Chen et al. Journal of Cloud Computing: Advances, Systems and Applications (2017) 6:1
DOI 10.1186/s13677-016-0071-y

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

Framework for context-aware computation offloading in mobile cloud computing

Xing Chen[1,2], Shihong Chen[1,2], Xuee Zeng[1,2], Xianghan Zheng[1,2*], Ying Zhang[3] and Chunming Rong[4]

Abstract

Computation offloading is a promising way to improve the performance as well as reducing the battery power consumption of a mobile application by executing some parts of the application on a remote server. Recent researches on mobile cloud computing mainly focus on the code partitioning and offloading techniques, assuming that mobile codes are offloaded to a prepared server. However, the context of a mobile device, such as locations, network conditions and available cloud resources, changes continuously as it moves throughout the day. And applications are also different in computation complexity and coupling degree. So it needs to dynamically select the appropriate cloud resources and offload mobile codes to them on demand, in order to offload in a more effective way. Supporting such capability is not easy for application developers due to (1) adaptability: mobile applications often face changes of runtime environments so that the adaptation on offloading is needed. (2) effectiveness: when the context of the mobile device changes, it needs to decide which cloud resource is used for offloading, and the reduced execution time must be greater than the network delay caused by offloading. This paper proposes a framework, which supports mobile applications with the context-aware computation offloading capability. First, a design pattern is proposed to enable an application to be computation offloaded on-demand. Second, an estimation model is presented to automatically select the cloud resource for offloading. Third, a framework at both client and server sides is implemented to support the design pattern and the estimation model. A thorough evaluation on two real-world applications is proposed, and the results show that our approach can help reduce execution time by 6–96% and power consumption by 60–96% for computation-intensive applications.

Keywords: Computation Offloading, Mobile Cloud Computing, Context-aware

Introduction

Current estimates indicate the number of mobile devices (mostly smart phones, tablets, and laptops) around three or four billions. This number is expected to grow to a trillion in a few years [1, 2]. At the same time, hundreds of thousands of developers have produced more than millions of applications [3]. Following the fast improvement of smartphone hardware and increased user experience, applications try to provide more and more functionality and then they inevitably become so complex as to make the two most critical limits of mobile devices worse. The first limit is the diversity of hardware configurations, which gives users very different experiences even running the same applications [4]. Generally speaking, the low hardware configuration of a mobile device implies the low performance of the application running on the device, and gives a poor experience to the user. The second limit is the battery power. Complex applications usually have intensive computations and consume a great deal of energy [5, 6]. Although the battery capacity keeps growing continuously, it still cannot keep pace with the growing requirements of mobile applications [7].

Computation offloading is a popular technique to help improve the mobile application performance and meanwhile reduce its power consumption [8–13]. Offloading, also referred to as remote execution, is to make some computation intensive code of an application executed on a remote server, so that the application can take advantage of the powerful hardware and the sufficient power supply for increasing its responsiveness and decreasing its battery power consumption.

* Correspondence: xianghan.zheng@fzu.edu.cn
[1]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China
[2]Fujian Provincial Key Laboratory of Networking Computing and Intelligent Information Processing, Fuzhou University, Fuzhou 350108, China
Full list of author information is available at the end of the article

Recent researches on computation offloading mainly focus on the code partitioning and offloading techniques, assuming that mobile codes are offloaded to a prepared server or a predefined Cloud [14–18]. But, in practice, it needs to offload computation in a dynamic manner. On one hand, applications are different in computation complexity and coupling degree. So mobile codes need to be executed on different places even if the device contexts are the same. For instance, when the mobile device uses a remote cloud with a poor network connection, some computation-intensive codes need to be offloaded to the server, but others are better executed locally. On the other hand, the context of a mobile device, such as locations, networks and available computing resources, changes continuously as it moves throughout the day. So it needs to select an appropriate cloud resource for offloading according to the device context. For instance, when a mobile device has a good network connection, it can use a remote cloud to outsource the computation-intensive tasks, in order to improve application performance and user experience. When the mobile device suffers from a poor network connection, it can use a nearby cloudlet [19] instead.

In order to offload in a more effective way, it needs to dynamically select the appropriate cloud resources and offload mobile codes to them on demand. However, it is not an easy matter for application developers to support such capability, which can be broken into two parts.

First, mobile applications often face changes of runtime environments so that the adaptation on offloading is needed. It is usually decided at runtime whether mobile codes need to be offloaded and which parts should be executed remotely [20]. For instance, if the remote server becomes unavailable due to unstable network connection, the computation executed on the server should come back to the device or go to another available server on the fly.

Second, when the device context changes, it needs to decide which cloud resource is used for offloading. The processing power of cloud resource decides the time spent on the server-side processing. The quality of network connections between the device and cloud resources decides the time spent on network communication. The computation complexity and coupling degree of the application should be also considered to assist in making a decision and the trade-off is between the reduced execution time and the network delay.

In this paper, we present a framework for context-aware computation offloading, in order to dynamically select the appropriate cloud resources and offload mobile codes to them on demand, according to the device context. Our paper makes three major contributions:

- We propose a design pattern, which implements the adaptation on offloading and enables a mobile application to offload computation on the fly.
- We present an estimation model, which calculates the reduced execution time and the network delay, and selects the appropriate cloud resource for offloading.
- We implement a framework, which supports the design pattern and the estimation model as mentioned above.

We successfully applied our framework to support context-aware computation offloading on two real-world applications. The thorough evaluation is proposed and the results show that our approach can help reduce execution time by 6–96% and power consumption by 60–96% for computation-intensive applications.

We organize the rest of this paper as follows. Section Approach overview overviews our approach with a motivating example. Section Design pattern for computation offloading presents the design pattern for computation offloading. Section Estimation model illustrates the estimation model for cloud resource selection. Section Implementation gives the implementation of the framework. Section Evaluation reports the evaluation on two real-world applications. Section Related work introduces related work and we conclude the paper in Section Conclusion and future work.

Approach overview

In this section, we give an overview about our approach. We first present a motivating example, which interprets requirements for context-aware computation offloading. Then we discuss how to leverage resources to offload computation in a traditional way. Finally, we briefly introduce our novel framework for context-aware computation offloading.

A motivating example

The example is shown in Fig. 1. The upper part shows a real-world Android application, Gobang, which is a mobile game. There are three difficulty levels of games and they are implemented by three algorithms with different computation complexities. The high-level game is a computation-intensive task, which needs to be offloaded to a remote server, while the low-level game is better executed locally in most of the time. The lower part shows that the context of the smartphone changes continuously as it moves throughout the day. There are several available cloud resources, such as the public cloud service and the library cloudlet, which can be used for offloading under different device contexts. For instance, in the library, the smartphone can use the WIFI connection to access the library cloudlet, the lab cloudlet or the public cloud service, while it can just use the 3G connection to access the public cloud service outside the buildings. In order to improve application performance

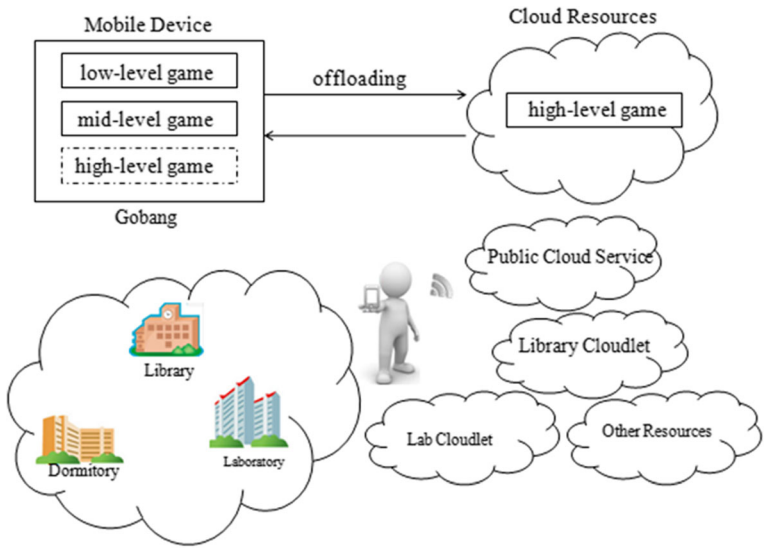

Fig. 1 The motivating example

and user experience, it needs to decide which cloud resource is used for offloading according to the device context, and then offload mobile codes to it on the fly.

Context-aware computation offloading

In order to offload in a more effective way, it needs to dynamically select the appropriate cloud resources and offload mobile codes to them on demand. It is not an easy matter for our motivating example, which mainly comes from the following two aspects:

On one hand, it needs to be decided at runtime whether mobile codes need to be offloaded and which parts should be executed remotely. So when the smartphone moves from one place to another, it needs to re-configure the application and deploy parts of the application onto the resource before using it for offloading.

On the other hand, it needs to dynamically select the appropriate cloud resource for offloading, according to the device context. There are several available cloud resources that are different in the processing power and the quality of network connection. So it will take a long time to collect runtime status for comparison, and the trade-off is between the reduced execution time and the network delay.

Our framework

The core idea of our approach is that the issues above can be solved automatically by the underlying framework. We propose a design pattern and an estimation model, and implement a framework to support them.

The design pattern can support an application to be offloaded on demand. The application is developed in the service style and its different parts are able to be executed locally or remotely. The service pool is introduced to enable the application to use several cloud resources at the same time, and improve the availability of offloading service.

The estimation model is used to automatically select the appropriate cloud resource for offloading, according to the device context. We model computation tasks, network connections and processing power of cloud resources, and propose an algorithm to calculate the reduced execution time and the network delay, which uses the history data as well as the device context.

