Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing

**Feng Xia · Fangwei Ding · Jie Li · Xiangjie Kong ·
Laurence T. Yang · Jianhua Ma**

Abstract With prosperity of applications on smartphones, energy saving for smartphones has drawn increasing attention. In this paper we devise Phone2Cloud, a computation offloading-based system for energy saving on smartphones in the context of mobile cloud computing. Phone2Cloud offloads computation of an application running on smartphones to the cloud. The objective is to improve energy efficiency of smartphones and at the same time, enhance the application's performance through reducing its execution time. In this way, the user's experience can be improved. We implement the prototype of Phone2Cloud on Android and Hadoop environment. Two sets of experiments, including application experiments and scenario experiments, are conducted to evaluate the system. The experimental results show that Phone2Cloud can effectively save energy for smartphones and reduce the application's execution time.

F. Xia · F. Ding · J. Li · X. Kong (✉)
School of Software, Dalian University of Technology,
Dalian 116620, China
e-mail: xjkong@ieee.org

F. Xia
e-mail: f.xia@ieee.org

L. T. Yang
Department of Computer Science, St. Francis Xavier University,
Nova Scotia, Canada
e-mail: ltyang@stfx.ca

J. Ma
Faculty of Computer and Information Sciences,
Hosei University, Tokyo, Japan
e-mail: jianhua@hosei.ac.jp

1 Introduction

Smartphones have become increasingly popular in our daily life. They offer users more powerful processors, larger memory, multi-network interfaces and a wide variety of operating systems such as Apple iOS, Android and Windows Phone. Not only have the processor speed and memory size of smartphone increased significantly, but also the resolution of screen and the quality of the available sensors (Kemp et al. 2010a). The smartphone's processor speed has grown enormously in recent years as well as memory size, which have arrived at the same level as notebooks of five years ago. Furthermore, cellular networking technology grows from GSM networks to current 4G networks, which significantly increases bandwidth of wireless networks.

As processors are getting faster, memory is getting larger, screens are getting sharper and devices are equipped with multiple sensors (Cuervo et al. 2010), a large spectrum of novel and innovative applications have appeared. They are ranging from mobile games, to multimedia applications, social networking services and more (Thimmarayaswamy et al. 2011). Currently, users can easily get applications from market places, like Apple App Store and Google Play. Additionally, users are more likely to run resource-demanding applications, such as rich media applications using multiple inputs like cameras and sensors (Chun and Maniatis 2010).

These applications imply a heavy workload on processors, wireless network interfaces and display, which causes a significant energy cost (Xiao et al. 2010).

To sum up, smartphones provide multi-core processors, sharper screens, larger memory, multiple sensors and radios as well as enormous applications. These together put a heavy burden on battery's energy consumption (Kemp et al. 2010a). In the meantime, advances in battery technology and energy saving solutions have not kept pace with rapidly growing energy demands (Kemp et al. 2012). Furthermore, smartphones are getting thinner and smaller, which implies smaller batteries and less battery capacities. Therefore, the energy consumption has always been primary bottleneck for smartphones.

Many researchers and engineers have made great efforts (Heikkinen et al. 2012; Ju et al. 2012; Sheng et al. 2012; Rozner et al. 2010; Zhou et al. 2010; Wen et al. 2012; Kosta et al. 2012) on saving energy to extend lifetime of batteries. One popular approach of energy saving for mobile devices is computation offloading: applications take advantage of resource-rich infrastructures by migrating computation to these infrastructures (Cuervo et al. 2010). Recently, researchers have recognized offloading computation via networks to cloud can help reduce power consumption of smartphones (Wen et al. 2012; Kosta et al. 2012; Kumar and Lu 2010; Kemp et al. 2010b; Miettinen and Nurminen 2010; Aggarwal et al. 2010).

In this paper, we develop a computation offloading-based system for energy saving on smartphones, called Phone2Cloud. It offloads all or part of an application running on smartphones to cloud to achieve energy conservation, reduce the application's execution time and improve user's experience, i.e. meet user's delay-tolerance threshold as described in Balasubramanian et al. (2010).

Here we make the following contributions. First, we develop the Phone2Cloud system for energy saving on smartphones. Second, we propose and implement a modified offloading-decision making algorithm based on Kumar and Lu (2010), Xian et al. (2007), Liu et al. (2010) in our system, and a new element - user's delay-tolerance threshold is involved in the offloading-decision making algorithm. Finally, we conduct two types of experiments on our system, and the results demonstrate the superiority of our system.

The rest of the paper is organized as follows. In next section, we review related works on reducing power consumption for smartphones, especially on those taking advantage of computation offloading. Section 3 presents the architecture of the Phone2Cloud system. In Section 4, the major algorithms employed in Phone2Cloud are described. Evaluation and analysis of Phone2Cloud are conducted in Section 5. We conclude the paper and discuss extensions to our system in Section 6.

2 Related work

In this section, we briefly review four basic approaches for energy conservation and extending battery lifetime in smartphones.

Smart battery models and energy cost models To efficiently and effectively use a battery, it is important to treat the battery as a measurable resource whose attributes are available to the operating system and applications. For this purpose, smart battery models and energy cost models have been presented to model battery's attributes. A number of battery models (Jongerden and Haverkort 2009) such as ideal model, stochastic model, diffusion model and so forth have been proposed. As to energy cost models, for instance, Kim et al. (2007) present a low-level energy cost model for fast estimation of software energy consumption in off-the-shelf processor. Mahmud et al. (2005) have proposed a high-level energy cost model for predicting energy consumption in the wireless network access portion of a handheld device equipped with multiple radio interfaces. Moreover, a system-level energy model is proposed by Palit et al. (2008).

Avoiding energy waste In this line of research, the whole system or individual component is put into sleep state to save energy. Brakmo et al. (2004) present an energy reduction technique for handheld devices, called μSleep. It tries to put processor in sleep mode for short periods to save energy without affecting the user's experience. However, it is hard to precisely predict when to enter sleep state. Shih et al. (2002) propose an energy saving strategy, called wake on wireless, to reduce the phone's idle power. They power off the phone and its radio interfaces when the phone is not being used, and the phone is powered only when there is on-going traffic. Nevertheless users will completely lose network connection when the phone is not used.

Communication related energy saving Many works (Zhang et al. 2005; Pering et al. 2006; Ananthanarayanan and Stoica 2009) on reducing the energy of network communications have been done. Zhang et al. (2005) present and evaluate a system-level power management method for a mobile device to dynamically shut down its Wi-Fi interface. However, the idle state of Wi-Fi radio interface is hard to be predicted. Since Bluetooth causes much less energy than Wi-Fi, Pering et al. (2006) develop a system called Coolspots that automatically switches between Wi-Fi and Bluetooth to increase battery lifetime. However, the system needs to modify the infrastructure. Blue-Fi (Ananthanarayanan and Stoica 2009) uses a combination of Bluetooth contact-patterns and cell-tower information to predict the availability of the Wi-Fi connectivity, thus it avoids the long periods in idle state of Wi-Fi interface and

significantly reduces the number of scans for Wi-Fi discovery. However, it cannot work very well outside due to short range of Bluetooth.

Computation offloading-based energy saving The main idea in computation offloading is to migrate computation-intensive tasks from mobile device to a server or cloud via network in order to save energy on the mobile device. Quite a lot of works have been done on computation offloading, for example, (Kemp et al. 2010a, b; Kumar and Lu 2010; Miettinen and Nurminen 2010; Aggarwal et al. 2010; Xian et al. 2007; Gu et al. 2004; Chen et al. 2004; Ou et al. 2006; Nimmagadda et al. 2010). Most of them deal with offloading computation from mobile device to a desktop computer or server on the network. For example, Gu et al. (2004) develop an adaptive offloading system. It dynamically partitions an application and efficiently offloads part of the application to be executed on a nearby server. However, they need to modify JVM to support transparent migration of objects between mobile device and server. In Kemp et al. (2010a), Cuervo et al. present MAUI, a system enabling fine-grained energy-aware offloading of mobile code to the infrastructure. Although MAUI's energy savings and performance are impressive, it still needs to partition applications and incurs extra overhead. Some others focus on offloading computation to cloud via network. For instance, Kumar and Lu (2010) conduct a qualitative analysis on whether cloud computing can extend battery lifetime for users. They believe cloud computing can potentially save energy for mobile users. Kemp et al. (2010b) study how smartphones can benefit from the resources available in cloud. They are building a framework for applications to be offloaded to cloud. However, there is not any quantitative analysis in these papers. In Miettinen and Nurminen (2010), Miettinen and Nurminen provide a quantitative analysis of the critical factors affecting the energy consumption of mobile clients in cloud computing. Aggarwal et al. (2010) also conduct a quantitative analysis on mobile communication using cloud support. However, extensive quantitative analysis is still missing, and users' requirements, such as delay-tolerance threshold, are yet to be considered.

In contrast to the above state-of-the-art works, we focus on developing a computation offloading-based system and conducting a fully quantitative analysis on energy saving of the system. In addition, users' delay-tolerance threshold will be considered in our work.

3 System architecture

An illustration of the architecture of Phone2Cloud is provided in Fig. 1. Phone2Cloud consists of seven key components, including a bandwidth monitor, a resource monitor,
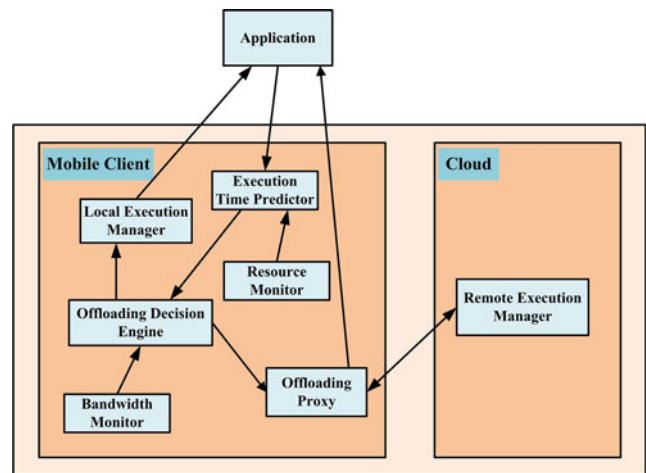

Fig. 1 Architecture of Phone2Cloud

an execution time predictor, an offloading decision engine, a local execution manager, a remote execution manager and an offloading proxy that ties the offloading decision engine and remote execution manager together. Among these components, the offloading decision engine is the core and the area of extensibility in the architecture. Offloading decision mechanisms can be easily added or removed from the framework. We will describe these components in the following sub-sections.

