Energy Efficient Task Scheduling in Mobile Cloud Computing

Dezhong Yao, Chen Yu, Hai Jin, Jiehan Zhou

Energy Efficient Task Scheduling in Mobile Cloud Computing

Dezhong Yao, Chen Yu, Hai Jin, and Jiehan Zhou+

Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan 430074, China
{dyao,yuchen,hjin}@hust.edu.cn
+ Department of Computer Science and Engeering University of Oulu,
Oulu 90014, Finland

Abstract. Cloud computing can enhance the computing capability of mobile systems by offloading. However, the communication between the mobile device and the cloud is not free. Transmitting large data to cloud consumes much more energy than processing data in mobile device, especially in a low bandwidth condition. Further, some processing tasks can avoid transmitting large data between mobile device and server. Those processing tasks (encoding, rendering) are as the compress algorithm, which can reduce the size of raw data before it is sent to server. In this paper, we present an *energy efficient task scheduling* strategy (EETS) to determine what kind of task with certain amount of data should be chosen to be offloaded under different environment. We have evaluated the scheduler by using an Android smartphone. The results show that our strategy can achieve 99% of accuracy to choose the right action in order to minimize the system energy usage.

Keywords: Mobile cloud computing, Energy-efficient, Task scheduling, Offloading

1 Introduction

With a rapid development of embedded systems, high speed wireless networks and could computing, mobile devices (e.g., smartphone, tablets, wearable devices, etc.) are increasingly becoming a common stuff of human daily life. We use mobile devices to do many of our jobs that we used to do on desktop. Mobile cloud computing technique allows those ideas to become a reality. Nonetheless, the more mobile devices we have, the less happiness we are with the battery lifetime. This is because we are still using the traditional power supplying method and materials.

Computing offloading technique is proposed with the objective to migrate the complex computation works from resource-limited devices to powerful machines. This avoids taking a long application execution time on mobile devices which results in large amount of power consumption. However, current offloading studies do not consider about the energy consumption used by data transport. Offloading mechanisms just focus on avoiding use local CPU resource in order to save energy spent on processing. They pay limit attention on long time data transmission problem and they consider best case scenarios: ideal high speed networks. Unfortunately, long wireless network latencies are a fundamental obstacle [1]. Some researches [2,3] prove that it can significantly save energy when we minimize the use of network. For example, speech recognition utility is a heavy processing task, it should be processed in cloud servers according the offloading scheduling algorithm [4]. While voice data is usually larger than text data, it will take more energy to send the big data to server than to process the recognition algorithm locally and then send the text result to server.

In this paper, we propose an *energy efficient task scheduling* strategy (EETS) which focuses on reducing the amount of data transmission. Our goal is to identify which kind of task is suitable for offloading and which is not. To address this problem we build an energy consumption model for each task. At last, we give an evaluation work on a suite of *Mediabench* [5] programs to show the efficiency of our scheduling mechanism. Our main contributions of this paper are as: 1)We present a task allocation algorithm which is based on a task's input/output data's size and storage path. 2)We study the data compression that is helpful to achieve energy saving between mobile device and cloud. 3)We give an energy consumption model for each task. Using this model, we can calculate the energy consuming difference between offloading or not offloading. We prove our model using real data from Android phones.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 introduces task execution scenario. In section 4, we describe energy consumption modeling technique for application task and explain task scheduling algorithm. Then we carry out a performance evaluation of all considered methods in section 5. Section 6 discusses our work and future works, whereas our conclusions are drawn in section 7.

2 Related Work

Could computing can provide computing resource, and users can use these to reduce the amounts of computation on mobile systems and save energy. Thus, cloud computing can save energy for mobile users through computation offloading [6,7]. However, to migrate large amount of data in a low-bandwidth network will cost much more energy than local computation.