The framework is implemented to support the design pattern and the estimation model. On one hand, all parts of the application, which are developed in the service style, are able to be executed locally or remotely, through the adapters at both client and server sides. The service pool at the client side maintains a list of available services for each part of the application. On the other hand, run data of computation tasks, network connections and processing power of cloud resources, is continually collected and stored. The history data is modeled at the client side and used in the estimation model.

Design pattern for computation offloading

A design pattern and a runtime mechanism are proposed in this section. The design pattern enables a mobile application to be computation offloaded. And the runtime mechanism supports the application to be offloaded on the fly.

Design pattern

In our approach, mobile applications need to be developed, following a design pattern. As shown in Fig. 2, the design pattern consists of a main module which must

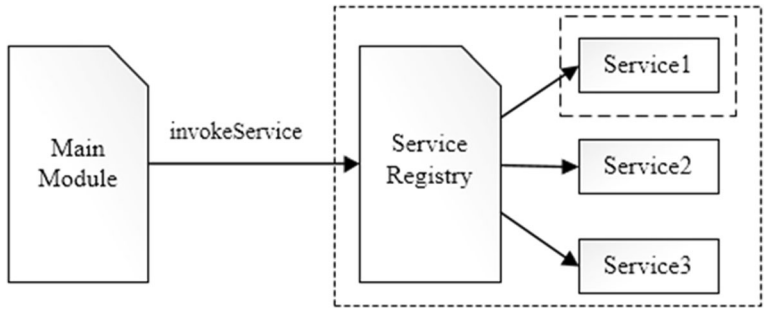

Fig. 2 The design pattern for offloading

stay on the device, and a mobile module which contains a set of movable services.

When developing an application, developers first need to classify the methods into two categories: Anchored and Movable. The anchored methods are included in the main module, which can be divided into three types: 1) methods that implement the application's user interface; 2) methods that interact with I/O devices, such as the accelerometer and the GPS; 3) methods that interact with external services, such as e-commerce transaction. Besides these anchored methods, other methods are movable, which can run on either the mobile device or remote cloud resources. The movable methods are encapsulated in the service style and included in the service registry.

These services can be invoked by the main module, based on a dynamic proxy mechanism. And it is transparent to the developers whether the service is executed locally or remotely. As shown in Fig. 3, the main module does not invoke the services directly, but sends a request to the *adapter* object instead. The *adapter* object is a dynamic proxy which is used to invoke the services in the service registry. It judges which service to call and decides whether to offload.

```
/*
 * function: an example for the invocation of Service1
 */
public static void main(String[] args)
{
    ...
    Adapter adapter = Adapter.getInstance();
    outputParam = (int[]) adapter.invokeService(
        {"serviceName": "Service1",
         "inputParams": ["100000", "110000", "113000",
                         "112200", "112100"]
        });
    ...
}
```

Fig. 3 The example code snippet for service invocation

Runtime mechanism

The runtime mechanism supports the adaptation on offloading. It allows the main module to effectively invoke the services in the service registry, no matter they are deployed on the client side or other cloud resources across the network. The core of the mechanism is composed of three parts: the local adapter, the remote adapter and the service pool, as shown in Fig. 4.

For each service in the service registry, there is a service pool. It consists of multiple candidate services that provide the same function but are deployed on different places. The services of the service pool can be added or removed dynamically. And the service descriptions, including their names, addresses, protocols and qualities, are recorded by the service pool. For instance, in Fig. 4, there are three services in the service pool. The three services have the same function, and they are separately deployed on Google App Engine, a nearby cloudlet and the client side. In addition, the service pool acts as a virtual service. When comes a request, the service pool will give preference to the service with best quality and return the corresponding service description.

The adapters are used to invoke the local and remote services. When comes a request, the local adapter can get the description of the service with best quality from the service pool. If the service is a local one, the local adapter invokes the service directly. And if the service is a remote one, the local adapter forwards the request to the remote adapter that is running on the remote server, across the network. Then the remote adapter invokes the remote service and sends the result back to the client side. In addition, if the device context changes during remote service invocation, the client side may fail to receive the result and another invocation needs to be executed.

Estimation model

The estimation model consists of the information models and the selection algorithm. The information models are used to calculate the reduced execution time and the network delay. And the selection algorithm is used to decide the appropriate cloud resource for offloading.

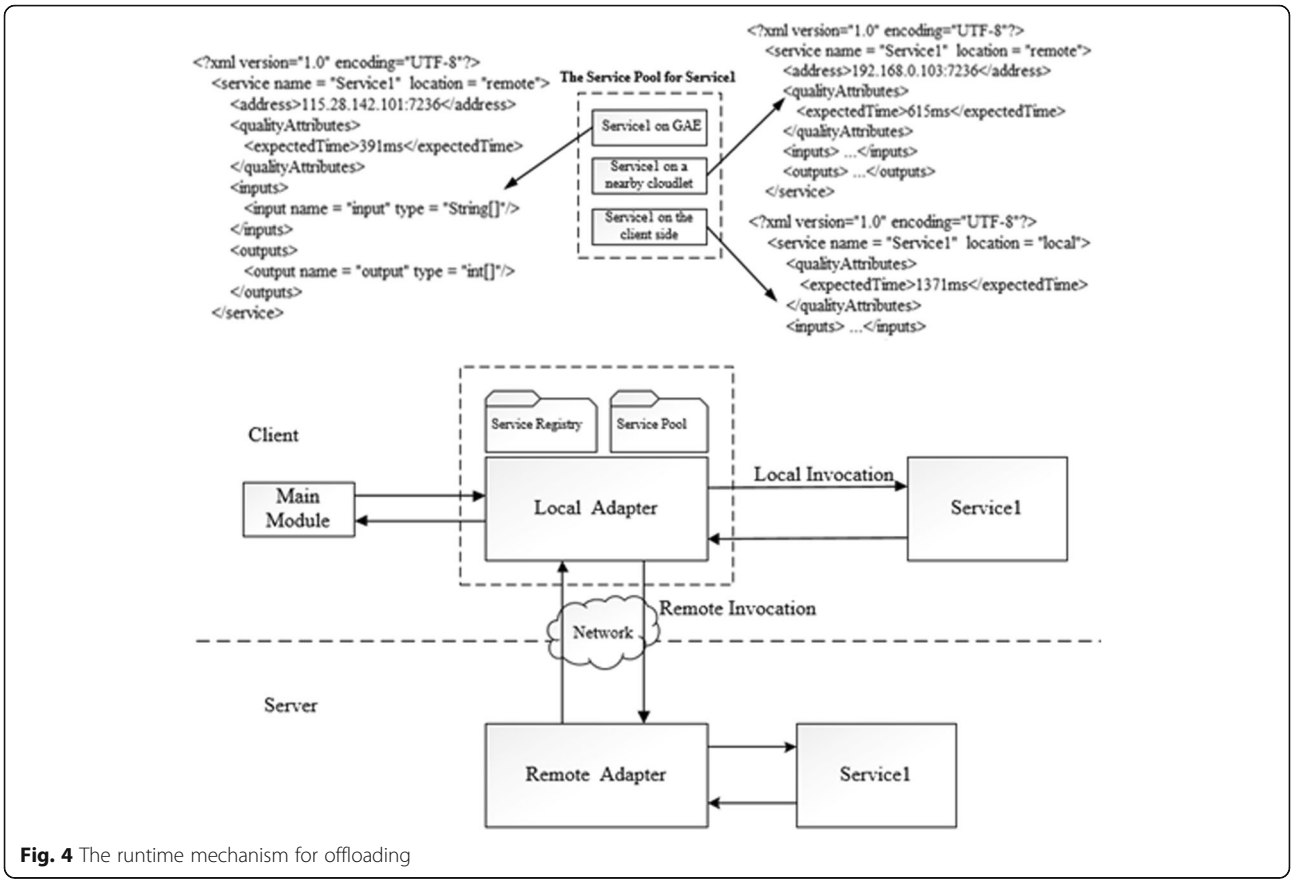

Fig. 4 The runtime mechanism for offloading

Information models

The response time for remote service invocation consists of the server time and the network time, as shown in formula (1). There are several factors that can influence the response time, such as mobile services, cloud resources and network connections, as shown in Table 1. In order to predict the response time, we build three information models to predict the server time and the network time, including *Network Connection* model, *Network Time* model and *Server Time* model. *Network Connection* model is used to predict the quality of network connection. *Network Time* model is used to calculate the network time for the service with a network connection. *Server Time* model is used to predict the service execution time on a cloud resource. We detail these three information models in the following paragraphs.

$$T_{response} = T_{server} + T_{network} \quad (1)$$

1) *Network Connection* model: The quality of network connection changes continuously, as the device moves throughout the day. The quality of network connection includes the data transmission rate and the round trip time from the device to the cloud resource. *Network*

Table 1 List of the factors that can influence the response time

Symbol	Description
S	the set of movable services
R	the set of cloud resources
N	the set of networks
V	the expectations of the data transmission rate
v_{ij}	the expectation of the data transmission rate between the device in network i and cloud resource j
RTT	the expectations of the round trip time
rtt_{ij}	the expectation of the round trip time between the device in network i and cloud resource j
C	the changing trends of networks that the device uses
c_{ij}	the number of device movements from network i to network j
E	the expectations of the server time for service execution on cloud resources
e_{kj}	the expectation of the server time for service k execution on cloud resource j
$D(s_k)$	the average data traffic for a single invocation of service k
W^T	the weight factors

Connection model is used to predict the quality of network connection based on history data gathered by the client. We let $N = \{n_1, n_2, ..., n_h\}$ denote the set of networks and $R = \{r_1, r_2, ..., r_m\}$ demote the set of cloud resources. And there are two matrices V and RTT that record the expectations of the data transmission rate and the round trip time respectively, as shown in formula (2) and (3).