3.1 Bandwidth monitor and resource monitor

In order to support making an offloading decision, we need to monitor current bandwidth usage of the network and status of the smartphone, such as CPU workload. Therefore we employ the bandwidth monitor and the resource monitor. The former periodically monitors bandwidth of current network to predict average bandwidth when making an offloading decision. The latter is used to monitor the status of smartphone. For the sake of simplicity, we leverage it to monitor CPU workload of smartphone and predict the average CPU workload when offloading computation from smartphone to cloud. These two monitors serve the offloading decision engine and the execution time predictor separately, which will be discussed in detail below.

3.2 Execution time predictor

The execution time predictor is one of the key parts in Phone2Cloud. It is used to predict average execution time of an entire application on smartphone. Many works (Engblom et al. 2003; Chen et al. 2005; Schoeberl et al. 2010; Kirner et al. 2009) have been conducted on execution time prediction. In order to simplify the complexity of Phone2Cloud, we use a simple history-based method to predict execution time of an application running on smartphone. Given an

application, firstly, it gets the input size of the application and triggers the resource monitor to get predicted average CPU workload; secondly, it leverages the input size and the average CPU workload to search the log *L* to find two nearest points in distance, and then returns the mean of the two points' execution times as the average execution time of the application. Here, the log *L* is the execution history log and can be easily got by using a simple tool which repeatedly runs applications on smartphone. Besides, it is full of data in fixed form (*Application, Input size, Average CPU workload, Execution time*), and we see entries of a specified application in log *L* as points in a three-dimension space of input size, average CPU workload and execution time.

3.3 Offloading decision engine

In this component, Phone2Cloud decides whether to offload the whole or part of an application to cloud, namely, whether to offload computation of the application from smartphone to cloud. Given an application, it calls the bandwidth monitor to get average bandwidth of current network, triggers the execution time predictor to get average execution time of the application, and then uses the offloading-decision making algorithm described in Section 4.3 to make an offloading decision. When it decides to run the application locally, it calls local execution manager to execute the application. Otherwise, it invokes offloading proxy to handle offloading computation to cloud.

3.4 Offloading proxy

The offloading proxy sends required input data to the remote execution manager, receives the results returned by the remote execution manager, and delivers the results to the application. However, due to the fact that the application on smartphone cannot be directly run on cloud, we need to manually modify the application to make it possible not only to be run on cloud, but also to receive results from the offloading proxy. As we can see, Phone2Cloud is a semi-automatic offloading system.

3.5 Local execution manager and remote execution manager

The local and remote execution managers are mainly used to manage execution of the application in Phone2Cloud. The local execution manager is designed to execute the application on smartphones, simply invoking smartphone's operating system, such as Android and iOS, to run the application and log the execution information into log *L*. When the remote execution manger gets required input data from the offloading proxy, it executes offloading computation on cloud, and returns results to the offloading proxy.

4 Algorithms

In this section, we will describe the key methods used in Phone2Cloud, including CPU workload prediction in the resource monitor, bandwidth prediction in the bandwidth monitor, and the offloading-decision making algorithm.

4.1 CPU workload prediction

As discussed above, we propose a naive history-based method to predict average execution time of an application on smartphone. It leverages average CPU workload got from the resource monitor and input size of the application to predict execution time using the history log *L*.

As to CPU workload prediction, the resource monitor uses the basic exponential moving average algorithm (EMA for short) (Burgstahler and Neubauer 2002) to predict the average CPU workload of smartphone. It records the CPU workload c_t in database in the form of (*Timestamp, CPU workload*) periodically. Given current time period *t*, the EMA value for CPU workload is calculated recursively by Eqs. 1, 2 and 3, where C_t is the value of the EMA at any time period *t*, coefficient α represents the degree of weighting decrease and *N* is the number of time periods. We simply use the EMA value C_t as the average CPU workload.

$$C_1 = c_1 \tag{1}$$

$$C_t = \alpha \cdot c_t + (1 - \alpha) \cdot C_{t-1} \tag{2}$$

$$\alpha = 2/(N + 1) \tag{3}$$

4.2 Bandwidth prediction

To predict average bandwidth, EMA is also used in the bandwidth monitor when making an offloading decision. Moreover, it uses (4), (5) and (6) to recursively calculate EMA value B_t for bandwidth at current time period *t*, where b_t is the bandwidth recorded by bandwidth monitor periodically, coefficient β has the same meaning with α, and *N* is also the number of time periods.

$$B_1 = b_1 \tag{4}$$

$$B_t = \beta \cdot b_t + (1 - \beta) \cdot B_{t-1} \tag{5}$$

$$\beta = 2/(N + 1) \tag{6}$$

4.3 Offloading-decision making algorithm

Offloading-decision making is the core of Phone2Cloud. It decides whether or not to offload computation of the application from smartphone to cloud, which is mainly used in the offloading decision engine. Figure 2 shows the workflow of offloading-decision making algorithm. Table 1 lists the symbols and their meanings used in this paper.

Given an application, it needs five steps to make offloading decision:

Step 1: It gets the average execution time T_{exec} of the application running on smartphone predicted by the execution time predictor.

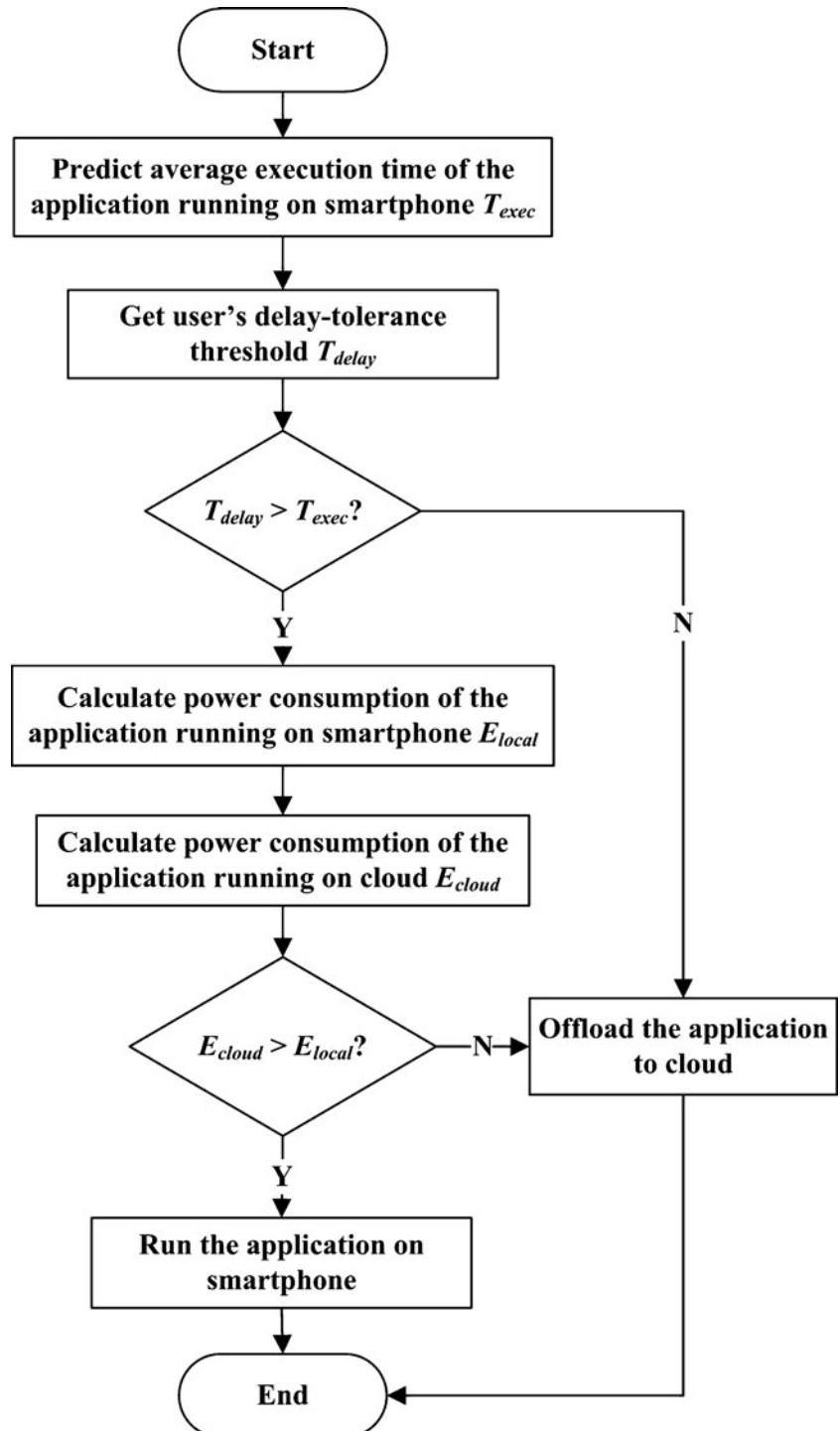**Step 2:** It gets user's delay-tolerance threshold T_{delay} specified by user, and then compares T_{exec} with

Fig. 2 Workflow of offloading-decision making algorithm

Table 1 List of notations

Variable	Description	Unit
C	Computation complexity of an application	I (instructions)
M	Computation execution rate (speed) of smartphone	I/s
S	Computation execution rate (speed) of cloud	I/s
n	Ratio of computation execution rate on cloud and smartphone	
P_{exec}	Active power of the processor on smartphone	$W = J/s$
P_{idle}	Idle power of smartphone (network interface & processor included)	$W = J/s$
P_{send}	Power consumed to send data (network interface & processor included)	$W = J/s$
D_{send}	Data size need to send to cloud	B
B_{send}	Sending bandwidth of the network	B/s
$P_{receive}$	Power consumed to receive data (network interface & processor included)	$W = J/s$
$D_{receive}$	Data size need to receive from cloud (execution results)	B
$B_{receive}$	Receiving bandwidth of the network	B/s
T_{exec}	Execution time on smartphone	s
T_{idle}	Idle time on smartphone (execution time on cloud)	s
T_{send}	Time for sending data	s
$T_{receive}$	Time for receiving data	s
E_{local}	Energy consumed by running application locally (on smartphone)	J
E_{cloud}	Energy consumed by running application on cloud	J
$E_{tradeoff}$	Trade-off energy consumed by computation offloading	J

T_{delay}. If user's delay-tolerance threshold is less than the average execution time of the application, then it decides to offload computation of the application from smartphone to cloud, expecting to buy some time for the application, and the algorithm ends. Otherwise, it goes to step 3.