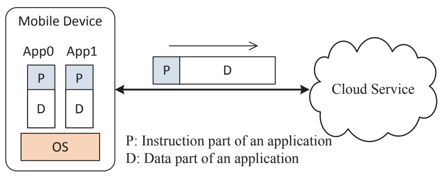
Offloading [6] is a method to offload part of the computation from mobile device to another resource rich platform. A number of recent works propose different methodologies to offload computation for specific applications [8,9]. In another part, lots of works discuss on how to pre-estimate the actual gain in terms of energy [10]. For a code complication, offloading might consume more energy than that of local processing when the size of codes is small [11].

The research work [7] suggests a program partitioning based on the estimation of the energy consumption (communication energy and computation energy) before the program execution. The optimal program partitioning for offloading is calculated based on the trade-off between the communication and computation costs. To do offloading in the dynamic environments, Tang et al. [12] consider three common environmental changes: power level, connection status and bandwidth. But they just explain the suitable solutions for offloading for different environments separately. Chun et al. [13] present a system to partition an application in dynamic environments. The proposed system follows three steps with different requirements related to the application structuring, the partitioning choice, and the security. Cuervo [14] introduces an architecture to dynamically partition an application at a runtime in three steps. However, all those works do not consider more about the task input and output data, which may cost a lot of energy in communication. None of the previous works related to mobile cloud computing explicitly studies the actual input data and output data location problem. They assume that the data comes from local and to be stored in local device. Otherwise, we need to spend much energy on useless data transmission work.

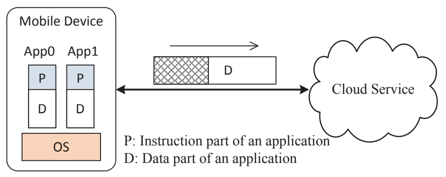
3 Task Execution Scenario

Simple task does not take too much time to execute, so it does not need offloading [7]. Complex applications will consume more energy than the simple one. So in this section we will classify the applications with complex computation task.

With the offloading's help, we can put the rendering work on cloud, as shown in Fig. 1(a). After the rendering result is ready, the data will be sent back to mobile device. The energy spent on computation part will be saved with an extra cost of data communication. Sometimes, it is not worth to offload rendering work from mobile client to server when the data size is big and the network bandwidth is slow. The mobile device will keep communication with cloud servers for a long time. For some compression like applications, we would better to do compress work on mobile device than to do it on the cloud servers. The data size can be reduced after it is processed. So the communication time will be saved. This progress is shown in Fig. 1(b).


Mobile Device
App0
App1
P
P
D
D
OS
P
D
Cloud Service
P: Instruction part of an application
D: Data part of an application

(a) Offloading: mobile device needs to transmit all data to server


Mobile Device
App0
App1
P
P
D
D
OS
D
Cloud Service
P: Instruction part of an application
D: Data part of an application

(b) Non-offloading: the mobile device just transmits the executed data

Fig. 1. Offloading scheme

4 Energy Efficient Task Scheduler

In this section, we will talk about the energy consumption model. Our proposed task scheduler makes following assumptions: 1) Each task can be processed on mobile device. 2) Each task has a fixed compress ratio of input data and output data. 3) The network bandwidth between mobile device and remote cloud is fixed and stable. 4) The computational power required by a task has a liner relationship with the processing time.

4.1 Energy Model

Energy consumption of mobile devices depends on the computation and communication loads. To explore the energy consumption of each task, we suppose the task computation requires I instructions. The task needs to deal with D bytes of data and will generate D' bytes result. We use B to stand for current network bandwidth. It will task D/B seconds to transmit and receive data. We define our task as follows:

Definition 1. *[Application Task] $T(I, D, D')$. A mobile application task T has I instructions to be executed. The task uses D bytes input data and generates D' bytes output data.*

The mobile system consumes P_c (watt per instruction) for computing and P_{tr} (watt per second) for sending and receiving data.