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1m} \\ v_{21} & v_{22} & \cdots & v_{2m} \\ \cdots & \cdots & \ddots & \cdots \\ v_{h1} & v_{h2} & \cdots & v_{hm} \end{bmatrix} \tag{2}$$

$$RTT = \begin{bmatrix} rtt_{11} & rtt_{12} & \cdots & rtt_{1m} \\ rtt_{21} & rtt_{22} & \cdots & rtt_{2m} \\ \cdots & \cdots & \ddots & \cdots \\ rtt_{h1} & rtt_{h2} & \cdots & rtt_{hm} \end{bmatrix} \tag{3}$$

Each v_{ij} in the matrix V, represents the expectations of the data transmission rate between the device in network i and cloud resource j, and it takes a weighted average of the history data V^{ij}, as shown in formula (4). Each rtt_{ij} in the matrix RTT, represents the expectations of the round trip time between the device in network i and cloud resource j, and it takes a weighted average of the history data RTT^{ij}, as shown in formula (5).

$$v_{ij} = W^T \bullet V^{ij} = (w_1 \quad w_2 \quad \cdots \quad w_p) \begin{pmatrix} {v_1}^{ij} \\ {v_2}^{ij} \\ \cdots \\ {v_p}^{ij} \end{pmatrix} \tag{4}$$

$$rtt_{ij} = W^T \bullet RTT^{ij} = (w_1 \quad w_2 \quad \cdots \quad w_p) \begin{pmatrix} {rtt_1}^{ij} \\ {rtt_2}^{ij} \\ \cdots \\ {rtt_p}^{ij} \end{pmatrix} \tag{5}$$

$$\text{s.t.} \quad w_1 + w_2 + \cdots + w_p = 1$$

In addition, there is a matrix C that records the changing trends of networks that the device uses, as shown in formula (6). Each c_{ij} in the matrix C, represents the number of device movements from network i to network j.

$$C = \begin{bmatrix} 0 & c_{12} & \cdots & c_{1h} \\ c_{21} & 0 & \cdots & c_{2h} \\ \cdots & \cdots & \ddots & \cdots \\ c_{h1} & c_{h2} & \cdots & 0 \end{bmatrix} \tag{6}$$

2) *Network Time* model: There are three factors that can influence the network time, including the service, the data transmission rate and the round trip time. We let $S = \{s_1, s_2, ..., s_g\}$ denote the set of movable services. The network time can be calculated by formula (7). $D(s_k)$ represents the average data traffic for a single invocation of service k. This average data traffic can be obtained from training. When $D(s_k)$ is large, the network time mainly depends on the data transmission rate. Instead, when $D(s_k)$ is small, the network time mainly depends on the round trip time. Thus, given the service and the quality of network connection, we can calculate the network time based on this model.

$$T_{network}(s_k, v, rtt) = \max\{\frac{D(s_k)}{v}, rtt\} \tag{7}$$

3) *Server Time* model: Mobile services have differences in computation complexity and cloud resources are different in processing power, so the server time for service execution on server is not the same. The server time cannot be directly monitored by the client, but it can be calculate by formula (8). The response time for service invocation can be monitored, and the network time can be calculated based on *Network Time* model.

$$T_{server} = T_{response} - T_{network} \tag{8}$$

There is the matrix E that records the expectations of server time, as shown in formula (9). Each e_{kj} in the matrix E, represents the expectations of server time for service k execution on cloud resource j, and it takes a weighted average of the history data E^{kj}, as shown in formula (10).

$$E = \begin{bmatrix} e_{11} & e_{12} & \cdots & e_{1m} \\ e_{21} & e_{22} & \cdots & e_{2m} \\ \cdots & \cdots & \ddots & \cdots \\ e_{g1} & e_{g2} & \cdots & e_{gm} \end{bmatrix} \tag{9}$$

$$e_{kj} = W^T \bullet E^{kj} = (w_1 \quad w_2 \quad \cdots \quad w_p) \begin{pmatrix} {e_1}^{kj} \\ {e_2}^{kj} \\ \cdots \\ {e_p}^{kj} \end{pmatrix} \tag{10}$$

$$\text{s.t.} \quad w_1 + w_2 + \cdots + w_p = 1$$

Selection algorithm

When the device context, such as the network connection, changes, it needs to reselect the resource for offloading, in order to improve application performance. It costs a lot of extra time and energy to deploy mobile codes onto all available cloud resources, invoke these services and compare their response time. So we need to predict the response time for remote service invocation and select the appropriate services. On one hand, the

optimal service needs to be selected, in order to improve application performance in the current network. On the other hand, the standby service also needs to be selected, in order to assure the quality of offloading service when the device context changes.

1) *Selection for the Optimal Service*: The optimal service is the one with the shortest response time in the current network. We propose the decision algorithm to select the optimal service based on information models, as shown in Algorithm 1. There are three steps in the decision procedure.

Algorithm 1: Algorithm to select the optimal service

1: procedure selectOptimalService (models, context , s_k)
2: Matrices V , E , RTT , R obtained from models (described in Section 4.1)
3: $T_{local} \leftarrow$estimate execution time of s_k on the r_{local}
4: $r_{optimal} \leftarrow T_{offload} = \min_{\forall r_j \in R}\left(e_{kj} + T_{transfer}(s_k, v_{ij}, rtt_{ij})\right)$
5: While (the context of $r_{optimal}$ is abnormal) do
6: {
7: $R = R - \{r_{optimal}\}$
8: $r_{optimal} \leftarrow T_{offload} = \min_{\forall r_j \in R}\left(e_{kj} + T_{transfer}(s_k, v_{ij}, rtt_{ij})\right)$
9: }
10: if ($T_{offload} < T_{local}$) then return $r_{optimal}$
11: else return r_{local}

Step1: Obtain the context of the mobile device, such as the service and the network connection.
Step2: Predict the response time for service execution on different cloud resources, based on information models.
Step3: Select the optimal service and check it.

2) *Selection for the Standby Service*: When the device context changes, it takes a long time to reselect the cloud resource and deploy the service. In order to ensure application performance during this period of time, there needs to be a standby service that is not only available in the current network, but also in next possible networks. There are two considerations for selecting the standby service. On one hand, the service should be available in as many networks as possible. On the other hand, the total time saved by offloading should be as much as possible. The two aspects need to be traded off, as shown in formula (11), and the weights can be tuned according to different requirements.

$$\mathrm{Value}(r_j) = w_1 * \mathrm{Value}_a(r_j) + w_2 * \mathrm{Value}_e(r_j) \quad (11)$$

We propose the decision algorithm to select the standby service based on information models, as shown in Algorithm 2. There are three steps in the decision procedure.

Algorithm 2: Algorithm to select the standby service

1: procedure selectStandbyService (models , context, s_k)
2: Matrices V , E , RTT , R , N , C obtained from models (described in Section 4.1)
3: $T_{local} \leftarrow$estimate execution time of s_k on the r_{local}
4: $f(x) = \begin{cases} 0, & x \le 0 \\ 1, & x > 0 \end{cases}$ where 1 denotes that it is worth offloading ; 0 denotes that it is not suitable to offload
5: $\triangle\phi(s_k, n_f, r_j) = T_{local} - \left(e_{kj} + T_{transfer}(s_k, v_{ij}, rtt_{ij})\right)$ It denotes the reduced time when offloaded to r_j in n_f
6: $V_a(r_j) \leftarrow \sum_{f=1}^{n}\left(\frac{c_{if}}{\sum_{k=1}^{n} c_{ik}} \times f(\triangle\phi(s_k, n_f, r_j))\right)$ It denotes the availability when offloaded to r_j
7: $V_e(r_j) \leftarrow \sum_{f=1}^{n}\left(\frac{c_{if}}{\sum_{k=1}^{n} c_{ik}} \times f(\triangle\phi(s_k, n_f, r_j)) \times \triangle\phi(s_k, n_f, r_j)\right)$ It denotes the effectiveness when offloaded to r_j
8: $r_{standby} \leftarrow \max_{\forall r_j \in R}\left(w_1 \times V_a(r_j) + w_2 \times V_e(r_j)\right)$
9: While(the context of $r_{standby}$ is abnormal) do
10: {
11: $R = R - \{r_{standby}\}$
12: $r_{standby} \leftarrow \max_{\forall r_j \in R}\left(w_1 \times V_a(r_j) + w_2 \times V_e(r_j)\right)$
13: }
14: return $r_{standby}$

Step1: Obtain the context of the mobile device, such as the service and the network connection.
Step2: Predict availability and effectiveness of services on different cloud resources, based on information models.
Step3: Select the standby service and check it.

Implementation

The proposed framework is divided into two parts, including the client framework and the server framework, as shown in Fig. 5. The client framework is deployed on mobile devices, and the server framework is deployed on cloud resources.