Step 3: It calculates the power consumption of running the application on smartphone, called E_{local}, which will be described in Section 4.3.1.

Step 4: It calculates the power consumption of running the application on cloud, called E_{cloud}, and we will explain how to calculate this power consumption in Section 4.3.2.

Step 5: It compares E_{local} with E_{cloud}. If E_{cloud} is greater than E_{local}, then it decides to run the application on smartphone and then stops. Otherwise, it decides to offload computation of the application from smartphone to cloud, and the algorithm also ends.

In a nutshell, it basically compares two sets of variables (T_{exec} and T_{delay}, E_{local} and E_{cloud}) to make the offloading decision.

4.3.1 Energy consumption on smartphone

In order to calculate the energy consumption consumed by running the application on smartphone, we need to know the active power of the processor on smartphone P_{exec} and the average execution time T_{exec} predicted by the execution time predictor, and then we use (7) to get power consumption E_{local}.

$$E_{local} = P_{exec} \cdot T_{exec} \tag{7}$$

4.3.2 Energy consumption on cloud

Calculating E_{cloud} is more complicated than E_{local}. Before introducing how to calculate E_{cloud}, we make an assumption that the offloading part of the application is already on cloud when making an offloading decision.

As abovementioned, it needs three steps to finish computation offloading: sending the required input data, waiting the cloud completing execution of the offloaded computation and receiving execution results from cloud. Thus, E_{cloud} includes three parts: the energy consumed by sending required input data on smartphone E_{send}, the energy of waiting execution results on smartphone E_{idle} and the energy consumed by receiving execution results on smartphone $E_{receive}$.

E_{send} is calculated by Eq. 8, and the time for sending data from smartphone to cloud T_{send} is calculated by Eq. 9. Similarly, we calculate $E_{receive}$ by using Eqs. 10 and 11.

$$E_{send} = P_{send} \cdot T_{send} \tag{8}$$

$$T_{send} = D_{send}/B_{send} \tag{9}$$

$$E_{receive} = P_{receive} \cdot T_{receive} \tag{10}$$

$$T_{receive} = D_{receive}/B_{receive} \tag{11}$$

It is easy to get (12), and we use it to represent execution time of the application on smartphone. Since we just offload whole or part of the application to cloud, we are easy to come to Eq. 13 and use it to present idle time on smartphone, i.e. execution time of the cloud-version application. Assume that n represents the ratio of computation execution rate on cloud and smartphone, and then we get (14). Due to relatively large value of n, we simply use the maximum value $T_{idlemax}$ to calculate the idle time on smartphone in our experiments. Therefore, we can use Eqs. 14 and 15 to calculate E_{idle}.

$$T_{exec} = C/M \tag{12}$$

$$T_{idle} \leq C/S \tag{13}$$

$$T_{idle} \leq T_{idlemax} = T_{exec}/n \tag{14}$$

$$E_{idle} = P_{idle} \cdot T_{idle} \tag{15}$$

Thus, we can derive (16) to calculate E_{cloud} from Eqs. 8, 10 and 15. Furthermore, we get the trade-off energy consumption for computation offloading $E_{tradeoff}$ from Eqs. 7 and 16, i.e. the difference of energy consumption of running the application on smartphone and cloud, as shown in Eq. 17.

$$
\begin{aligned}
E_{cloud} &= E_{send} + E_{idle} + E_{receive} \\
&= P_{send} \cdot T_{send} + P_{idle} \cdot T_{idle} \\
&\quad + P_{receive} \cdot T_{receive}
\end{aligned} \tag{16}
$$

$$
\begin{aligned}
E_{tradeoff} &= E_{local} - E_{cloud} \\
&= P_{exec} \cdot T_{exec} - P_{send} \cdot T_{send} \\
&\quad - P_{idle} \cdot T_{idle} - P_{receive} \cdot T_{receive}
\end{aligned} \tag{17}
$$

We transform Eq. 17 to Eqs. 18 and 19. Then we make $E_{tradeoff}$ equal to zero and get a constant, called break-even transmission energy E'_0, as shown in Eq. 20. Once the application and the status of smartphone are specified, the value of E'_0 is constant.

$$E_{tradeoff} = P_{exec} \cdot T_{exec} - P_{idle} \cdot T_{idle} - E' \tag{18}$$

$$E' = P_{send} \cdot T_{send} + P_{receive} \cdot T_{receive} \tag{19}$$

$$E'_0 = P_{exec} \cdot T_{exec} - P_{idle} \cdot T_{idle} \tag{20}$$

In summary, we use Eqs. 14 and 20 to calculate E'_0, use Eqs. 9, 11 and 19 to calculate the variable E' and then compare E'_0 with E'. If E'_0 is greater than E', Phone2Cloud

offloads computation of the application from smartphone to cloud. Otherwise, it runs the application on smartphone.

5 Evaluation and analysis

In this section, we conduct two different sets of experiments: application experiments and scenario experiments. Then we analyze the energy consumption and execution time of the applications in our experiments under four different factors, including input size, bandwidth, CPU workload and delay-tolerance threshold.

5.1 Experiment setup

The application and scenario experiments are based on an environment depicted in Fig. 3. We use ZTE V880 smartphone in our experiments. It uses Android operating system in version 2.2, integrates with Wi-Fi interface and is capable of EDGE data connectivity. It has a Qualcomm MSM7227-1 CPU with 600MHz frequency, a 256MB memory and a battery capacity of 1250mAh at 3.7 volts. The mobile client part of Phone2Cloud, including resource monitor, bandwidth monitor, execution time predictor, offloading decision engine, local execution manager, and offloading proxy, will be run on it. As we can see from Fig. 3, we use Wi-Fi network to connect cloud in our experiments. As to the cloud,
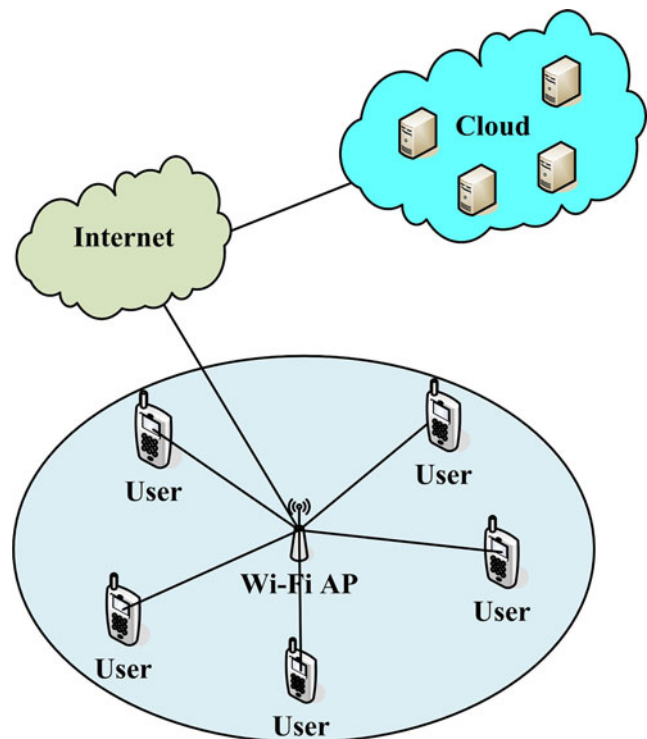

Fig. 3 Environment used in application experiments and scenario experiments

we use a desktop computer running Hadoop in version 0.20.2 in Linux operating system (Ubuntu 10.04) to serve as a cloud. It has a Pentium dual-core CPU with 2.60GHz frequency, a 2GB memory, a 320GB hard disk and a 100Mbps network interface. Besides, we run the cloud part of Phone2Cloud, i.e. remote execution manager, on the desktop computer.

We use three applications, as shown in Table 2, to do application experiments and analyze the energy consumption and execution time of three applications from the experimental data. The scenario experiments are based on a scenario that many students are using smartphones to learn an image process course via Internet. In our scenario experiments, we assume that they simply use an application running on smartphones, called *face finder*, to find the number of faces in a picture. Moreover, we should run the above four applications under different input sizes and CPU workloads as much as possible before experiments, so that we can get a large history log L used for execution time prediction described in Section 3.2.

We also utilize two experimental tools to record our experimental data and change CPU workload with minimal intrusion to our system respectively. One is called *PowerUsage* (Ding et al. 2011), which is used to measure power consumption of smartphones. It uses battery interfaces provided by Google APIs to record power consumption of smartphones. The other one is called *CPUChanger*, and we develop it to change CPU workload on demand.

5.2 Analysis of application experiments

We evaluate both energy consumption and execution time of three applications in Table 2 with respect to four factors. We will examine how these factors will affect energy consumption and execution time of the applications.