If we choose to execute our task using offloading, we need to send our code and data to server, as shown in Fig. 1(b). So the total energy consumption is:

$$E_{off} = P_{tr} * \frac{D}{B} \tag{1}$$

Suppose the output data D' is k times smaller than the original data D, we use compress ratio k to describe the relationship between input data and output data: ($D' = D * k$). In order to simplify the compression algorithm impact on the size of output data when the size of input data is different, we just consider the application task with fixed compression ratio. If the mobile device performs the task, the energy consumption is:

$$E_{nonoff} = P_c * I + P_{tr} \frac{D'}{B} \tag{2}$$

Definition 2. *[Energy saving value S] The amount of energy that can be saved, when the task chooses to do offloading.*

The amount of energy saved is:

$$\begin{aligned} S &= E_{off} - E_{nonoff} && (3) \\ &= P_{tr} * \frac{D}{B} - P_c * I - P_{tr} * \frac{D'}{B} = P_{tr} * \frac{D}{B} - P_c * I - P_{tr} * \frac{D * k}{B} \\ &= P_{tr} * (1-k)\frac{D}{B} - P_c * I && (4) \end{aligned}$$

If offloading use less energy to finish a task than non-offloading, this formula produces a negative result: $S < 0$. It's better to choose offloading. When we try to transmit large amount of data D for a compression task ($k < 1$), Eq. (3) will be positive. It means that it is better to execute the task locally.

4.2 Execution Model

It is a big advantage to do offloading in cloud environment, if a task has to download a large number of data from remote servers. In the previous studies, researchers focus on determining whether to offload computation by predicting the relationships among the three factors: network bandwidth B, the amount of processing instructions I and the amount of data to be transmitted D. However, there is a fundamental assumption underling this analysis: the server does not already contain the input data and all the data must be sent to the server to do offloading. The mobile client has to offload the program code and data to the server. Now, cloud computing changes this assumption. Many mobile devices have cloud storage service. Offloading can use the data stored in the cloud. This will significantly reduce the energy consumption on data transmission. The execution workflow is shown in Fig. 2. The input file of an application could come from local files or remote storage service. The execution code is only on mobile device. Then, we can choose to offload computation task to virtual machine or run locally in mobile device. How to store the output result data of the application is depended by user settings.

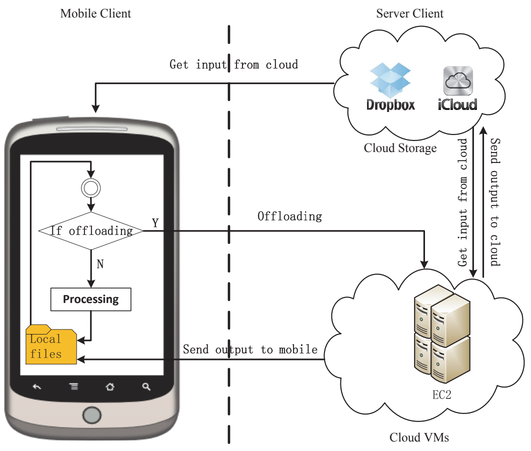


Fig. 2. Execution workflow of a mobile application

The detail of each possible workflow is listed in Fig. 3. The computing code is only stored in local mobile device. Compare with offloading and non-offloading mechanism, the main difference is processing code in cloud or in mobile device.