The client framework is used to dynamically select the appropriate cloud resources and offload mobile codes to them on demand. It consists of three modules, including the service selection module, the computation offloading module and the runtime management module. The service selection module collects runtime data, builds information models and selects appropriate cloud resources, as mentioned in Section 4. The data collection manager records history data about runtime status, such as the response time, the data transmission rate and the round trip time. The information modeling manager builds the three information models based on the history data, including the *Network Connection* model, the *Network Time* model and the *Server Time* model. The service selection manager predicts the response time for remote

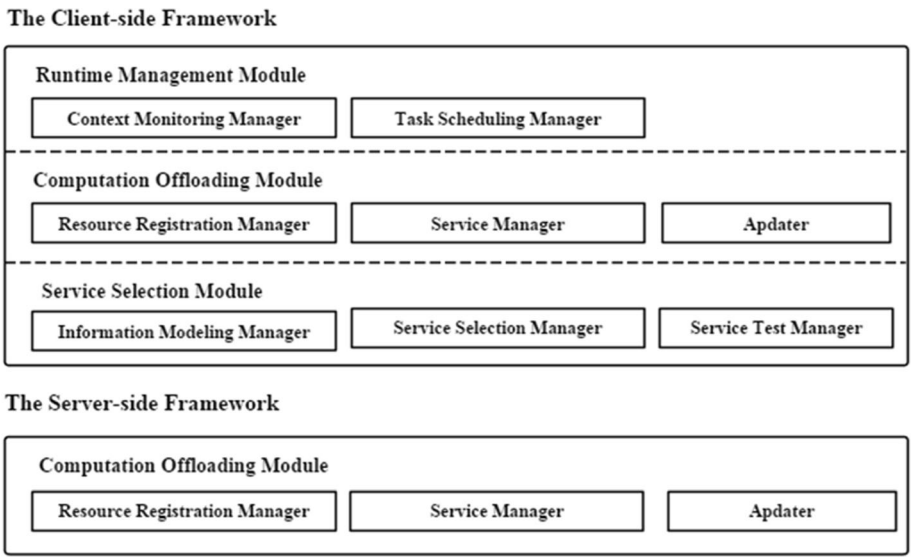

Fig. 5 The framework overview

service invocation and selects the appropriate cloud resources based on the information models. The computation offloading module establishes connections to cloud resources, manages available mobile services and helps invoke remote services, as mentioned in Section 3. The resource registration manager establishes connections to servers and deploys services onto them. The service manager consists of the service registry and the service pools, which manages available mobile services. The adaptor is the dynamic proxy and it supports the adaptation on offloading. The runtime management module monitors the device context and schedules different tasks of framework.

The server framework is used to support offloading, and it mainly works with the computation offloading module on the client side, as mentioned in Section 3.

Evaluation

The goals of the evaluation are to (1) validate whether our framework is feasible to offload real-world applications; (2) compare the performance of offloaded applications with original ones; (3) compare the battery power consumption of offloaded applications with original ones.

Case study

In this case study, there are two tested devices, five locations with different wireless networks, and five cloud resources, as following.

First, the two tested devices are an HTC M8St [21] with 2.5GHz quad-core CPU, 2GB RAM, and a SAMSUNG GT-N7000 [22] with 1.4GHz dual-core CPU, 1GB RAM.

Second, we simulate five locations, with their names from *location1* to *location5*, as shown in Fig. 6. There are free WIFI connection services in all of these locations except *location2*. And each connection performs differently in speed.

Third, there are multiple options of cloud resources, including four nearby cloudlets and one public Cloud service, which can be used for offloading in different networks. The details of these cloud resources are listed as follows. The *location1* Cloudlet is a device with 3.6GHz octa-core CPU and 32GB RAM, which can be publicly accessed using 3G and WIFI; the *location3* Cloudlet is also a device with 3.6GHz octa-core CPU and 32GB RAM; the *location4* Cloudlet is a device with 3.2GHz dual-core CPU and 4GB RAM; the *location5* Cloudlet is a device with 2.2GHz dual-core CPU and 2GB RAM; in addition, the device running in Ali Cloud, with 2.6GHz dual-core CPU and 4GB RAM, serves as a public Cloud Service.

We apply our framework to support context-aware computation offloading on two real-world mobile applications. The first application is the Gobang game with three difficulty levels. It is an interactive app that human and AI players place chess pieces in turns. When it is the turn of AI player, all the chess piece positions on the chess board will be transferred to the AI module that plans the move. The second application is the Face Finder that is to find out how many faces are in the picture. Each picture is of a fixed size of 1288 KB and is transferred to the Finder module that counts faces. The two applications are developed following the design pattern described in Section 3. The Gobang game is of three movable services that are used to plan moves for different difficulty levels, including

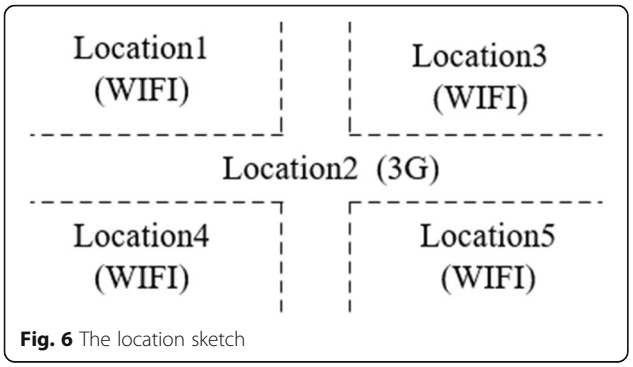

Fig. 6 The location sketch

the Low-level AI service, the Mid-level AI service and the High-level AI service. These three AI services are implemented by three algorithms with different computation complexities. The Face Finder is of just one movable service, named the Finder service. The Finder service is used to count faces in the picture, and it is a computation intensive service.

In our framework, the runtime data is collected from the monitoring activities, in order to build the information models that are used to calculate the server time and the network time. In this case study, the three information models are as follows.

1) *Network Connection* model: The information about network connections is collected when the mobile device moves during the day. The *Network Connection* models of the two tested devices are shown in Tables 2 and 3. The two models are almost the same. The reason is that the *Network Connection* model mainly reflects the quality of network connections between the device in different networks and cloud resources, including the data transmission rate and the round trip time. Thus, there is no great difference between different devices. As shown in Tables 2 and 3, there are five networks and five cloud resources. Both of the Public Cloud Service and *location1* Cloudlet can be used for offloading in any network. But other cloud resources can just be available in the WIFI networks that are limited to their own locations. For instance, the *location3* Cloudlet can only be used when the device is in the *location3* WIFI network. In addition, each connection performs differently in terms of the data transmission rate and the round trip time, the details of which are shown in Tables 2 and 3.

2) *Network Time* model: The average data traffic for service invocation can be obtained from training. The *Network Time* models of mobile services are shown in formula (12). The data traffic of the Finder service is much more than AI services for a single invocation.

$$\begin{aligned} T_{network}(\text{Finder}, v, rtt) &= \max\{\frac{1288KB}{v}, rtt\} \\ T_{network}(\text{AI}, v, rtt) &= \max\{\frac{3KB}{v}, rtt\} \end{aligned} \tag{12}$$

3) *Server Time* model: The server time of services cannot be directly monitored by the client, but it can be calculated as described in Section 4.1. The *Server Time* models of the two tested devices are shown in Tables 4 and 5. The two models are almost the same. The reason is that the *Server Time* model reflects the remote execution time that mainly depends on computation complexity of mobile services and processing power of cloud resources, thus, there is no great difference in the server time between different devices. Compared with local execution, it takes much less time to execute the same service remotely. The more powerful the hardware of cloud resource is, the less the server time should be.

Based on the above information models, our framework can automatically compare the response time of services on different cloud resources and use appropriate cloud resources for offloading. Tables 6 and 7 show the cloud resource selection of two tested devices for each service in different networks. The trade-off is between the reduced execution time and the network delay. For instance, when the mobile device runs applications in the *location5* WIFI network, the framework can use the *location5* cloudlet or the public cloud for offloading, as shown in Tables 2 and 3. The network connection to the

Table 2 The *network connection* model of the htc device

Resource Network	*location1* Cloudlet	Public Cloud Service	*location3* Cloudlet	*location4* Cloudlet	*location5* Cloudlet
$WIFI_{location1}$	RTT = 40 ms V = 1200 KB/s	RTT = 150 ms V = 500 KB/s	×	×	×
3G	RTT = 480 ms V = 12 KB/s	RTT = 340 ms V = 20 KB/s	×	×	×
$WIFI_{location3}$	RTT = 400 ms V = 140 KB/s	RTT = 150 ms V = 500 KB/s	RTT = 40 ms V = 1200 KB/s	×	×
$WIFI_{location4}$	RTT = 600 ms V = 25 KB/s	RTT = 230 ms V = 300 KB/s	×	RTT = 60 ms V = 1024 KB/s	×
$WIFI_{location5}$	RTT = 650 ms V = 15 KB/s	RTT = 400 ms V = 240 KB/s	×	×	RTT = 75 ms V = 800 KB/s

Table 3 The *network connection* model of the samsung device

Resource Network	*location1* Cloudlet	Public Cloud Service	*location3* Cloudlet	*location4* Cloudlet	*location5* Cloudlet
$WIFI_{location1}$	RTT = 45 ms V = 1190 KB/s	RTT = 150 ms V = 500 KB/s	×	×	×
3G	RTT = 480 ms V = 11 KB/s	RTT = 350 ms V = 20 KB/s	×	×	×
$WIFI_{location3}$	RTT = 410 ms V = 142 KB/s	RTT = 155 ms V = 498 KB/s	RTT = 43 ms V = 1195 KB/s	×	×
$WIFI_{location4}$	RTT = 600 ms V = 25 KB/s	RTT = 230 ms V = 300 KB/s	×	RTT = 60 ms V = 1020 KB/s	×
$WIFI_{location5}$	RTT = 650 ms V = 15 KB/s	RTT = 405 ms V = 240 KB/s	×	×	RTT = 75 ms V = 796 KB/s

location5 cloudlet is better than the public cloud, but the hardware of the public cloud is more powerful than the *location5* cloudlet. For the Finder service and the High-level AI service, the framework uses the public cloud for offloading, because the two services are computation intensive ones and the powerful hardware can reduce a lot of execution time. For the Mid-level AI service, the framework uses the *location5* cloudlet for offloading, because the reduced execution time of two cloud resources is almost the same, but the network delay of the public could is higher than the *location5* cloudlet. For the Low-level AI service, the framework invokes the local service, because the reduced execution time cannot be greater than the network delay caused by offloading. In addition, due to the difference in processing power between two devices, the frameworks on them are possible to make different decisions whether to offload, for the same service and in the same network. For instance, when the devices run the Gobang game with Mid-level AI in the 3G network, the framework on the HTC device invokes the local service, while the framework on the SAMSUNG device uses the public cloud for offloading, as shown in Tables 6 and 7. The reason is that the reduced execution time for the SAMSUNG device is greater than the network delay caused by offloading, while the reduced execution time for the HTC device is less than the network delay, as shown in Tables 4 and 5.