Table 3 describes the four factors' values in application experiments for evaluations of both energy consumption and execution time. For a specified factor, we evaluate its influence on both energy consumption and execution time of

Table 2 Applications used in experiments

No.	Application	Description
1	sort	Sort a given set of integer array elements by using Quick sort
2	path finder	Given a map and a source location (node), finds the shortest path tree with the source location as root by using Bellman-Ford algorithm
3	word count	Count the number of words from a block of text

three applications under different ranges, and other three factors are set to their default values.

5.2.1 Energy consumption

In this section, we show the connections between energy consumption of three applications and four factors including input size, bandwidth, CPU workload, and delay-tolerance threshold. As a matter of fact, most of computation of these applications can be offloaded to the cloud, so the data needed to be sent to the cloud is the input data of these applications, and it is reasonable that we use the maximum value $T_{idlemax}$ as the idle time on smartphone in Section 4.3.2. As we mentioned before, our system is not a fully automatic system, and it needs us to manually modify applications, so that their offloading parts can be run on the cloud and they can receive execution results returned from the cloud. Thus, there are three corresponding cloud-version applications.

Input size Figure 4 shows the results of the power consumption of three applications with different input sizes, based on the values in Table 3. In the figure, offloading (the green line) represents the power consumption of different three applications running in Phone2Cloud, which has the same indications in Figs. 5–7. As we can see from Fig. 4a, the energy consumed by *sort* running on cloud is much more than that on smartphone. The reason is that energy consumed by sorting on smartphone is much less than that by data transmission, including sending input data and receiving results. We also see that *sort* should be always directly run on smartphone. Due to the overhead of Phone2Cloud itself, such as the power consumption of making the offloading decision, Phone2Cloud consumes a little more power than running *sort* on smartphone, which can be seen in Fig. 4a. Moreover, overhead of Phone2Cloud can be seen among all figures.

On the contrary, the power consumption of running *path finder* on smartphone is much more than running it on cloud, as we can see from Fig. 4b. Because of the high complexity of finding a path in a map and relatively small data transmission, the energy consumed by searching paths in a map on smartphone is much more than transmitting data, and this leads to the above result. We also observe that *path finder* should be always offloaded to cloud. Moreover, we can see that our system save much energy for *path finder* under these circumstances.

As to *word count*, the result is interesting, as shown in Fig. 4c. When input size is smaller than 256KB, the application running on cloud costs more energy than running locally. After that, the power consumption on smartphone is over that on cloud. There is a reason for this result. When the input size is smaller than 256KB, transferring data costs more power than counting words. The more the input size

Table 3 Parameters in application experiments

Application	Input size		CPU workload		Delay-tolerance threshold		Bandwidth	
	Range (KB)	Default (B)	Range (%)	Default (%)	Range (ms)	Default (ms)	Range (KB/s)	Default (KB/s)
Sort	0~4000	40000			40~140			
Path finder	0~250	101,111	0~80	51.36	4000~22000	infinite	0~800	731.50
Word count	0~2000	524,337			2000~4000			

is, the more power is used to count words. We also see that *word count* should be run on smartphone when input size is smaller than 256KB and should be offloaded to cloud when input size is larger than 256KB. Therefore, we can save energy for *word count* when input size is larger than 256KB, and it increases as input grows.

Bandwidth Figure 5 describes how bandwidth affects three applications' energy consumption. For *sort* in Fig. 5a, the value of power consumption on cloud is always higher than that on smartphone, that is to say, *sort* should be always run on smartphone. The explanation for this is energy consumed by sorting computation on smartphone is less than that consumed by data transmission. However, energy consumption of *sort* running on cloud and smartphone are getting closer and closer with bandwidth increasing. Furthermore, the power consumed on smartphone keeps the same and we can also see this in Fig. 5b and c.

We can see from Fig. 5b that power consumption of *path finder* on smartphone is much more than that on cloud. The reason is the same as that of Fig. 4b. Additionally, the power consumption on cloud is decreasing with bandwidth increasing, and we see the same in Fig. 5c. So we should always offload *path finder* to cloud under such circumstances, and then we can save much energy for users.

The results in Fig. 5c are similar to those in Fig. 5a, but there is a difference. That is *word count* consumes more

energy on cloud compared with smartphone when bandwidth is below 600KB/s, while the power consumption on cloud is less than smartphone when bandwidth gets higher. The reason is that the power consumption of data transmission is getting less and less. Furthermore, we can reach that *word count* should be run on smartphone when bandwidth is below 600KB/s and offloaded to cloud when bandwidth is higher. Therefore, we can save energy for *word count* when the bandwidth is greater than 600KB/s, and the benefit enlarges as bandwidth increases.

CPU workload The power consumption of three applications under different CPU workloads is clearly shown in Fig. 6. In Fig. 6a, we can see that energy consumed by *sort* running on cloud is always more than running on smartphone, meaning the application should be always run on smartphone under these situations. Moreover, the power consumption of *sort* running on smartphone is getting more and more with CPU workload growing. As CPU workload increases, the energy consumption of smartphone for waiting execution results from cloud grows. However, the growth is too small to be seen in Fig. 6a. Furthermore, we can also see such situations in Fig. 6b and c. Figure 6b witnesses that power consumption of *path finder* running on smartphone is always much more than that on cloud, and it increases as CPU workload grows. Therefore, we should always offload *path finder* to cloud under these situations to
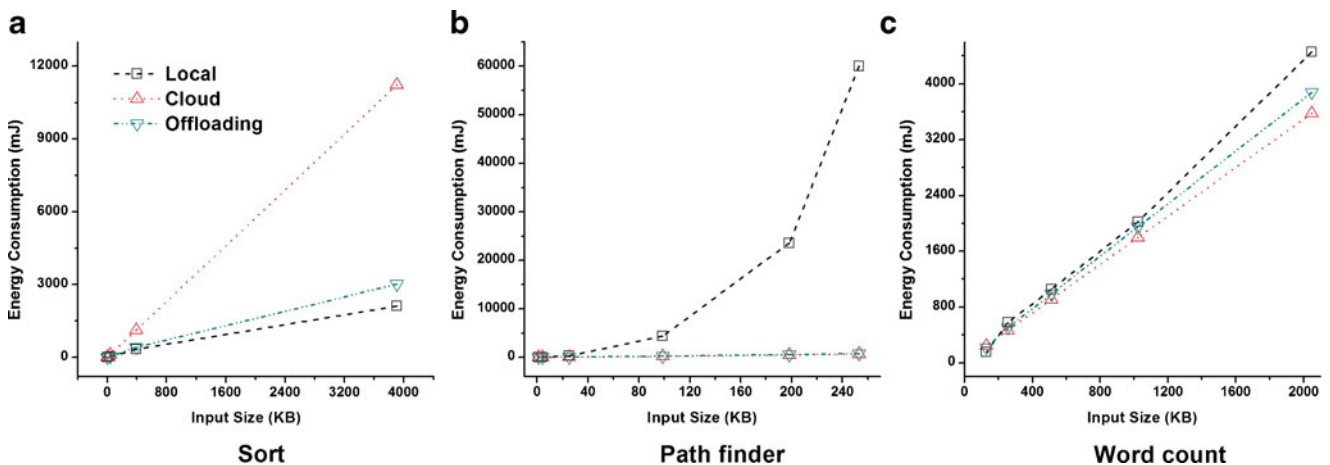
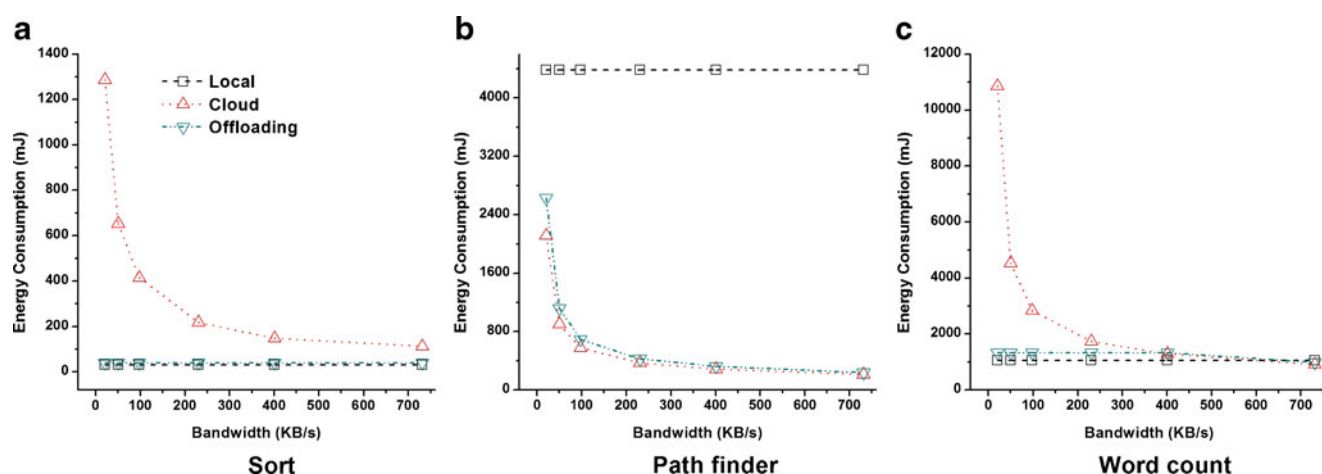**Fig. 4** Energy consumption of three applications under different input sizes

Fig. 5 Energy consumption of three applications under different bandwidths

save energy. However, Fig. 6c shows a very different situation compared with Fig. 6b when it comes to *word count*. Its power consumption on smartphone is less than that on cloud when CPU workload is below 25 %, and the opposite occurs when CPU workload is above 25 %. The reason is that energy consumption of *word count* running on smartphone grows with CPU workload increasing and energy consumption of running on cloud almost does not change. Therefore, *word count* should be offloaded to cloud when CPU workload is greater than 25 %, and we can save energy for users.

