[1]http://introcs.cs.princeton.edu/java/23recursion/Queens.java.html



(a)



(b)

Fig. 4. Total time distribution of N-queens: (a) local execution and (b) remote execution

especially from $N = 9$, the execution time of calculating solution occupies more than half of the total time. Meanwhile, the overhead, our framework brings, stays constant. As for the remote execution, the overhead is broken down to three parts, one is the package offloading time, one is the decision maker time, the rest one is the residual overhead. It shows that our decision model costs only little time to finish the determination, and the transmission time of remote package occupies also few parts of total time, since the remote package is small. The execution time of solving the N-queens is relative stable, except for the $N = 11$, which is a deviation during the execution and measurement.

The last Figure 5 shows the results of consumed energy with and without offloading. As for the local execution, most of the time is spent on computation, since our energy model involves CPU and LCD, and the LCD is always on while computation, so that the energy consumption of CPU and LCD dominates the total energy consumption of local execution. The execution time is significantly increased from $N = 9$ compared to the remote execution, which leads to the highest energy value. In contrast to those, the remote execution time is nearly stable,

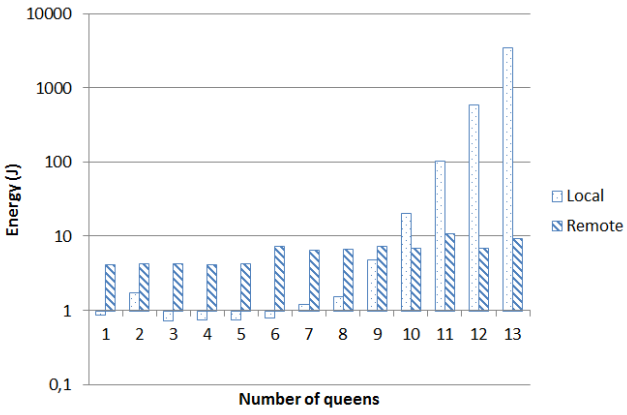


Fig. 5. Energy consumption of N-queens

TABLE III
VIDEO DURATION AND FILE SIZE

Duration (seconds)	File size (bytes)
10	1864984
20	3864612
30	5827420
40	7754219
50	9633240
60	11584020

so that the consumed energy is almost at the same level.

C. Results of the Use Case 2

Six video files are used in the evaluation. All of them belong to a same original video file with different length of time, 10, 20, 30, 40, 50 and 60 seconds (see Table III). The video resolution is 720 pixels $\times$ 480 pixels, the fps is set to 30, the overall bit rate is 1500 Kbps and the video is compressed with MPEG-4 format, 3GPP Media Release 5 profile. The audio codec is AAC, and the bit rate for audio is 30 Kbps.

In order to get an more accurate estimation of execution time which is used in the model, we first run the face detection services locally, and keep track of the spent time (second) and the file size (bytes), and then a linear regression is used to reflect the relationship between the spent time and the file size. Considering the number of CPU instructions provided by Android API, it can only be used to make estimation on the execution which involves no native calls, we don't directly use that count, but focus on the execution time. The regression shows that,

$$Time = 0.0005 * FileSize - 246.09 \quad (16)$$

We use this heuristic equation in our model to make determination of the execution time.

On Figure 6 can be seen clearly that the execution time is reduced hugely while offloading compared with the local execution. Even dealing with the 10 seconds video file, the local device spends more than 15 minutes on processing and

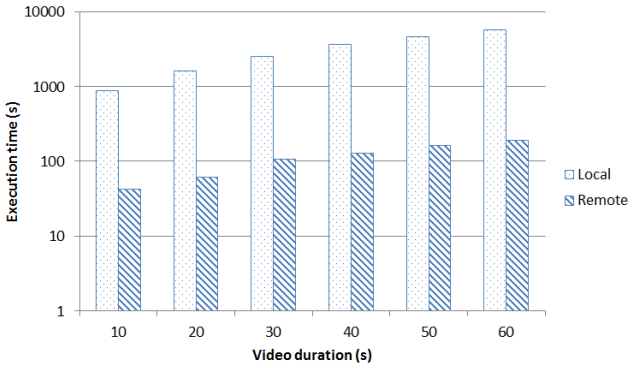


Fig. 6. Execution time of face detection

TABLE IV
VIDEO DURATION AND SPEED UP OF FACE DETECTION IN VIDEO FILE

Duration (seconds)	10	20	30	40	50	60
Speed up	×20	×26	×23	×28	×28	×29

detecting, but the corresponding remote offloading takes only less than 1 minute. Each time the computation is offloaded to the remote cloud, the execution speed can be reduced to more than 20 times, see Table IV. It is absolutely not acceptable to let the CPU of local mobile device 100% load for such long time, and it confirms that the video processing task is still a huge burden for the mobile device.