Overheads of the estimation model can be broken into two parts. On one hand, some extra service invocations are needed to gather sufficient runtime data. For instance, when running the High-level AI service in the *location5* WIFI network, the framework not only uses the public cloud service for offloading, but also occasionally invokes the services provided by the location 1 and 5 cloudlets in order to collect information such as server time. However, the execution frequency can be controlled and the framework can invoke these services during idle time, thus, the overhead of gathering runtime data is acceptable. On the other hand, the framework needs to calculate and compare the response time of services on different cloud resources, in order to select the appropriate cloud resources for offloading. Figure 7 shows the execution time of selection algorithm in different locations. Compared with service invocation, the overhead of service selection is acceptable.

Comparison of App performance

We evaluate our approach in performance from two aspects. On one hand, we compare the response time of context-aware offloaded applications with original and traditional offloaded ones for the same services, when the devices stay in different locations. On the other hand, we compare the response time of our offloaded applications with ones without the standby service for the same services, when the devices move between different locations.

Staying in different locations

There are three types of applications in this experiment, including the original one, the traditional offloaded one

Table 4 The *server time* model (solid line) and processing power (dotted line) of the samsung device

Resource Service	*location1* Cloudlet	Public Cloud Service	*location3* Cloudlet	*location4* Cloudlet	*location5* Cloudlet	Local
Finder	1953 ms	6987 ms	1952 ms	5455 ms	11269 ms	34801 ms
Low-level AI	0 ms	4 ms	0 ms	4 ms	11 ms	15 ms
Mid-level AI	17 ms	43 ms	18 ms	58 ms	110 ms	835 ms
High-level AI	175 ms	297 ms	176 ms	246 ms	840 ms	5818 ms

Table 5 The *server time* model (solid line) and processing power (dotted line) of the htc device

Resource / Service	*location1* Cloudlet	Public Cloud Service	*location3* Cloudlet	*location4* Cloudlet	*location5* Cloudlet	Local
Finder	1946 ms	6981 ms	1946 ms	5457 ms	11272 ms	13168 ms
Low-level AI	0 ms	4 ms	0 ms	4 ms	10 ms	2 ms
Mid-level AI	16 ms	39 ms	16 ms	53 ms	112 ms	272 ms
High-level AI	175 ms	297 ms	175 ms	246 ms	840 ms	2396 ms

and the context-aware offloaded one. The original application runs entirely on the phone; the traditional offloaded application just uses the public cloud service for offloading; the context-aware offloaded application can use any of the above cloud resources for offloading. We compare their performance for each service in five different networks, using SAMSUNG and HTC devices respectively, as shown in Figs. 8 and 9.

The computation complexity of service impacts the offloading effect significantly. If the service has a high computation complexity, it is worth offloading. Instead, if the service has a low computation complexity, it is better to be executed locally. For instance, when using the SAMSUNG device to invoke the High-level AI service in the *location3* WIFI network, the response time for the original application is 5818 ms on average, while the time for the context-aware offloaded one is 245 ms or reduced by 96%, and the traditional offloaded one is 447 ms or reduced by 92%. The reason for the great performance improvement is that the computation intensive code is executed on a more powerful processor of the server other than the phone's own processor. However, when using the SAMSUNG device to invoke the Low-level AI service in the *location3* WIFI network, the response time for the original application is 15 ms on average, while the time for the context-aware offloaded one and the traditional offloaded one is separately 45 ms and 154 ms. The reason for the negative impact of offloading is that the network delay is greater than the reduced execution time caused by offloading. In addition, the context-aware offloaded application executes the Low-level AI service locally as same as the original one, but with an overhead of 30 ms. The slight increase of execution time is due to that service invocation will be forwarded by the adapter.

The quality of network connection impacts the offloading effect a lot. If the RTT value becomes larger or the data transmission rate becomes lower, the performance of the traditional offloaded application will get decreased due to the fact that more time will be spent on network communication. For instance, when using the SAMSUNG device to invoke the Finder service in the *location1* WIFI network (RTT = 150 ms, V = 500 KB/s), the response time for the original application is 34801 ms, while the time for the traditional offloaded one is 9589 ms or reduced by 72%. However, when using the SAMSUNG device to invoke the Finder service in the 3G network (RTT = 340 ms, V = 20 KB/s), the response time for the traditional offloaded application is 72181 ms or increased by about one time. The reason for the performance degradation of offloading is that the average data traffic for a single invocation of the Finder service is great, so a lot of time will be spent on network communication if the speed of network connection is slow.

The processing power of device also impacts the offloading effect. If the device has a lower computing capability, the mobile service is more worth offloading. For instance, when using the SAMSUNG device to invoke the Mid-level AI service in the 3G network, the response time for the original application is 835 ms, while the time for the traditional offloaded one is 379 ms or reduced by 55%. However, when using the HTC device to invoke the Mid-level AI service in the 3G network, the response time for the original application is 272 ms,

Table 6 The cloud resource selection of the samsung device for each service and in different networks

Network / Service	$WIFI_{location1}$	3G	$WIFI_{location3}$	$WIFI_{location4}$	$WIFI_{location5}$
Finder	*location1* Cloudlet	Local	*location3* Cloudlet	*location4* Cloudlet	Public Cloud Service
Low-level AI	Local	Local	Local	Local	Local
Mid-level AI	*location1* Cloudlet	Public Cloud Service	*location3* Cloudlet	*location4* Cloudlet	*location5* Cloudlet
High-level AI	*location1* Cloudlet	Public Cloud Service	*location3* Cloudlet	*location4* Cloudlet	Public Cloud Service

Table 7 The cloud resource selection of the htc device for each service and in different networks

Network Service	$WIFI_{location1}$	3G	$WIFI_{location3}$	$WIFI_{location4}$	$WIFI_{location5}$
Finder	*location1* Cloudlet	Local	*location3* Cloudlet	*location4* Cloudlet	Public Cloud Service
Low-level AI	Local	Local	Local	Local	Local
Mid-level AI	*location1* Cloudlet	Local	*location3* Cloudlet	*location4* Cloudlet	*location5* Cloudlet
High-level AI	*location1* Cloudlet	Public Cloud Service	*location3* Cloudlet	*location4* Cloudlet	Public Cloud Service

while the time for the traditional offloaded one is still 379 ms or increased by 39%. The reason for the difference in the offloading effect is that the HTC M8St is much better than the SAMSUNG GT-N7000 in processing power, so the local execution time of the HTC device is much less than the SAMSUNG device for the same service.

We can see that, the computation complexity of the service, the quality of the network and the performance of the device can all impact the offloading effect. It needs to dynamically decide whether the service is executed locally or remotely and which cloud resource is used for offloading according to the device context. As a whole, our approach can help reduce execution time by 6–96% for computation-intensive applications.

Moving between different locations

There are two types of applications in this experiment, including our context-aware offloaded ones and modified ones without the standby service. We constantly invoke the High-level AI service while we move between different locations in sequence, remaining in each location for three minutes. Figure 10 shows each invocation time of the two applications for the High-level AI service on the SAMSUNG device during this process.

For the modified applications, it takes a much longer time to invoke the service when the device just enters a new location. For instance, it takes 667 ms for our offloaded application to invoke the High-level AI service while 5818 ms for the modified one, when the device just enters *location2* from *location1*. The reason for the