Delay-tolerance threshold Figure 7 illustrates the energy consumption of three applications under different delay-tolerance thresholds. We see that all of three applications experience the same phenomenon: the energy consumption of different applications running on cloud does not change

with delay-tolerance threshold growing as well as running on smartphone. For this phenomenon, we can explain that delay-tolerance threshold does not affect three applications' power consumption on cloud and smartphone. However, the power consumption of *sort* running in our system is changing as delay-tolerance grows in Fig. 7a. According to Fig. 4a, the power consumption of *sort* running in our system should be close to running on smartphone all the time. However, it is close to the power consumption of running on cloud when delay-tolerance threshold is smaller than 70 milliseconds, which means the execution time of *sort* running on smartphone is between 70 milliseconds and 90 milliseconds. Therefore, we should set delay-tolerance threshold to be greater than 90 milliseconds for *sort*. As shown in Fig. 7b and c, *path finder* and *word count* should be offloaded to cloud all the time; however, we can not know their execution time of running on smartphone.
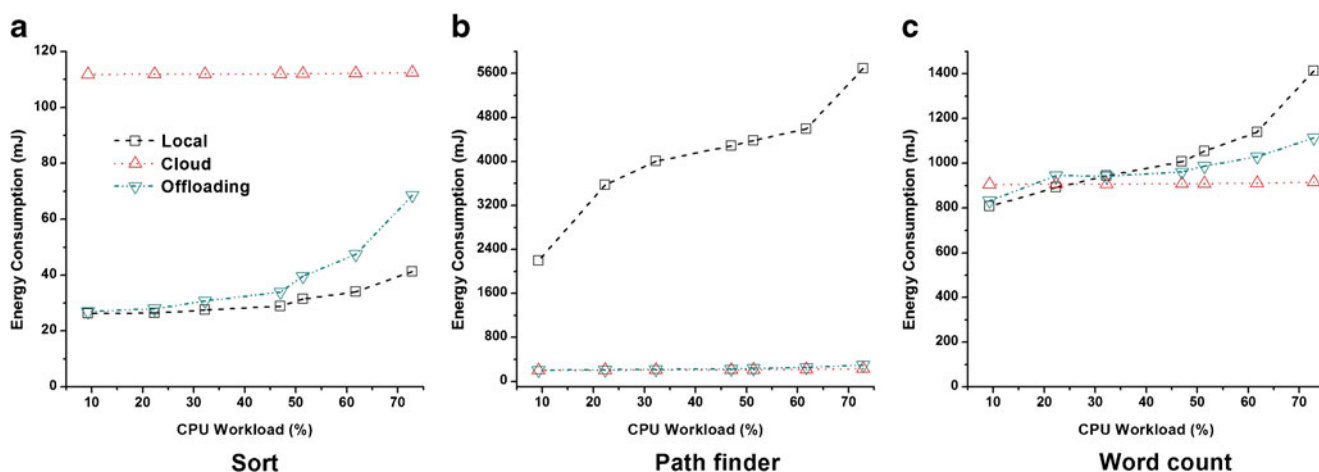

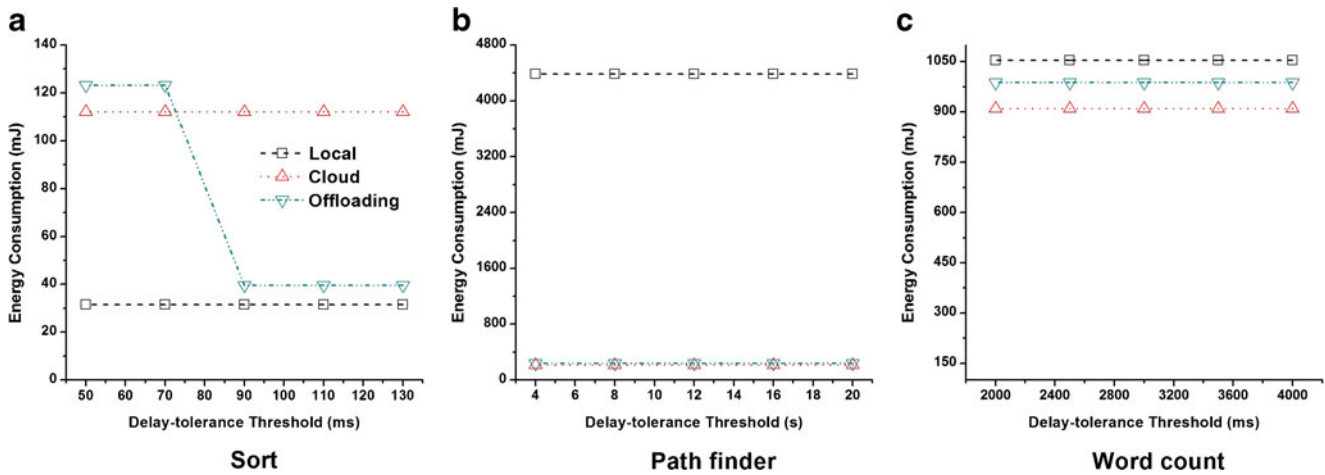**Fig. 6** Energy consumption of three applications under different CPU workloads

Fig. 7 Energy consumption of three applications under different delay-tolerance thresholds

5.2.2 Execution time

We describe the relationship between the execution time of three applications and four factors mentioned before in this section. The values of each factor for different applications are given in Table 3, and the results are shown in Figs. 8–10. In these figures, offloading (the green line) represents the execution time of different applications running in our Phone2Cloud.

Input size Figure 8a shows the execution time of *sort* under different input sizes. The value on cloud is more than that on smartphone, and the gap between them expands as input grows. The reason lies in that time cost by data transmission between cloud and smartphone is more than that cost by sorting on smartphone. Therefore, *sort* should be run on smartphone on this condition. Except for this, we see that the offloading one keeps the same as the local one. In

this case, we see our system can make a wise offloading decision.

For *path finder*, the execution time on smartphone over-weighs that on cloud, and the difference becomes more obvious as input increases, shown in Fig. 8b. As a result, we should run *path finder* on cloud. Besides, the execution time on cloud increases a little, but the change is too tiny to be seen in the figure. It can be seen that our system can make excellent decisions for *path finder*.

The wisdom of Phone2Cloud can also be proved by Fig. 8c. Due to the fact that execution time of *word count* on cloud is less than that on smartphone when input size is smaller than 256KB, we should run the application on cloud. While, the fact is that our Phone2Cloud runs *word count* on smartphone, and it is a wise decision. As we already know that user's delay-tolerance threshold is infinite, so the first task of our Phone2Cloud is to reduce the application's energy consumption. Furthermore, the power
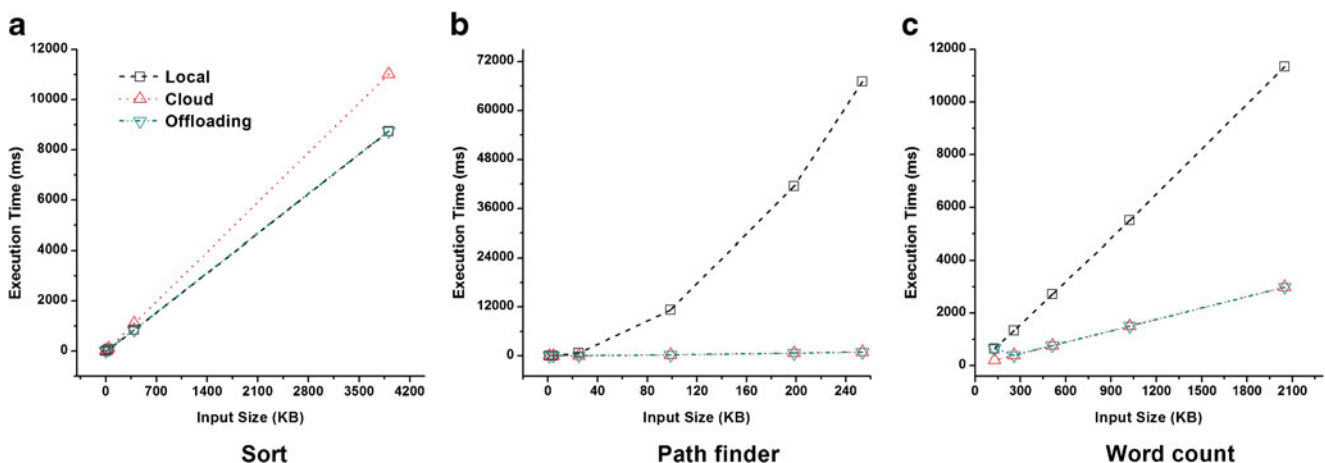

Fig. 8 Execution time of the three applications under different input sizes
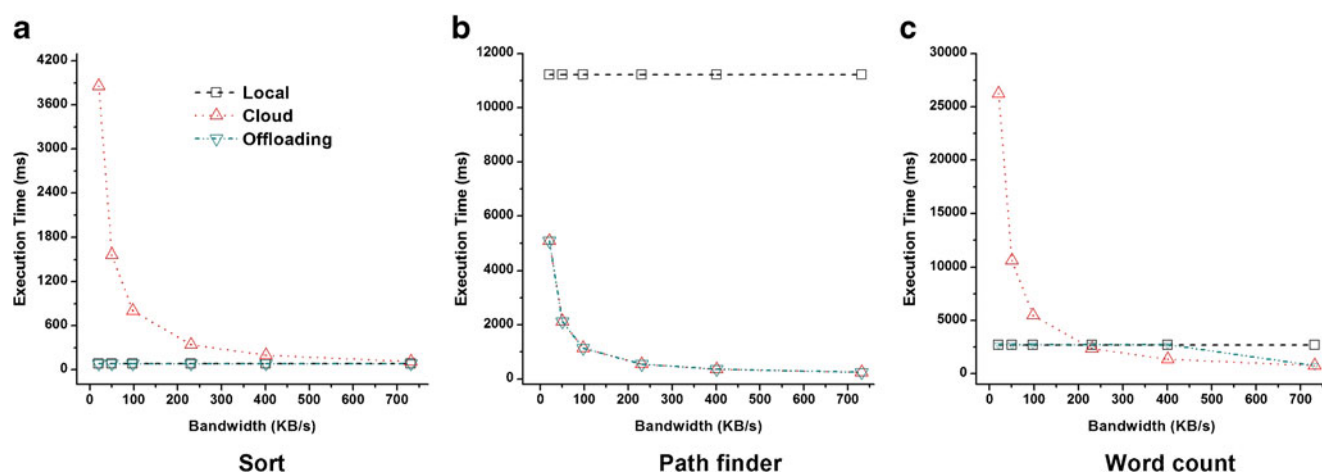
Fig. 9 Execution time of three applications under different bandwidths

cost on cloud is more than that on smartphone when input size is below 256KB in Fig. 4c. Therefore, Phone2Cloud decides to run *word count* on smartphone. When input size is greater than 256KB, our Phone2Cloud still makes a wise decision, thus user's experience is improved.