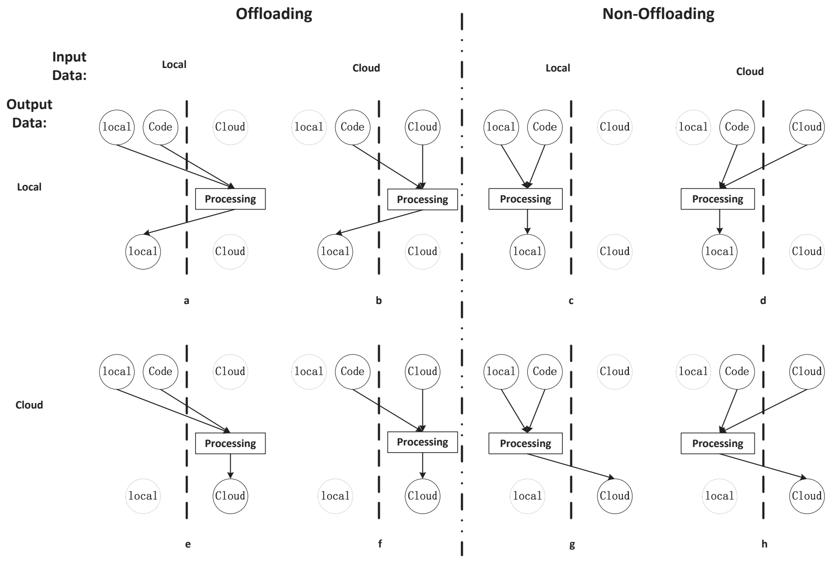


Fig. 3. Execution mode for different input/output situation

There are 8 executing mode depending on where we get the input data and where we store the output data. In the next section, we will present a cost model for how much energy can be saved for the same input and same output situation.

4.3 The Cost Graph

In this section, we will consider the energy consumption cost for each situation shown in Fig. 3. We use the energy consumption model in section 4.1 to describe the power spent on computation part and data transmission part.

Fig. 4 shows the cost map of data communication with different data storage. Task is executed on the mobile device while the other shown task runs on the cloud server after offloading. The input data of task could be D_m in the local mobile device or D_s from cloud storage service. As the instruction code is usually smaller than the data, the cost of transmitting code could be ignored. The cost of transmitting the input data is described as $D_{i,j}(i, j \in m, s)$. After the execution, the result data, $D'_{j,l}(j, l \in m, s)$, needs to be transported to the target place. As we process the same code in mobile device and VM, it will generate the same result. The size of data file should be the same.

Based on above discussion, we calculate the energy consumption for each suitable shown in Fig. 3. The data stored in mobile device and in cloud are the same. So we have $D_m = D_s$. The cost of transport data from mobile to cloud and from cloud to mobile are the same, $D_{ms} = D_{sm}$. The data transport within the mobile device cost nothing, $D_{mm} = 0$. The data does transport from storage datacenter to virtual machine (VM) also do not cost energy consumption

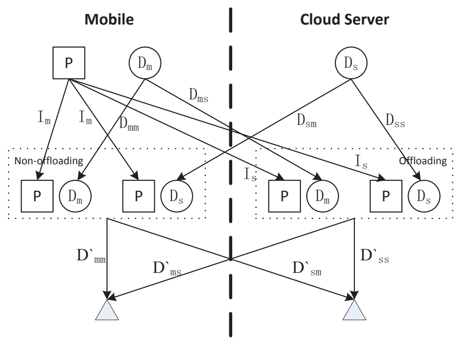


Fig. 4. Graph based cost map of data communication for offloading and non-offloading

in mobile device, $D_{ss} = 0$. For the output data, which does not need to be transported to another place, it will not consume the energy on mobile device. So we have $D'_{mm} = D'_{ss}$. As the result computed by the VM is the same data have computed in the mobile, the data exchange between mobile and server is equal: $D'_{ms} = D'_{ms}$.

Table 1. The energy saved S for different storage place

Input / Output	Local$(j = m)$	Cloud$(j = s)$
Local$(i = m)$	$P_{tr} * (1 + k)\frac{D_{ms}}{B} - P_c * I$	$P_{tr} * (1 - k)\frac{D_{ms}}{B} - P_c * I$
Cloud$(i = s)$	$P_{tr} * (k - 1)\frac{D_{ms}}{B} - P_c * I$	$P_{tr} * (-1 - k)\frac{D_{ms}}{B} - P_c * I$