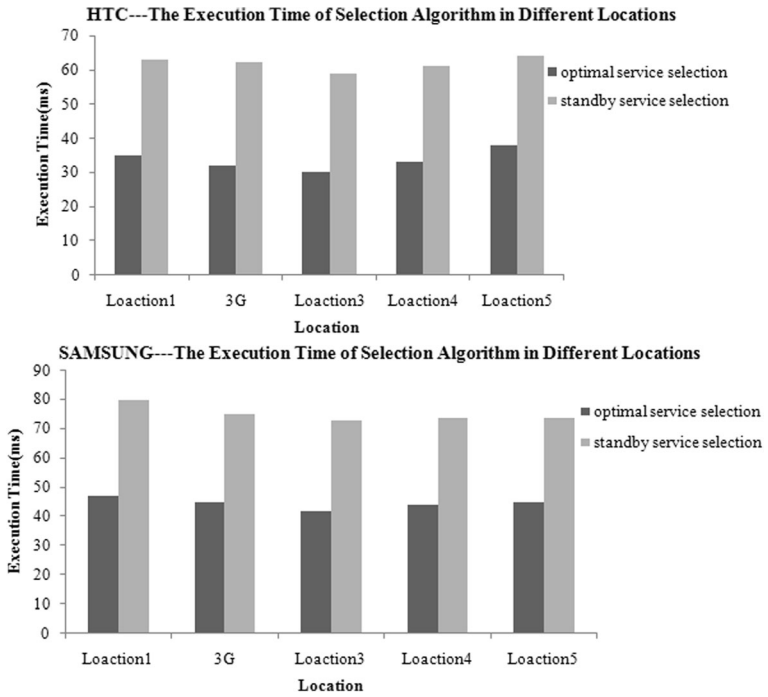

Fig. 7 The execution time of selection algorithm in different locations

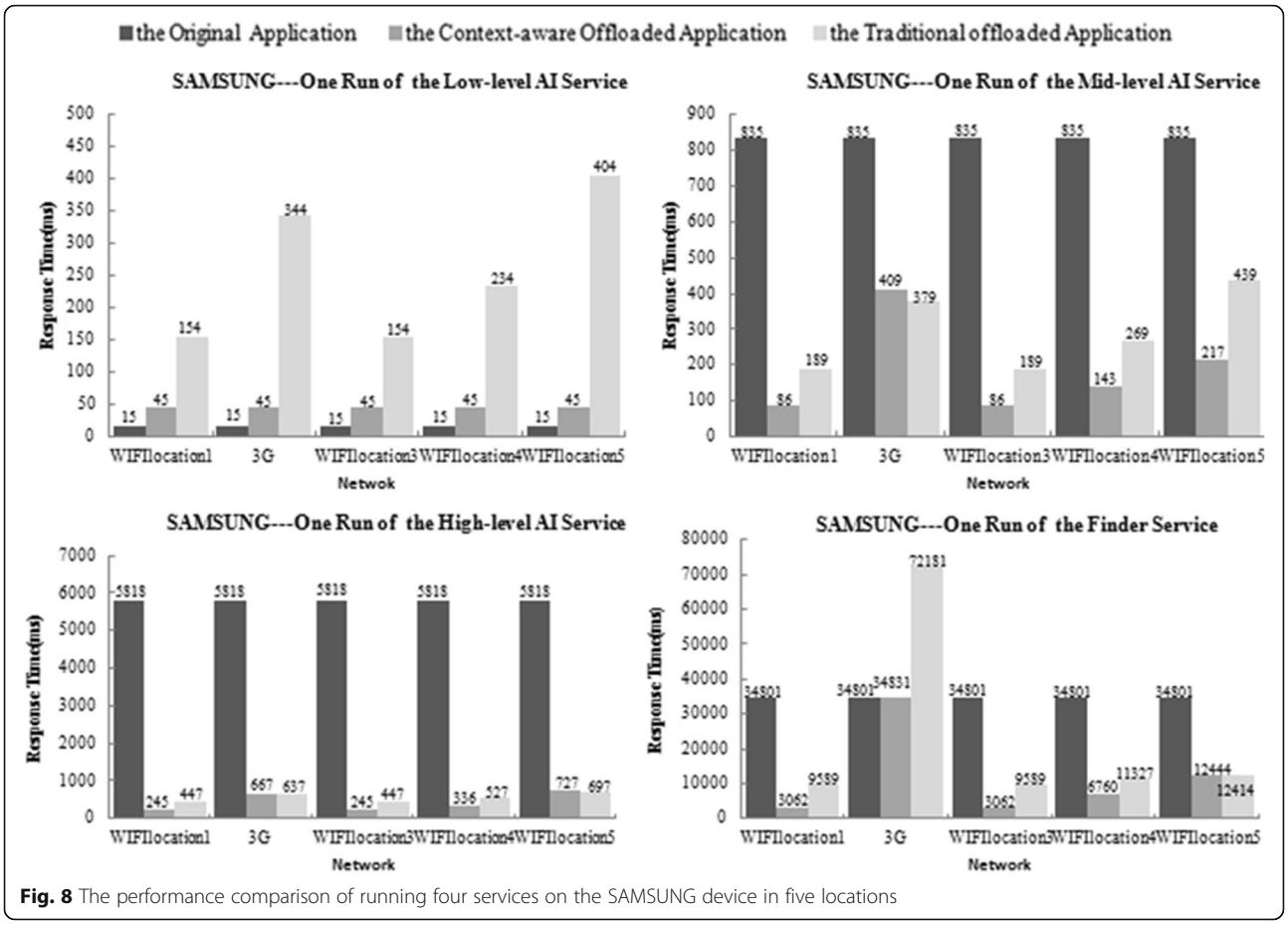

Fig. 8 The performance comparison of running four services on the SAMSUNG device in five locations

performance difference is that when the device context changes, it takes a long time to reselect the cloud resource and deploy the service. During this period of time, our offloaded application can use the standby service that ensures application performance, while the modified one can just use the original service or the local service. So our offloaded application outperforms the one without the standby service.

Comparison of App power consumption

We evaluate our approach in power consumption, using the three versions of the same application that are mentioned in Section 6.2.1. As shown in Figs. 11 and 12, for each service and under different networks, we measure their power consumption on SAMSUNG and HTC devices by the PowerTutor Android application [23] that gives the details of the power consumption for each targeted service.

When running the service locally, the power consumption of our context-aware offloaded applications will increase slightly compared with the original ones, because the adapters in our offloaded ones cost energy to run. For instance, when invoking the Low-level AI service on the SAMSUNG device, the power consumption of the original application and our offloaded one is separately 14.57 Joules and 14.87 Joules.

When running the computation-intensive service, the energy consumption of two offloaded applications is often reduced, as offloading makes some computation intensive code be executed on the server. For instance, when invoking the High-level AI service and the Finder service, the power consumption of our offloaded applications can be reduced by 60–96% compared with the original one.

What should be noted is that, the power used on network communication can makes the total power consumption of offloaded applications be greater than that of the original ones. For instance, when invoking the Finder service on the SAMSUNG device under the 3G network, the power consumption for offloading is increased by 25% compared with running locally.

We can see that, our offloaded applications outperform the other two applications in power consumption in most of the time.

Related work

The idea of using a strong server to enhance the processing capabilities of a weak device is not new [24].

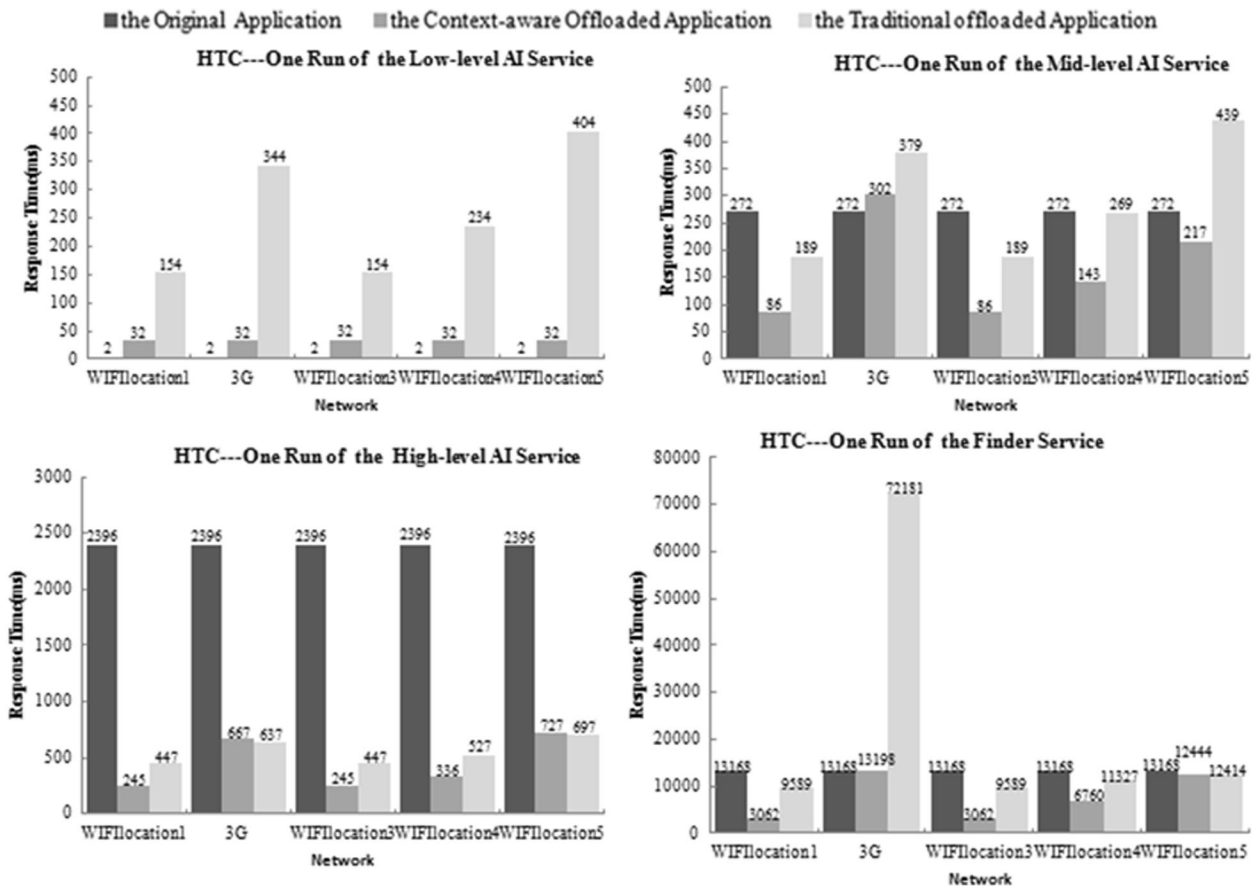

Fig. 9 The performance comparison of running four services on the HTC device in five locations