Bandwidth Figure 9 shows the relationship between execution time of three applications and bandwidth. The running on cloud always consumes more time than running locally, shown in Fig. 9a. The reason is that sorting on smartphone costs less time than data transmission. Moreover, the gap between running on cloud and locally becomes smaller as bandwidth grows. We can see from Fig. 9a that the execution time on Phone2Cloud almost keeps the same as that on smartphone, regardless of the overhead of Phone2Cloud. This shows our system always makes the right decision.

Figure 9b shows a different result comparing with Fig. 9a. *Path finder* spends more time on smartphone than on cloud. This is because the time of data transmission

is less than finding a path in a map on smartphone, and it decreases as bandwidth increases. Therefore our system chooses to offload *path finder* to cloud, and in this situation, we can save time for users.

For *word count*, the execution time on cloud is more than that on smartphone when bandwidth is less than 230KB/s, while the opposite occasion occurs when bandwidth is greater. But that does not mean we should offload the application to cloud even though it can save time for users. Considering the energy consumption of *word count* on cloud and smartphone, our system offloads *word count* to cloud only when the bandwidth exceeds 600KB/s, and it makes the right decision again.

CPU workload Figure 10 shows the execution time of *sort*, *path finder* and *word count* under different CPU workloads. A notable feature shared by the subfigures is that the execution time of three applications running on cloud does not change with CPU workload growing. For *sort* in
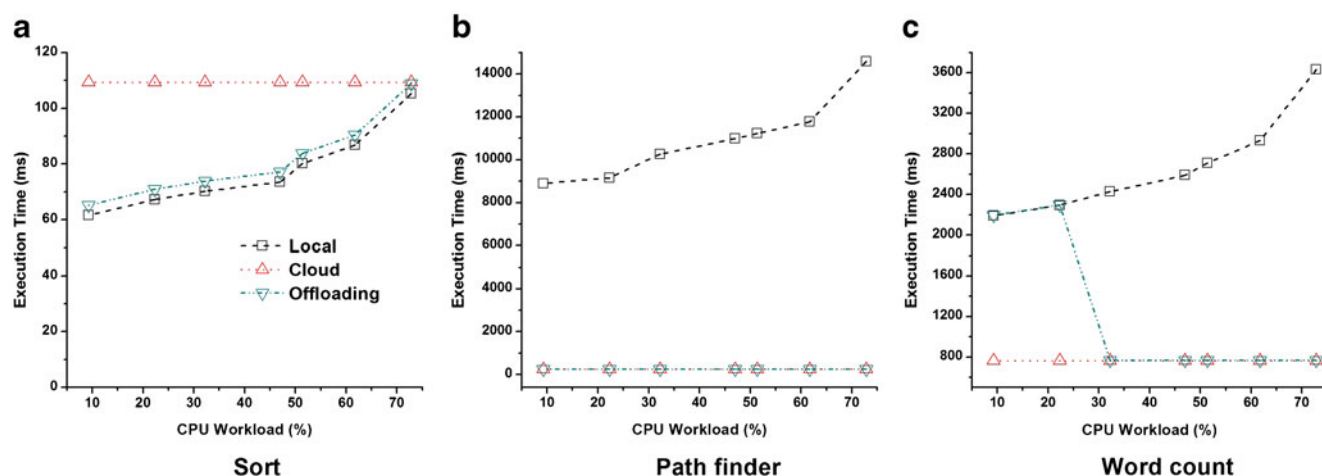

Fig. 10 Execution time of three applications under different CPU workloads

Fig. 10a, the time consumed on cloud is consistently higher than that on smartphone. Furthermore, the gap between running on smartphone and cloud reduces when CPU workload increases. However, the execution time of *path finder* on cloud keeps much less than on smartphone and this is quite different from *sort* (see Fig. 10b). Additionally, the execution time on smartphone grows as CPU workload increases, so we should always offload *path finder* to cloud under these situations to save time for users. As to *word count*, the results in Fig. 10c correspond to the power consumption in Fig. 6c. Due to the infinity of delay-tolerance threshold, users pay much attention to energy consumption, so Phone2Cloud runs *word count* locally when CPU workload is lower than 25 % from Fig. 6c. While CPU workload is greater than 25 %, it is offloaded to cloud to save energy and time, as shown in Fig. 10c.

Delay-tolerance threshold Figure 11 describes the relation between execution time and delay-tolerance threshold. We can see that the trend this figure shows is similar to Fig. 7. Because there is no connection between delay-tolerance threshold and execution time on smartphone and cloud, the execution time of each application on both cloud and smartphone does not change with delay-tolerance threshold growing. Besides, the execution time of *sort* in Phone2Cloud changes as delay-tolerance threshold increases, similar to the energy consumption in Fig. 7a. Thus, we get the conclusion that the execution time of *sort* on smartphone is about 80 milliseconds as shown in Fig. 11a. Therefore, delay-tolerance threshold should be larger than 90 milliseconds, to be precise, 80 milliseconds. In this way, time can be saved. Based on what Fig. 4b shows, we can understand that the energy consumption of *path finder* on smartphone is more than on cloud. Therefore, no matter how much delay-tolerance threshold is, our Phone2Cloud always offloads

this application to cloud. Specifically, the local execution time of *path finder* is around 11 seconds, while both cloud and offloading ones are less than 1 second, thus we achieve the goal of saving time for users. Fig. 11c shows very similar results to Fig. 11b. *Word count* is always offloaded to cloud in regardless of delay-tolerance threshold. Also, time can be saved by Phone2Cloud for *word count* as shown in Fig. 11c.

5.3 Analysis of scenario experiments

This section examines the energy consumption and execution time of *face finder* in scenario experiments under the aforementioned factors. The connections between energy consumption and factors will be discussed in Section 5.3.1, and the results on execution time will be presented in Section 5.3.2.

Table 4 shows each factor's range and default value in the scenario experiments for evaluations of both energy consumption and execution time. When we evaluate a specified factor's influence on both energy consumption and execution time of face finder under different ranges, the other three factors are set to their default values.

5.3.1 Energy consumption

Before discussing the results shown in Fig. 12, we need to manually modify *face finder* to be able to receive results from cloud, and there is also a corresponding cloud-version *face finder* on cloud. Since most of computation of *face finder* can be offloaded to the cloud, the required data of the cloud-version *face finder* is the input data of *face finder*, and it is also reasonable that we use the maximum value $T_{idlemax}$ to calculate the idle time on smartphone.
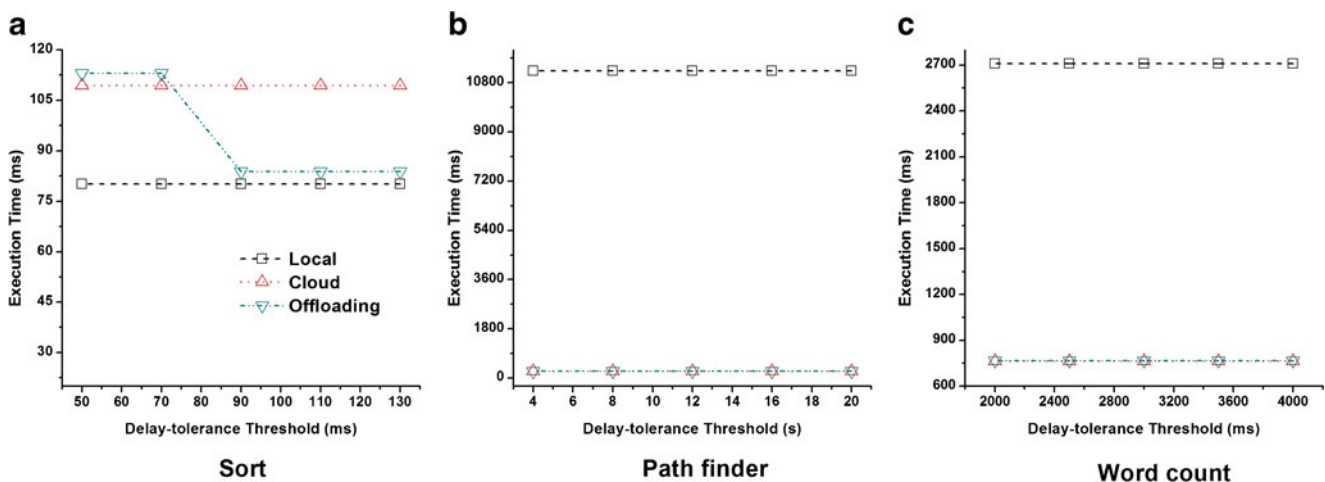

Fig. 11 Execution time of three applications under different delay-tolerance thresholds

Table 4 Parameters in scenario experiments

Factor	Input size	CPU workload	Delay-tolerance threshold	Bandwidth
Range	0∼1200 KB	20∼80 %	10∼50 s	0∼800 KB/s
Default	559,064 B	51.36 %	infinite	20.27 KB/s

Figure 12a shows that *face finder* costs more energy on smartphone than on cloud and the difference between them gets larger as input grows. This is because data transmission costs less energy than running the application locally. Furthermore, the local energy consumption grows faster than that on cloud. Therefore, *face finder* should be offloaded to cloud.