Following the energy saving computation model in Eq. (4), we calculate the amount of energy saving for the 8 situations described in Fig. 3. Table 1 presents the different energy saving results according to the place data stored. If the energy is saved, the formula should be negative. Table 1 shows that:

1. Input data comes from local and output data to be stored in local.
   Offloading could take a great advantage in saving energy when computation work consumes more power than data communication. Otherwise, we should not choose offloading.
2. Input data comes from local and output data to be stored in cloud.
   Offloading can save lots of energy when the task compression ratio is above 1. It means that when the output data is larger than the input one, we should choose offloading. When the compression ratio is below 1, it also depends on the amount of data to be transmitted, network bandwidth and the amount of processing time.
3. Input data comes from cloud and output data to be stored in local.
   It is a little different with above situation. The offloading can save energy

```
Procedure Task_Scheduler
Input: (1) costgraph (2) task_profile
Output: offloading decision for each task
Procedure:
I, D, k, input_path, output_path
Procedure Scheduler(task_profile)
...
If the task can be processed
  if input_path is local
      if output_path is local
          use costgraph evaluate energy saved value S
          if S>0
            return False
          else
            return True
      else
          use costgraph evaluate energy saved value S
          if S>0
            return False
          else
            return True
  else
...
```

Fig. 5. Cost graph based task scheduling algorithm

when the compression ratio is below 1. When the task compression ratio is above 1, it is more valuable to do offloading when computation part uses more energy than data transmission.

4. Input data comes from cloud and output data to be stored in cloud.
   In this situation, it should always choose to do offloading that can save large amount of energy.

4.4 Task Scheduling Strategy

In this section, we will present a cost graph based task scheduling algorithm for saving energy on mobile devices. The scheduling algorithm, as shown in Fig. 5, will decide whether the task should be processed locally or offloading. If the answer is *true* then the task is placed in the scheduling queue. Otherwise, the task will be executed locally. Part of the algorithm is presented as produce *Task_Scheduler* in Fig. 5, which uses cost-graph and task profile as input data. The scheduling algorithm will be executed before each task. After check if current task is suitable for offloading, the scheduler will decide where to process the task.

5 Experiments Results

Our experiments are based on an Android-based platform device. In this section we describe the experiment setup, and performance evaluation results of all execution models as shown in Fig. 3. As processing time is also an important consideration for the use, there is a tradeoff between performance and energy consumption. In our paper, we only focus on energy issues on smartphone.

5.1 Experiments Setup

In the experiments, we assume that the WiFi network is always connected. In addition, we use *dropbox* [15] as our cloud data storage service. The software clones is running on virtual machines supplied by Amazon EC2 [16] platform. We use "Google Nexus One" as our test-device. Before each test program, the device will be continued charging for 2 hours after it is fully charged. The Android OS is restored to the factory settings to start a new test in order to make sure that there is only one test program installed. We use the percentage changes of the battery to show how much energy is used by the program.

5.2 Performance Evaluation

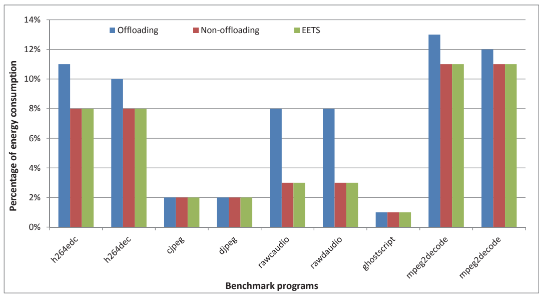
We implement our algorithm in Java and C using Android NDK. The task scheduler and data transmission parts use Java and the computation part uses C. To obtain a preliminary evaluation of the effectiveness of our method, we manually apply our algorithm to a suite of *Mediabench* [5] programs.

Test Programs. We select 9 execution models: $h264edc$, $h264dec$, *cjpeg*, *djpeg*, *rawcaudio*, *rawdaudio*, *ghostscript*, $mpeg2encode$, and $mpeg2decod$. The processing time for the application is measured at mobile device. Our task scheduling algorithm will determine determine to do offloading according to current data file size, data path, compression ratio and bandwidth.