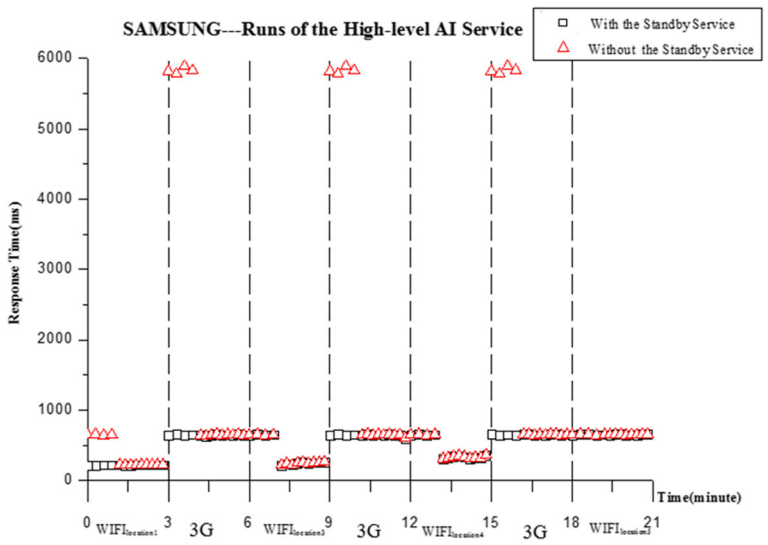

Fig. 10 The effect of standby service

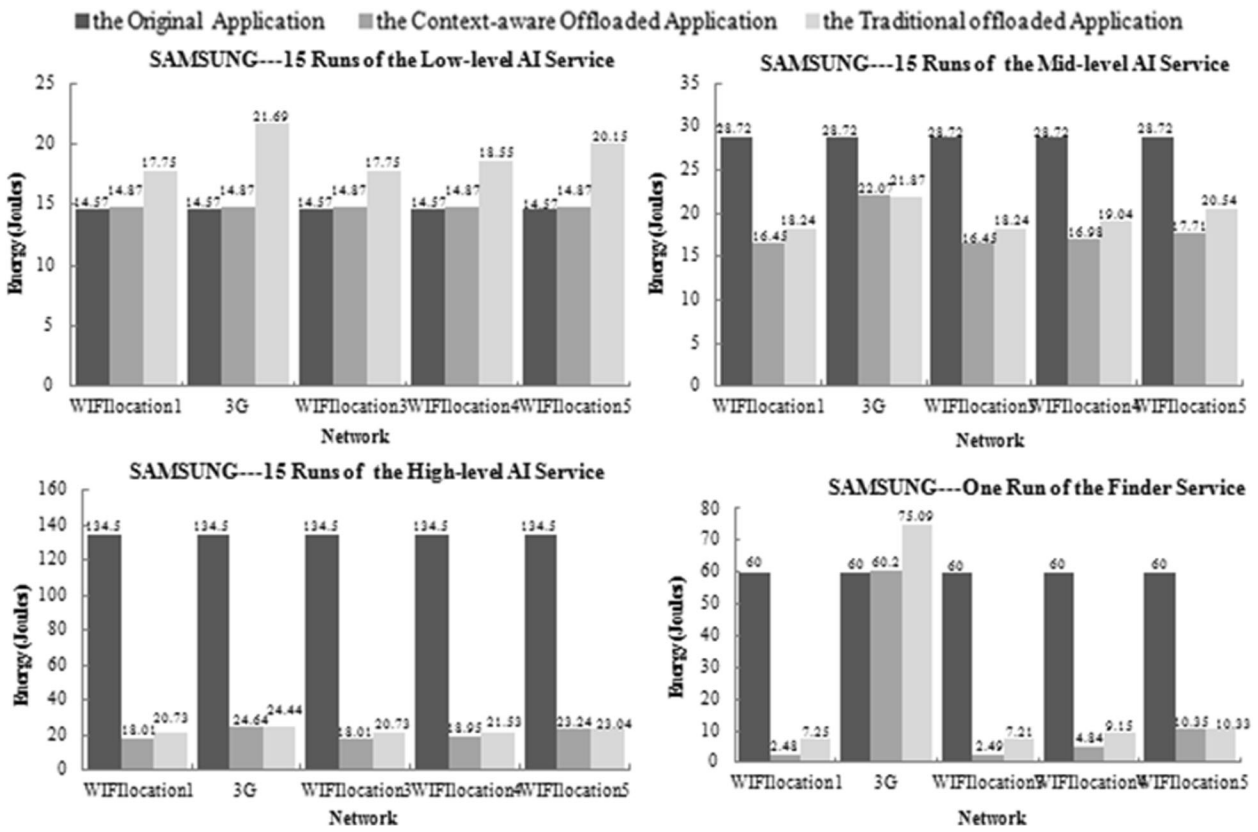

Fig. 11 The power consumption comparison of running four services on the SAMSUNG device in five locations

Many early researches [25–27] have tried to automatically partition a standalone desktop app to have some parts of it executed remotely on the server. J-Orchestra [27] and JavaParty [25] are the early work on offloading standalone desktop Java applications. J-Orchestra works on the Java byte code level, while JavaParty works on the source code level. They both require developers to manually tell the offloading tool about which class can be offloaded. Then the compiler provided by these tools will compile these selected classes to generate the RMI stubs/skeletons for them, so that the application is turned to be a client/server one by using RMI as the communication channel. The work of J-Orchestra and JavaParty cannot be directly used for offloading Android applications.

The researches on mobile cloud computing then leverage such an idea to realize computation offloading on mobile devices [28–32]. Cuckoo [28] and MAUI [29] provide method-level computation offloading. Cuckoo requires developers to follow a specific programming model to make some parts of the application be offloaded. MAUI requires developers to annotate the can-be-offloaded methods of a .Net mobile application by using the "Remoteable" annotation. Then their analyzers will decide which method should really be offloaded through runtime profiling. ThinkAir [30] also provides method-level computation offloading, but it focuses on the elasticity and scalability of the cloud and enhances the power of mobile cloud computing by parallelizing method execution using multiple virtual machine images. CloneCloud [31] provides thread-level computation offloading. It modifies the Android Dalvik VM to provide an application partitioner and an execution runtime to help applications running on the VM offload tasks to execute on a cloned VM hosted by a Cloud server. DPartner [32] provides class-level computation offloading. It can automatically refactor Android applications to be the ones with computation offloading capability and generates two artifacts to be deployed onto an Android phone and the server, respectively. The above works mainly focus on the code partitioning and offloading techniques, assuming that mobile codes are offloaded to a prepared server or a predefined Cloud.

Some recent works also consider the mobility issue [33–35], which provide context-aware offloading schemes for mobile cloud computing. The work proposed by Lin et al. [33], provides a context-aware decision engine that takes into consideration signal strength, transmission time, geographical location and the time slots of the offloading opportunities to decide whether to offload a

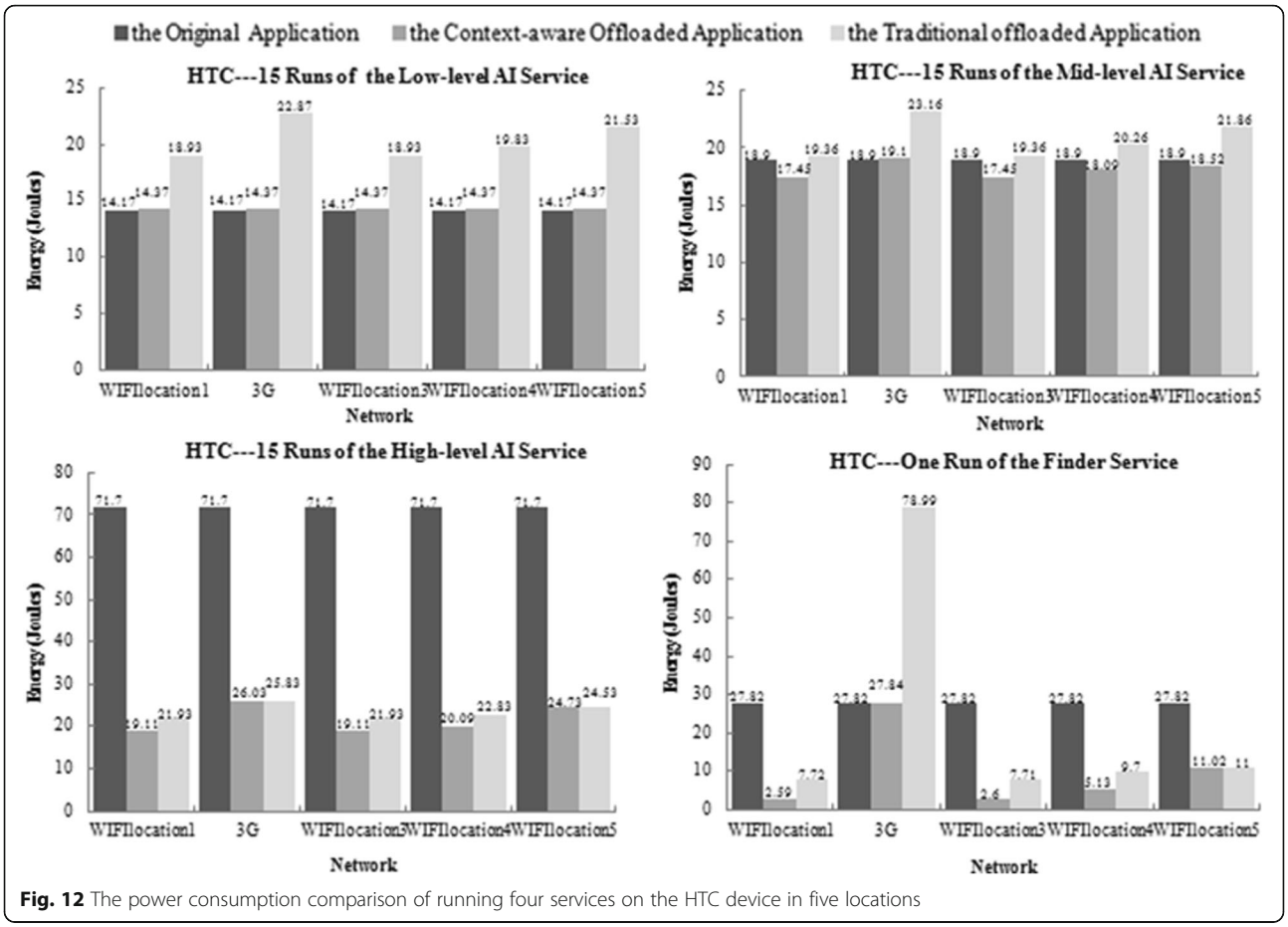

Fig. 12 The power consumption comparison of running four services on the HTC device in five locations

given method to the cloud server. The work proposed by Ravi and Peddoju [34], provides a multi criteria decision making (MCDM) algorithm for choosing the best possible resource to offload the computation tasks, and a handoff strategy to offload the tasks between different resources. Similarly, the work proposed by B Zhou et.al. [35], provides a context-aware offloading decision algorithm that takes into consideration context changes such as network condition, device information and the availability of multiple types of cloud resources to provide decisions on the selection of the wireless medium to utilize and the cloud resource to offload at runtime. The above works mainly focus on context-aware offloading decision algorithms, assuming that these algorithms can work with the mobile cloud offloading system.