With the huge difference between local and remote execution time, it can apparently conclude that the local energy consumption is worse than the remote ones, because most of the time are spent on CPU and LCD, which are the top two of energy consuming components, see Figure 7. The Table V also depicts the energy saving situation while offloading, the energy can be saved more than 94 percentage thanks to the offloading.

Figure 8 describes the composition of the local and remote total spent time in details. As the execution time increases with the bigger video file size, the overhead our framework

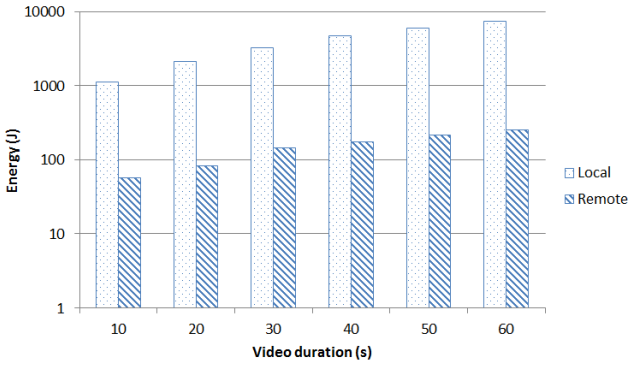
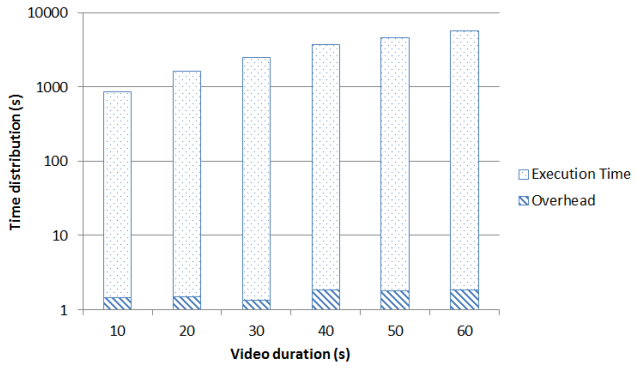


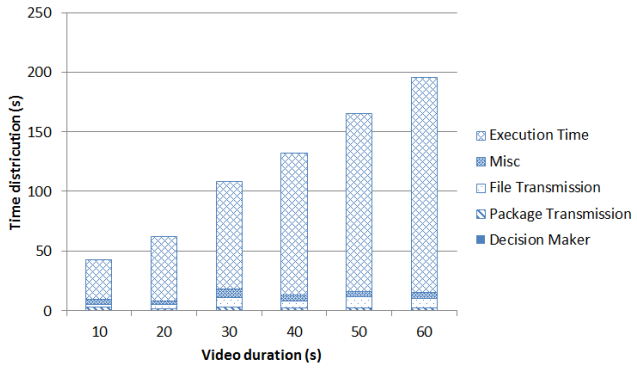
Fig. 7. Energy consumption of face detection

TABLE V
VIDEO DURATION AND ENERGY SAVE

Duration (seconds)	10	20	30	40	50	60
Energy save (%)	94.98	96.07	95.59	96.37	96.33	96.55



(a)



(b)

Fig. 8. Total time distribution of face detection: (a) local execution and (b) remote execution

brings occupies about only 0.1%, which can be nearly omitted. Regarding to the remote execution, the total spent time consists of execution time, needed file transmission time, package transmission (service offloading) time, decision maker time and so on. With the increase of the video file size, the file transmission time also raises, but compared to the total time, it is not significant. The decision maker does its determination in less than 1 second, which is only 1 percentage of the total spent time. The total overhead our framework brings is about 5 percentage of the total time, which is acceptable considering about the speed up and energy save above.

The face clustering can only be done on the remote cloud because of the software limitation on the local mobile device. Most of the execution time is spent on building/rebuilding training set. If the training set is already available before the remote execution, then the estimated execution time can be significantly reduced.

VI. Discussion

Offloading perhaps is not the suited for every mobile applications, but from the results of the two use cases, we see that when an application uses complex or time-consuming algorithms such as recursion, by offloading those parts into the cloud, time and energy consumption are reduced, so that the local execution time is reduced to an acceptable level. Offloading can lower the CPU load on a mobile device significantly. It can also save lots of energy, which indicates that the battery time can be increased compared to the local execution, as shown in the second use case, where more than 90% of energy is saved and the calculation speed is up to 20 times over local execution.