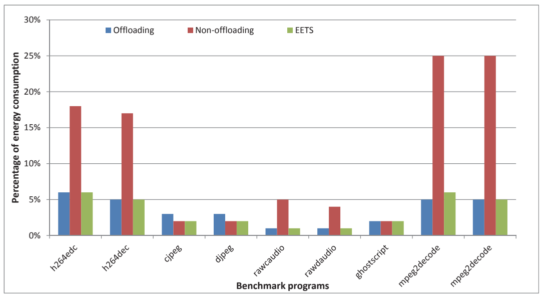
In our performance studies, we will compare our method with offloading and non-offloading as described in Eq. (1) and Eq. (2).

Test Measurement. By applying our algorithm to the 9 programs, we find that 6 of those 9 programs can get better energy saving by offloading. We evaluate our programs for four different groups which has different path of input data and output data. Then, we arrange each of our programs to do offloading and execute locally. After that, we use our proposed task scheduling algorithm to determine whether to do offloading. For each processing step, we record how much energy used before task finished. In Android mobile, we use how much percentage changes after the phone is fully charged. There is no other application running on the mobile device except system applications.

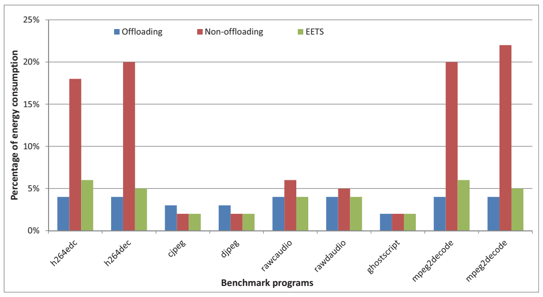
We use *dropbox* [15] as our cloud storage service, which has $100kbps$ network bandwidth. From Fig. 6(a), the offloading uses much more energy when the data is stored on mobile devices. The x-axis lists the test programs. The y-axis tells us how much percentage of energy used on test-phone to archive different programs. From Fig. 6(b) and Fig. 6(c), the complex application will cost lots of energy



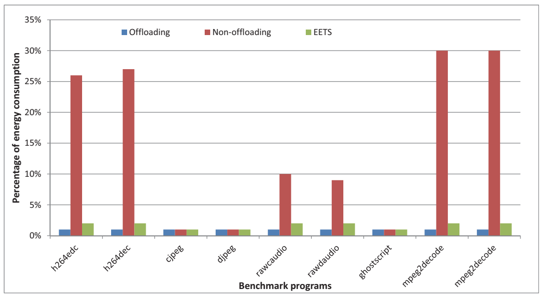
(a) Input=mobile and output=mobile



(b) Input=cloud and output=mobile



(c) Input=mobile and output=cloud



(d) Input=cloud and output=cloud

Fig. 6. Energy consumption on benchmark

when they are running on mobile device. It is better to do offloading. When we use the cloud data, we would use offloading to save our energy, which is shown in Fig. 6(d). As we can see from those figures, the proposed method also uses the minimum energy consumption to achieve the work.

For the case: ”input=mobile&output=mobile”, the data does not need to be transmitted to the server, so our proposed strategy (EETS) performs like non-offloading case, as shown in Fig. 6(a). H264 and MPEG are complex video encoding/decoding algorithms, they always use the most energy on smartphone. Those algorithms need to encode/decode each frame in a video. So they will use more energy than the jpeg algorithms. When those methods run on remote server, the smartphone only spends energy on data communication. EETS chooses the right strategy to schedule the data. Fig. 6(b), Fig. 6(c) and Fig. 6(d) are the some situations.