Our framework can dynamically select the appropriate cloud resources and offload mobile codes to them on demand, according to the device context. It is different from the above works mainly in two aspects. First, our framework introduces the service pool that enables the device to use several remote servers for offloading at the same time and improves the availability of offloading service, while few of the existing works can support such a feature. Second, our framework uses the history data as well as the context to decide whether to offload and which cloud resource to offload, while most of the existing works need to monitor lots of runtime status in order to calculate the reduced execution time and the network delay. In addition, the client side of our framework is independent from the server side, that is, builds information models and selects appropriate services independently. Therefore, our framework is capable to interwork with the game-theoretic model [36, 37] and be extended to the multi-user case.

Conclusion and future work

In this paper, we present a framework for context-aware computation offloading to select the appropriate Cloud resource and then offload mobile codes to it in a dynamic manner, which makes use of both remote cloud computing services and nearby cloudlets. First, a design pattern is proposed to enable an application to be computation offloaded on-demand. Second, an estimation model is presented to automatically select the cloud resource for offloading. Third, a framework is implemented to support the design pattern and the estimation model. Then we apply our framework to support context-aware computation offloading on two real-world applications and the evaluation results show that our

approach can help significantly reduce the app execution time and battery power consumption.

Our framework for context-aware computation offloading will be further validated and improved as follows. On one hand, more experiments with real-world mobile applications will be done. On the other hand, more rules and algorithms will be explored and applied in the estimation model in order to achieve better results.

Acknowledgements
This work was supported by the National Natural Science Foundation of China under Grant No. 61402111, 61300002, the National High-Tech R&D Program of China (863) under Grant No. 2015AA01A202, the Major Science and Technology Project of Fujian Province under Grant No. 2015H6013, the Science and Technology Platform Development Program of Fujian Province under Grant No. 2014H2005.

Authors' contributions
XC, SC and XZ carried out the computation offloading studies, participated in the study and drafted the manuscript. YZ participated in the design of the study and performed the statistical analysis. XZ and CR conceived of the study, and participated in its design and coordination and helped to draft the manuscript. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China. [2]Fujian Provincial Key Laboratory of Networking Computing and Intelligent Information Processing, Fuzhou University, Fuzhou 350108, China. [3]School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China. [4]Department of Computer Science and Electronic Engineering, University of Stavanger, Stavanger 4036, Norway.

Received: 2 September 2016 Accepted: 13 December 2016
Published online: 05 January 2017

References

1. Bonomi F, Milito R, Natarajan P, Zhu J (2014) Fog computing: a platform for internet of things and analytics. Big Data and Internet of Things: A Roadmap for Smart Environments 546:169–186
2. Ioana G, Oriana R, Dejan J, Ivan K, Gustavo A (2009) Calling the cloud: enabling mobile phones as interfaces to cloud applications. In: Proc. of the 10th ACM/IFIP/USENIX International Conference on Middleware., pp 83–102
3. Junxian H, Qiang X, Birjodh T, Morley Mao Z, Ming Z, Paramvir B (2010) Anatomizing application performance differences on smartphones. In: Proc. of the 8th international conference on Mobile systems, applications, and services., pp 165–178
4. Yang K, Shumao O, Chen H-H (2008) On effective offloading services for resource-constrained mobile devices running heavier mobile Internet applications. IEEE Communications Magazine 46(1):56–63
5. App drain battery power. www.droidforums.net/threads/battery-drops-40-after-playing-game-for-hour.18301/
6. Paradiso JA, Starner T (2005) Energy scavenging for mobile and wireless electronics. IEEE Pervasive Computing 4(1):18–27
7. Android Application Requirements. www.netmite.com/android/mydroid/development/pdk/docs/system_requirements.html
8. Kumar K, Lu YH (2010) Cloud Computing for Mobile Users. Computer PP (99):1–1
9. Goyal S, Carter J (2004) A lightweight secure cyber foraging infrastructure for resource-constrained devices. Proc. of the Sixth IEEE Workshop on Mobile Computing Systems and Applications. IEEE Computer Society, In, pp 186–195
10. Rajesh B, Jason F, Satyanarayanan M, Shafeeq S, Hen-I Y (2002) The case for cyber foraging. In: Proc. of the 10th workshop on ACM SIGOPS European workshop., pp 87–92
11. Rajesh Krishna B, Darren G, Mahadev S, Herbsleb JD (2007) Simplifying cyber foraging
12. Rajesh Krishna B, Mahadev S, So Young P, Tadashi O (2003) Tactics-based remote execution for mobile computing. In: Proc. of the 1st international conference on Mobile systems, applications and services., pp 273–286
13. Xiaohui G, Nahrstedt K, Messer A, Greenberg I, Milojicic D (2003) Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environments. Proc. of the First IEEE International Conference on Pervasive Computing and Communications, In, pp 107–114
14. Kallonen T, Porras J (2006) Use of distributed resources in mobile environment. Proc. of IEEE International Conference on Software in Telecommunications and Computer Networks, In, pp 281–285
15. Byung-Gon C, Petros M (2009) Augmented smartphone applications through clone cloud execution. In: Proc. of the 12th conference on Hot topics in operating systems., pp 8–8
16. Flinn J, Dushyanth N, Satyanarayanan M (2001) Self-tuned remote execution for pervasive computing. Proc. of the Eighth Workshop on Hot Topics in Operating Systems, In, pp 61–66
17. Alan M, Ira G, Philippe B, Dejan M, Deqing C, Giuli TJ, Xiaohui G (2002) Towards a distributed platform for resource-constrained devices. In: Proc. of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)., pp 43–51
18. Wolbach A, Harkes J, Srinivas C, Satyanarayanan M (2008) Transient customization of mobile computing infrastructure. Proc. of the First Workshop on Virtualization in Mobile Computing, In, pp 37–41
19. Satyanarayanan M, Bahl P, Caceres R, Davies N (2009) The case for VM-based cloudlets in mobile computing. IEEE Pervasive Computing 8(4):14–23
20. Jason F, SoYoung P, Satyanarayanan M (2002) Balancing Performance, Energy, and Quality in Pervasive Computing. In: Proc. of the 22nd International Conference on Distributed Computing Systems., pp 217–226
21. HTC M8St. https://en.wikipedia.org/wiki/HTC_One_(E8)
22. SAMSUNG GT-N7000. https://en.wikipedia.org/wiki/Samsung_Galaxy_Note_(original)
23. PowerTutor. http://ziyang.eecs.umich.edu/projects/powertutor/
24. Kumar K, Liu J, Lu Y-H, Bhargava B (2013) A survey of computation offloading for mobile systems. Mobile Networks & Applications 18(1):129–140
25. Philippsen M, Zenger M (2003) JavaParty-transparent remote objects in Java. Concurrency Practice & Experience 9(11):1225–1242
26. Hunt GC, Scott ML (1999) The Coign automatic distributed partitioning system. In: Proc. of the third symposium on Operating systems design and implementation (OSDI)., pp 187–200
27. Tilevich E, Smaragdakis Y (2009) J-orchestra: enhancing java programs with distribution capabilities. Acm Transactions on Software Engineering & Methodology 19(1):341–352
28. Kemp R, Palmer N, Kielmann T, Bal H (2012) Cuckoo: A Computation Offloading Framework for Smartphones. Mobile Computing, Applications, and Services 76:59–79
29. Eduardo C, Aruna B, Dae-ki C, Alec W, Stefan S, Ranveer C, Paramvir B (2010) MAUI: making smartphones last longer with code offload. In: Proc. of the 8th international conference on Mobile systems, applications, and services., pp 49–62
30. 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: Proc. of the 2012 IEEE International Conference on Computer Communications., pp 945–953
31. Chun B-G, Ihm S, Maniatis P, Naik M (2011) CloneCloud: elastic execution between mobile device and cloud. Proc. of the sixth conference on Computer systems, In, pp 301–314
32. Ying Z, Gang H, Xuanzhe L, Wei Z, Hong M, Shunxiang Y (2012) Refactoring android Java code for on-demand computation offloading. Proc of the ACM international conference on Object oriented programming systems languages and applications 47(10):233–248
33. Lin T-Y, Lin T-A, Hsu C-H, King C-T (2013) Context-aware decision engine for mobile cloud offloading. Wireless Communications and Networking Conference Workshops (WCNCW) 12(5):111–116
34. Ravi A, Peddoju SK (2014) Handoff strategy for improving energy efficiency and cloud service availability for mobile devices. Wireless Personal Communications 81(1):101–132
35. Bowen Z, Amir Vahid D, Calheiros RN, Satish Narayana S, Rajkumar B (2015) A context sensitive offloading scheme for mobile cloud computing service. In: Proc. of the 2015 IEEE 8th International Conference on Cloud Computing., pp 869–876
36. Xu C, Jiao L, Li W, Xiaoming F (2015) Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Transactions on Networking 24(5):2795–2808
37. Xu C (2015) Decentralized computation offloading game for mobile cloud computing. IEEE Transactions on Parallel and Distributed Systems 26(4):974–983