The results also prove that the overhead of our framework is small and acceptable with the increase of needed computation, it is better to push those computations which cost considerable resources to the remote cloud. But for the small N in the N-Queens problem, the overhead occupies almost half of the total execution time because of the needed computation is small so that it takes only little time to obtain the results. This shows a clear advantage of local execution over remote offloading when the needed computation is not much. In a word, the more computation is needed, offloading has more advantage. Since we use Wi-Fi in the evaluation, the time of sending files and receiving results has small proportion, but if 3G or GPRS are used, the offloading time will surely increase.

VII. Conclusions and Future Work

The results show that the local execution time can be reduced a lot through offloading, which is sometimes not acceptable for users to wait for, and by pushing the computation to the remote cloud can lower the CPU load on mobile devices significantly thanks to the remote cloud, since most of the computations are offloaded to the remote cloud. Meanwhile, lots of energy can be saved which indicates users can have more battery time compared to the local execution. The results also prove that the overhead of our framework is small.

Our framework supports offloading of multiple Android services. If there are multiple services in one application and all of those services can be offloaded to the remote clouds, our resource monitor natively supports this situation and can make the corresponding allocation determination, so that some of the services should be offloaded and the rest of the services should be run locally.

The next steps are to enable parallelization of the offloaded services. Additionally, we can extend the current middleware so that it supports automatic partitioning arbitrary mobile applications. A great challenge is how to estimate the characteristics of an application depending on different input parameters, which is precisely the relationship between the input of the invoked method and the execution time. We could characterize the relationship between execution time and input parameters by running the target application several times and adapt the offloading algorithm.

References

[1] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2009. [Online]. Available: http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

[3] D. Kovachev, Y. Cao, and R. Klamma, "Mobile Cloud Computing: A Comparison of Application Models," *CoRR*, vol. abs/1107.4940, 2011.

[4] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a Computation Offloading Framework for Smartphones," in *Proceedings of the 2nd International ICST Conference on Mobile Computing, Applications, and Services (MobiCASE 2010)*, Santa Clara, CA, USA, October 2010.

[5] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an Elastic Application Model for Augmenting the Computing Capabilities of Mobile Devices with Cloud Computing," *Mobile Networks and Applications*, vol. 16, pp. 270–284, 2011.

[6] B.-G. Chun and P. Maniatis, "Augmented Smartphone Applications Through Clone Cloud Execution," in *Proceedings of the 12th Workshop on Hot Topics in Operating Systems (HotOS XII)*. Monte Verita, Switzerland: USENIX, 2009.

[7] S. Kosta, A. Aucinas, P. Hui, and R. M. X. Zhang, "Unleashing the Power of Mobile Cloud Computing using ThinkAir," *CoRR*, vol. abs/1105.3232, 2011, informal publication.

[8] S. Ou, K. Yang, and J. Zhang, "An Effective Offloading Middleware for Pervasive Services on Mobile Devices," *Pervasive Mob. Comput.*, vol. 3, pp. 362–385, August 2007.

[9] 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 (ACM MobiSys '10)*. San Francisco, CA, USA: ACM, 2010, pp. 49–62.

[10] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications," in *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware (Middleware '09)*. Urbana Champaign, IL, USA: Springer, Nov. 2009, pp. 1–20.

[11] X. Gu, A. Messer, I. Greenberg, D. Milojicic, and K. Nahrstedt, "Adaptive Offloading for Pervasive Computing," *IEEE Pervasive Computing*, vol. 3, pp. 66–73, July 2004.

[12] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environments," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*. Dallas-Fort Worth, TX, USA: IEEE, 2003, pp. 107–114.

[13] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong, "Securing Elastic Applications on Mobile Devices for Cloud Computing," in *CCSW '09: Proceedings of the 2009 ACM Workshop on Cloud Computing Security*. Chicago, IL, USA: ACM, Nov. 2009, pp. 127–134.

[14] X. Zhang, S. Jeong, A. Kunjithapatham, and S. Gibbs, "Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms," in *Third International ICST Conference on Mobile Wireless Middleware, Operating Systems, and Applications*, Chicago, IL, USA, 2010.

[15] R. Kemp, N. Palmer, T. Kielmann, F. Seinstra, N. Drost, J. Maassen, and H. Bal, "eyeDentify: Multimedia Cyber Foraging from a Smartphone," in *Proceedings of the 11th IEEE International Symposium on Multimedia (ISM 2009)* , 2009, pp. 392–399.

[16] J. K. Ousterhout, "Scripting: Higher Level Programming for the 21st Century," *IEEE Computer*, vol. 31, pp. 23–30, 1997.

[17] 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.

[18] M. Turk and A. Pentland, "Eigenfaces for Recognition," *J. Cognitive Neuroscience, MIT Press*, vol. 3, pp. 71–86, January 1991.

[19] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. ACM, 2010, pp. 105–114.