Efficiency Analysis. Based on the experimental results, we can conclude how efficient of our algorithm. First, we will try to find which one costs the minimum energy consumption to execute the task. We use a minimum function to calculate the smaller value between offloading and non-offloading: *min*(percentage of offloading, percentage of nonoffloading). Then we compare the minimum value with the proposed task scheduling method. The accuracy of our proposed schemer is above 99% in average, which means our scheduling algorithm efficiently choose the best solution to allocate the tasks.

6 Discussion and Future Work

The proposed task scheduler is limited in some respects by it initially setups. It needs to collect the profile information of each task at the beginning. After learned the compression ratio and processing time, the scheduler can deal with new tasks. To solve this problem, we need to try some test programs to collect execution information or learn from system record after the task executed once.

The framework presented in this paper does not consider the power consumption on idle time waiting for task result and disk usage. While the task dose not execute for a long time, we think the power consumption is as same as the normal usage. The power consumption on disk usage is related to the type of storage device. HDD device uses much more energy than flash storage device. We plan to add this impact to our determine strategy in our future works.

7 Conclusion

In this paper, we present a novel energy aware dynamic task allocation strategy over mobile cloud computing environment. This scheme introduces a cost graph to determine offloading the work or not. The evaluation work shows that, our strategy can efficiently choose to offload or not on mobile device with 99% accuracy. The energy consumption on mobile device is controlled in minimum.

Acknowledgements

The work is supported by NSFC (No.61003220).

References

1. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for vm-based cloudlets in mobile computing. IEEE Pervasive Computing **8**(4) (2009) 14–23
2. Dinh, H.T., Lee, C., Niyato, D., Wang, P.: A survey of mobile cloud computing: architecture, applications, and approaches. Wireless Communications and Mobile Computing (2011)
3. Baliga, J., Ayre, R., Hinton, K.: Green cloud computing: Balancing energy in processing, storage, and transport. Proceedings of the IEEE **99**(1) (2011) 149–167
4. Barbera, M.V., Kosta, S., Mei, A., Stefa, J.: To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In: Proc. of the IEEE INFOCOM. (2013)
5. Lee, C., Potkonjak, M., Mangione-Smith, W.H.: Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems. In: Proc. of the 30th annual ACM/IEEE international symposium on Microarchitecture, IEEE Computer Society (1997) 330–335
6. Yang, K., Ou, S., Chen, H.H.: On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. IEEE Communications Magazine **46**(1) (2008) 56–63
7. Kumar, K., Lu, Y.H.: Cloud computing for mobile users: Can offloading computation save energy? Computer **43**(4) (2010) 51–56
8. Portokalidis, G., Homburg, P., Anagnostakis, K., Bos, H.: Paranoid android: versatile protection for smartphones. In: Proc. of the 26th Annual Computer Security Applications Conference, ACM (2010) 347–356
9. Chen, E.Y., Itoh, M.: Virtual smartphone over ip. In: Proc. of the IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM), IEEE (2010) 1–6
10. Wen, Y., Zhang, W., Luo, H.: Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones. In: Proc. of the 2012 IEEE INFOCOM, IEEE (2012) 2716–2720
11. Rudenko, A., Reiher, P., Popek, G.J., Kuenning, G.H.: Saving portable computer battery power through remote process execution. ACM SIGMOBILE Mobile Computing and Communications Review **2**(1) (1998) 19–26
12. Tang, M., Cao, J.: A dynamic mechanism for handling mobile computing environmental changes. In: Proc. of the 1st international conference on Scalable information systems, ACM (2006) 7
13. Chun, B.G., Maniatis, P.: Dynamically partitioning applications between weak devices and clouds. In: Proc. of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, ACM (2010) 7
14. Cuervo, E., Balasubramanian, A., Cho, D.K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: Maui: making smartphones last longer with code offload. In: Proc. of the 8th international conference on Mobile systems, applications, and services, ACM (2010) 49–62
15. Dropbox. http://dropbox.com
16. AmazonEC2. http://aws.amazon.com/ec2