From Fig. 12b, we can see that *face finder* cost less power on smartphone than on cloud when bandwidth is lower than 50KB/s and the result changes to the opposite after bandwidth gets higher. The former is because the energy consumed by data transmission is very enormous when bandwidth is low, due to the poor power efficiency of Wi-Fi. Similar to what Fig. 9a shows, the power consumption on smartphone has no relation to bandwidth, and the power consumed on cloud decreases with bandwidth growing. As a consequence, we should offload *face finder* to cloud when bandwidth is greater than 50KB/s for energy saving.

As to the effect of CPU workload, Fig. 12c shows that energy consumption of *face finder* on cloud is always much less than running locally, and the gap between them grows

as CPU workload increases. So *face finder* should be offloaded to cloud, which is what our system does.

For the delay-tolerance threshold, it nearly affects nothing on *face finder* on both smartphone and cloud as shown in Fig. 12d. According to energy consumption of *face finder*, Phone2Cloud should run *face finder* on smartphone when the threshold is smaller than 30 seconds. However, it offloads the application to cloud, meaning the execution time on smartphone can not meet user's threshold. So the local execution time is between 20 seconds and 30 seconds. Therefore, to save power, delay-tolerance threshold should be greater than 30 seconds.

5.3.2 Execution time

Figure 13 describes the execution time of *face finder* under the four factors. The subfigures in Fig. 13 illustrate the connection between execution time and the four factors respectively. To be specific, the execution time on smartphone is not only more than, but also grows faster than that on cloud, as shown in Fig. 13a. Therefore, Phone2Cloud offloads *face finder* to cloud all the time, meaning it makes a wise offloading decision.

As clearly shown in Fig. 13b, we can see the execution time experiences almost the same trend with the energy consumption in Fig. 12b, caused by the inefficient data transmission under low bandwidth. Furthermore, the execution time on cloud decreases with bandwidth ascending and there is no relationship between local execution time
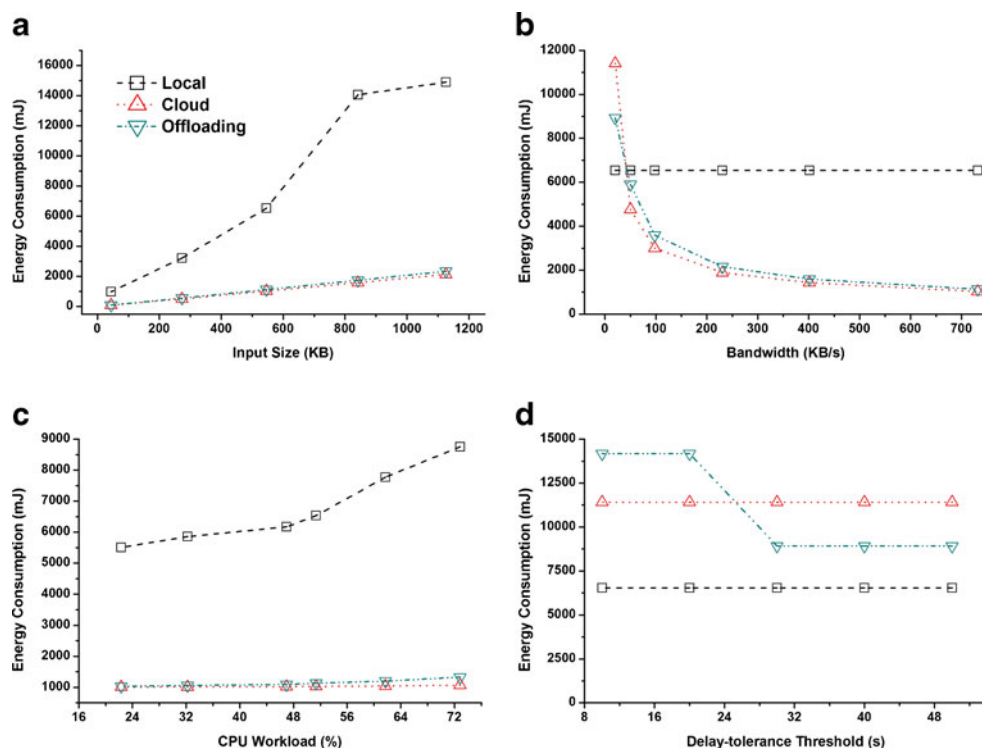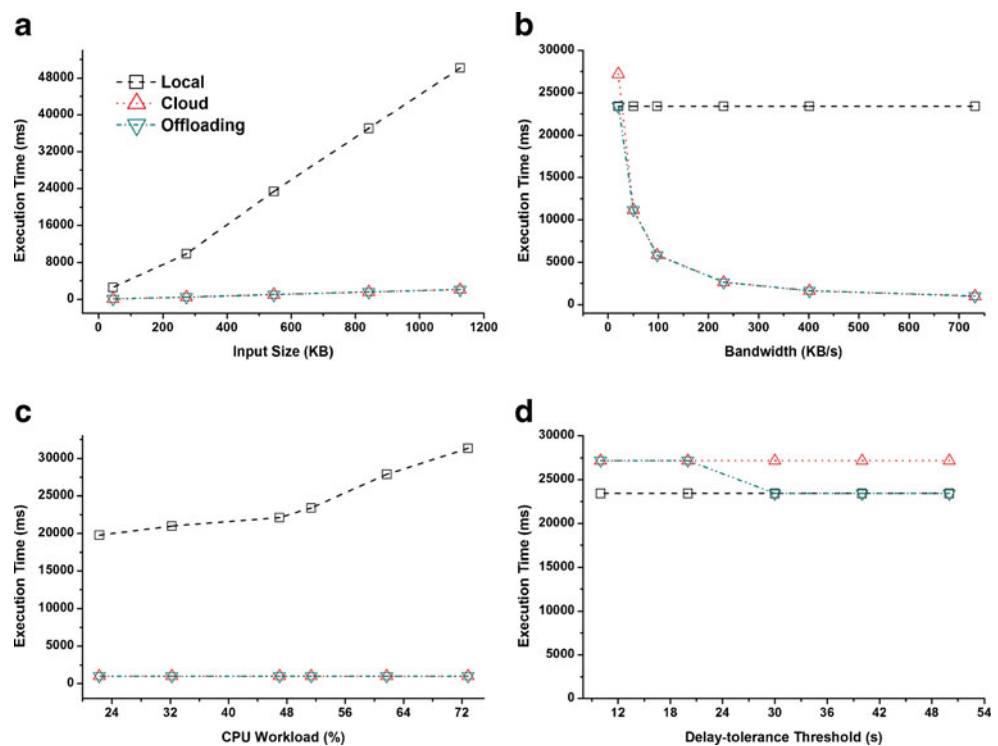
Fig. 12 Energy consumption of face finder under four factors

Fig. 13 Execution time of face finder under four factors

and bandwidth. So *face finder* should be offloaded to cloud when bandwidth is greater than 50KB/s, as we stated in Fig. 12b.

In Fig. 13c, we also see the similar trend shown in Fig. 12c, with the only difference that the execution time on cloud is a straight line, meaning CPU workload influences nothing on execution time on cloud. Then in this case, our Phone2Cloud makes an excellent offloading decision, that is, it offloads *face finder* to cloud, and saves time for users.

As regard to delay-tolerance threshold, Fig. 13d validates our thought again, which is the threshold does not correlate to the execution time on both cloud and smartphone. The execution time in Phone2Cloud changes with delay-tolerance threshold increasing. Ignoring the threshold, *face finder* should be run on smartphone according to its energy consumption in Fig. 12b. However, Phone2Cloud offloads it to cloud when delay-tolerance threshold is smaller than 25 seconds. This means the execution time of *face finder* on smartphone is between 20 seconds and 30 seconds, as what we got from Fig. 12d. Actually, it is about 23 seconds shown in Fig. 13d. Therefore, we should set delay-tolerance threshold to be greater than 23 seconds to save time for users.

6 Conclusion

This paper has presented an energy-efficient mobile cloud computing system called Phone2Cloud, which takes advantage of the computation offloading paradigm. It is able to save energy and improve applications' performance and users' experience of smartphones. Two sets of experiments are conducted and the results demonstrate that our system is of great effectiveness. For the sake of simplicity, we simply use a naive approach to predict application's execution time and only take CPU workload and input size into account. There are many alternative methods and factors we can take into consideration, and we will compare these methods with our own approach in the future work. Moreover, as we mentioned before our system is a semi-automatic offloading system. Further studies are demanded to make our system fully automatic.

References

Ananthanarayanan, G., & Stoica, I. (2009). Blue-Fi: enhancing Wi-Fi performance using bluetooth signals. In *Proceedings ACM MobiSys'09* (pp. 249–261). Krakow.

Aggarwal, B., Chitnis, P., Dey, A., Jain, K., Navda, V., Padmanabhan, V.N., Ramjee, R., Schulman, A., Spring, N. (2010). Stratus: energy-efficient mobile communication using cloud support. In *Proceedings ACM SIGCOMM'10* (pp. 477–478). New Delhi.

Balasubramanian, A., Mahajan, R., Venkataramani, A. (2010). Augmenting mobile 3G using WiFi. In *Proceedings ACM MobiSys'10* (pp. 209–222). Chicago.

Brakmo, L.S., Wallach, D.A., Viredaz, M.A. (2004). μSleep: a technique for reducing energy consumption in handheld devices. In *Proceedings MobiSys'04* (pp. 12–22). Boston.

Burgstahler, L., & Neubauer, M. (2002). New modifications of the exponential moving average algorithm for bandwidth estimation. In *Proceedings of the 15th ITC specialist seminar*.

Chen, G., Kang, B.T., Kandemir, M., Vijaykrishnan, N., Irwin, M.J., Chandramouli, R. (2004). Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices. *IEEE Transactions on Parallel and Distributed Systems*, *15*(9), 795–809.

Chen, S., Liu, Y., Gorton, I., Liu, A. (2005). Performance prediction of component-based applications. *Journal of Systems and Software*, *74*(1), 35–43.

Chun, B.G., & Maniatis, P. (2010). Dynamically partitioning applications between weak devices and clouds. In *Proceedings of the 1st ACM workshop on mobile cloud computing & services: social networks and beyond (MCS'10)* (p. 7). San Francisco: ACM Press.

Cuervo, E., Balasubramanian, A., Cho, D.K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P. (2010). MAUI: making smartphones last longer with code offload. In *Proceedings ACM MobiSys'10* (pp. 49–62). San Francisco.

Ding, F., Xia, F., Zhang, W., Zhao, X., Ma, C. (2011). Monitoring energy consumption of smartphones. In *Proceedings of 2011 international conference on internet of things and 4th international conference on cyber, physical and social computing (iThings/CPSCom)* (pp. 610–613). Dalian.

Engblom, J., Ermedahl, A., Sjodin, M., Gustafsson, J., Hansson, H. (2003). Worst-case execution-time analysis for embedded real-time systems. *International Journal on Software Tools for Technology Transfer*, *4*(4), 437–455.

Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., Milojicic, D. (2004). Adaptive offloading for pervasive computing. *IEEE Pervasive Computing*, *3*(3), 66–73.

Heikkinen, M.V., Nurminen, J.K., Smura, T., Hammainen, H. (2012). Energy efficiency of mobile handsets: measuring user attitudes and behavior. *Telematics and Informatics*, *29*(4), 387–399.

Jongerden, M.R., & Haverkort, B.R. (2009). Which battery model to use? *IET Software*, *3*(6), 445–457.

Ju, Y., Min, C., Lee, Y., Yu, J., Song, J. (2012). An efficient dataflow execution method for mobile context monitoring applications. In *Proceedings IEEE PerCom'12* (pp. 116–121). Lugano.

Kemp, R., Palmer, N., Kielmann, T., Bal, H. (2010a). Cuckoo: a computation offloading framework for smartphones. *Mobile Computing, Application, and Services, 76*, 59–79.

Kemp, R., Palmer, N., Kielmann, T., Bal, H. (2010b). The smartphone and the cloud: power to the user. *Mobile Computing, Application, and Services, 76*, 342–348.

Kemp, R., Palmer, N., Kielmann, T., Bal, H. (2012). Energy efficient information monitoring applications on smartphones through communication offloading, lecture notes of the institute for computer sciences. *Social Informatics and Telecommunications Engineering*, *95*, 60–79.

Kim, J., Kang, K., Shim, H., Hwangbo, W., Kyung, C.M. (2007). Fast estimation of software energy consumption using IPI (Inter-Prefetch Interval) energy model. In *Proceedings of IFIP WG 10.5 international conference on very large scale integration of system-onchip* (pp. 224–229). Atlanta.

Kirner, R., Kadlec, A., Puschner, P. (2009). Precise worst-case execution time analysis for processors with timing anomalies. In *Proceedings 21st euromicro conference on real-time systems (ECRTS)* (pp. 119–128). Dublin.

Kosta, S., Aucinas, A., Pan, H., Mortier, R., Zhang, X. (2012). ThinkAir: dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proceedings INFOCOM* (pp. 945–953). Orlando.

Kumar, K., & Lu, Y.H. (2010). Cloud computing for mobile users: can offloading computation save energy? *Computer*, *43*(4), 51–56.

Liu, J., Kumar, K., Lu, Y.H. (2010). Tradeoff between energy savings and privacy protection in computation offloading. In *ACM/IEEE international symposium on low power electronics and design* (pp. 213–218).

Mahmud, K., Inoue, M., Murakami, H., Hasegawa, M., Morikawa, H. (2005). Energy consumption measurement of wireless interfaces in multi-service user terminals for heterogeneous wireless networks. *IEICE Transactions on Communications*, *88*(3), 1097–1110.

Miettinen, A.P., & Nurminen, J.K. (2010). Energy efficiency of mobile clients in cloud computing. In *Proceedings of the 2nd USENIX conference on hot topics in cloud computing* (pp. 4). Berkeley.

Nimmagadda, Y., Kumar, K., Lu, Y.H., Lee, C.S.G. (2010). Real-time moving object recognition and tracking using computation offloading. In *Proceedings IEEE international conference on intelligent robots and systems* (pp. 2449–2455). Taipei.

Ou, S., Yang, K., Liotta, A. (2006). An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems. In *Proceedings IEEE PerCom'06* (pp. 116–125). Pisa.

Palit, R., Singh, A., Naik, K. (2008). Modeling the energy costs of applications on portable devices. In *The 11th ACM international conference on modeling, analysis, and simulation of wireless and mobile systems* (pp. 346–353). Vancouver.

Pering, T., Agarwal, Y., Gupta, R., Want, R. (2006). Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proceedings ACM MobiSys'06* (pp. 220–232). Uppsala.

Rozner, E., Navda, V., Ramjee, R., Rayanchu, S. (2010). NAP-man: network-assisted power management for WiFi devices. In *Proceedings ACM MobiSys'10* (pp. 91–106). San Francisco.

Schoeberl, M., Puffitsch, W., Pedersen, R.U., Huber, B. (2010). Worst-case execution time analysis for a Java processor. *Software: Practice and Experience*, *40*(6), 507–542.

Sheng, X., Tang, J., Zhang, W. (2012). Energy-efficient collaborative sensing with mobile phones. In *Proceedings IEEE INFOCOM* (pp. 1916–1924). Orlando.

Shih, E., Bahl, P., Sinclair, M.J. (2002). Wake on wireless: an event driven energy saving strategy for battery operated devices. In *Proceedings ACM MobiCom'02* (pp. 160–171). Atlanta.

Thimmarayaswamy, K., Dsouza, M.M., Varaprasad, G. (2011). Low power techniques for an android based phone. *ACM SIGARCH Computer Architecture New*, *39*(2), 26–35.

Wen, Y., Zhang, W., Luo, H. (2012). Energy-optimal mobile application execution: taming resource-poor mobile devices with cloud clones. In *Proceeding INFOCOM* (pp. 2716–2720). Orlando.

Xian, C., Lu, Y.H., Li, Z. (2007). Adaptive computation offloading for energy conservation on battery-powered systems. In *International conference on parallel and distributed systems* (pp. 1–8).

Xiao, Y., Bhaumik, R., Yang, Z., Siekkinen, M., Savolainen, P., Yla-Jaaski, A. (2010). A system-level model for runtime power estimation on mobile devices. In *2010 IEEE/ACM international conference on green computing and communications (GreenCom) & 2010 IEEE/ACM international conference on cyber, physical and social computing (CPSCom)* (pp. 27–34). Piscataway: IEEE Press.

Zhang, T., Madhani, S., Gurung, P., van den Berg, E. (2005). Reducing energy consumption on mobile devices with WiFi interfaces. In *Proceedings IEEE Globecom'05* (pp. 561–565). St. Louis.

Zhou, R., Xiong, Y., Xing, G., Sun, L., Ma, J. (2010). ZiFi: wireless LAN discovery via ZigBee interference signatures. In *Proceedings ACM MobiCom'10* (pp. 49–60). Chicago.

Feng Xia is an Associate Professor and PhD Supervisor in School of Software, Dalian University of Technology, China. He is the (Guest) Editor of several international journals. He serves as General Chair, PC Chair, Workshop Chair, Publicity Chair, or PC Member of a number of conferences. Dr. Xia has authored/co-authored one book and over 140 scientific papers in international journals and conferences. His research interests include social computing, mobile computing, and cyber-physical systems. He is a Senior Member of IEEE, IEEE Computer Society, IEEE SMC Society, and a member of ACM and ACM SIGMobile.

Fangwei Ding received his Bachelor degree from Dalian University of Technology (DUT), China, in 2010. He is currently a Master student in School of Software, Dalian University of Technology. His research interests include mobile cloud computing, green computing and communications.

Jie Li received his Bachelor degree in Network Engineering from Dalian University of Technology (DUT), China, in 2011. He is currently a Master student in School of Software, Dalian University of Technology. His research interests include mobile social networks, socially-aware networking, and ad hoc networks.

Xiangjie Kong received the Ph. D. degree from Zhejiang University, Hangzhou, China in 2009. Currently, he is an assistant professor in Dalian University of Technology, Dalian, China. His current research interests include urban road traffic modeling and control technology, cyber physical systems and complex networks. He is a member of IEEE.

Laurence T. Yang received the BE degree in computer science and technology from Tsinghua University, China and the PhD degree in computer science from University of Victoria, Canada. He is a professor in the School of Computer Science and Technology at Huazhong University of Science and Technology, China, and in the Department of Computer Science, St. Francis Xavier University, Canada. His research interests include parallel and distributed computing, embedded and ubiquitous/pervasive computing. His research has been supported by the National Sciences and Engineering Research Council, and the Canada Foundation for Innovation.

Jianhua Ma is a professor of the Department of Digital Media in Faculty of Computer and Information Sciences at Hosei University, Japan. Previously, he had 15 years working experience at NUDT, Xidian University and University of Aizu (Japan). His research interests include multimedia, networks, ubiquitous computing, social computing, and cyber intelligence. He has published over 200 papers, and edited over 20 books/proceedings and over 20 journal special issues. He is a founder of IEEE Int'l Conf. on Ubiquitous Intelligence and Computing (UIC), IEEE Conf. on Cyber, Physical and Social Computing (CPSCom), and IEEE Conf. on Internet of Things (iThings).